# Fine-tuning Deep Reinforcement Learning Policies with r-STDP for Domain Adaptation

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

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

Deniz Ergene
deniz.ergene@tum.de
Technical University Munich
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

Using deep reinforcement learning policies that are trained in simulation on real robotic platforms requires fine-tuning due to discrepancies between simulated and real environments. Multiple methods like domain randomization and system identification have been suggested to overcome this problem. However, sim-to-real transfer remains an open problem in robotics and deep reinforcement learning. In this paper, we present a spiking neural network (SNN) alternative for dealing with the sim-to-real problem. In particular, we train SNNs with backpropagation using surrogate gradients and the (Deep Q-Network) DQN algorithm to solve two classical control reinforcement learning tasks. The performance of the trained DQNs degrades when evaluated on randomized versions of the environments used during training. To compensate for the drop in performance, we apply the biologically plausible reward-modulated spike timing dependent plasticity (r-STDP) learning rule. Our results show that r-STDP can be successfully utilized to restore the network's ability to solve the task. Furthermore, since r-STDP can be directly implemented on neuromorphic hardware, we believe it provides a promising neuromorphic solution to the sim-to-real problem.

## CCS CONCEPTS

• **Computer systems organization** → Robotics; • **Computing methodologies** → **Reinforcement learning**.

## KEYWORDS

reinforcement learning, neural networks, spiking neural networks

**ACM Reference Format:**
Mahmoud Akl, Yulia Sandamirskaya, Deniz Ergene, Florian Walter, and Alois Knoll. 2022. Fine-tuning Deep Reinforcement Learning Policies with r-STDP for Domain Adaptation. In *International Conference on Neuromorphic Systems (ICONS 2022), July 27–29, 2022, Knoxville, TN, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3546790.3546804

## 1 INTRODUCTION

Deep Reinforcement Learning (DRL) has been gaining a lot of traction for learning control policies in the field of robotics that allows agents to solve complex tasks. It combines advancements from Deep Learning with ideas on learning from rewards developed in the field of Reinforcement Learning (RL) [36]. Over the past few years, multiple works demonstrated how DRL algorithms can be utilized to solve object manipulation [33, 50], navigation [42, 49], and locomotion tasks [13, 20].

While training DRL agents on real robotic platforms has been explored [21], using simulators to train agents is the preferred method as they offer multiple advantages over training on real robots. For example, simulators can run faster than real time, can be parallelized given enough computational resources, and are safer and more cost effective [12]. To that end, there has been an increasing number of robot learning simulators and frameworks connecting different physics simulators and RL frameworks in the past few years [7, 9, 10, 24].

However, transferring a policy that was trained in simulation to the real world is usually challenging due to mismatches between simulated and real environments, also known as the simulation-to-reality gap [46]. Differences between simulation and reality manifest in actuation due to inaccuracies of the physical parameters of the models, and in sensing due to differences between real and rendered images for example. To overcome this gap, multiple methods have been suggested [48]. Among the suggested methods is domain adaptation, which is a sub-domain of transfer learning, and has been applied to robotic sim-to-real transfer scenarios [4, 41].

In this work, we consider a spiking neural network (SNN) and reward-modulated synaptic time-dependent plasticity (r-STDP) for domain adaptation. SNNs have been studied as a possible energy efficient and biologically plausible alternative to artificial neural networks (ANNs) [23]. Recently, gradient based learning in SNNs was made possible through the introduction of gradient approximation techniques [15, 28, 34]. Since then, the information processing performance of SNNs is constantly improving. In the supervised learning domain, for example, SNNs almost match ANNs image classification performance measured on popular benchmark datasets

*ICONS 2022, July 27–29, 2022, Knoxville, TN, USA*
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9789-6/22/07…$15.00
https://doi.org/10.1145/3546790.3546804

[19, 43, 47]. Also in the reinforcement learning domain, it was shown that SNNs can be directly trained with backpropagation and state-of-the-art DRL algorithms to solve robot navigation tasks and OpenAI Gym benchmark environments [1, 38, 39], while other works focused on converting ANN DRL policies to SNNs [30, 37]. In a previous work [1], we trained quantized SNNs with the DQN algorithm to solve the CartPole-v0 and Acrobot-v1 environments from OpenAI gym, and were able to port them on Intel's neuromorphic research chip Loihi [8] without loss in performance. We noticed, however, that the networks' performance degrades when evaluated with modified environment parameters. Here, we expand on our previous work by utilizing local onchip plasticity mechanism r-STDP to adapt the backpropagation-trained networks to environments with parameter variations.

Our study shows that r-STDP fine-tuning of the weights acquired through training by backpropagation improves the network's performance when subjected to perturbations. Since using SNNs for RL tasks and robotics is gaining increasing popularity, the need for a neuromorphic sim-to-real solution is inevitable. We believe that online three-factor learning rules like r-STDP implemented on neuromorphic hardware can be used to address the transfer of SNN policies trained in simulation to real robots.

## 2 METHODS

In this section, we describe the overall experimental setup, the details of the environments we ran the experiments on, the training details of the spiking DQNs, and the r-STDP formulation used in our experiments.

### 2.1 Experimental Setup

We conducted our experiments on two classical control environments from OpenAI Gym [5]: CartPole-v0 and Acrobot-v1. These environments were chosen because of their discrete action spaces, which are required for the SNN-based implementation of the DQN algorithm [26] that we consider in this paper. Unlike vision-based tasks, they moreover do not require complex processing of the input signals. After training, we evaluate the Deep Spiking Q-Networks (DSQNs) on 100 randomly initialized environments with the same environment's parameters used during training. This evaluation measures the trained networks' performance and serves as a baseline for future experiments on randomized environments.

We then measure the DSQN's performance on randomized environments with the same initialization as before, to be able to compare the results. We created two types of random environments, one where only one physical parameter is modified (e.g. changing the pole's length in CartPole), and one where multiple physical parameters are modified simultaneously (e.g. changing the pole's length, the pole's mass and the force magnitude in CartPole). When modifying multiple parameters simultaneously, we sample the parameters' values from a uniform distribution. Inspired by a previous work on generalization in RL [29], we defined two ranges: one for "random normal" and one for "random extreme" environments. The details of the random environments are listed in Table 2.

When measuring the DSQNs' performance on randomized environments, we notice a degradation of performance which we discussed in our previous work [1]. In order to adapt the network to the new environment, we apply r-STDP to the synapses connecting hidden layers one and two, as shown in Figure 1.

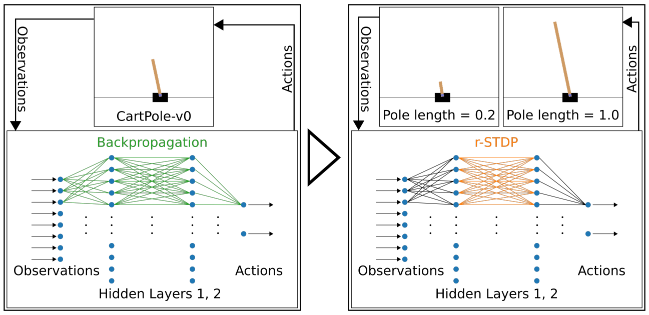


**Figure 1: Overview of the experimental setup. SNNs are trained with backpropagation to solve OpenAI Gym environments. After training, r-STDP is applied to the connections between hidden layers 1 and 2 to adapt the network to changes in the environment.**

### 2.2 Environments

Here we provide a brief overview of the environments used in our experiments, highlighting the observations, actions and rewards for each environment. Additionally, we explain which environment parameters are randomized. The default values for those parameters as well as the ranges for random normal and random extreme environments are shown in Table 2.

*2.2.1 CartPole.* The CartPole problem is a classic problem in the reinforcement learning literature [2] and has been used in previous works on generalization in RL [44]. It consists of an un-actuated rigid pole hinged to a cart which can move left and right on a one-dimensional track. The pole is free to move only in the plane vertical to the cart and track, and the task is to keep it 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. The maximum possible reward an agent can achieve is 200. An episode is terminated when the pole falls beyond 15 degrees from the vertical, or when the maximum time step 200 is reached. Three environment parameters can be modified: the pole's length, the pole's mass, and the force magnitude.

*2.2.2 Acrobot.* The Acrobot problem is also a classic reinforcement learning problem [35]. 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 $\theta_1$ and $\theta_2$ 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. An episode is terminated when the lower link reaches the required height, or when the maximum time step 500 is reached. Acrobot has no defined maximum achievable reward, since how fast swinging the link up to the target height depends on the initial configuration. Three environment parameters can be modified: the links' length, the links' mass and the links' moment of inertia. In the default environment, both links share the same parameters, and we kept that constraint for the randomized environments.

### 2.3 Training SNNs with the DQN algorithm

To train the networks in the experiments described in this paper, we used the surrogate gradient approach to utilize the backpropagation algorithm. All networks were trained with the SpyTorch framework [45], in which the neuronal dynamics are described by:

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

where $v_i(t)$ is the membrane potential at time $t$, $\beta \in [0, 1]$ is the membrane potential decay factor, and $u_i(t)$ is the input current at time $t$, in turn described by the equation:

$$u_i(t) = \alpha u_i(t-1) + \sum_j w_{ij} s_j(t), \tag{2}$$

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 $j$.

As in our previous work [1], we inject the weighted sum of the real-valued observations as current in the first hidden layer. However, we apply a two-neuron transformation to the observations before calculating the weighted sum, in which each observation value is assigned two mutually exclusive input neurons, one representing positive, and the other representing negative values. The two-neuron encoding method was first introduced in [31], and we found that it has stabilizing effects on surrogate gradient based training.

To read out the Q-values, we removed the spiking mechanism from the output layer, i.e. incoming spikes depolarize the output neurons, but the threshold is set to infinity so that the neurons never spike. We then read the membrane potential values of the output neurons as the Q-values. After that the network state is reset, i.e. all membrane potentials and synaptic currents are set to zero before feeding in a new observation.

### 2.4 r-STDP

STDP is a Hebbian-like learning rule [6]. The strength of synapses is adapted according to the timing of pre- and postsynaptic spikes:

$$\begin{gathered}\Delta t = t_i - t_j, \\ W(\Delta t) = \begin{cases} A^+ \, e^{(-\frac{\Delta t}{\tau^+})}, & \text{if } \Delta t > 0 \\ -A^- \, e^{(\frac{\Delta t}{\tau^-})}, & \text{if } \Delta t < 0 \end{cases} \\ \Delta w_{ij} = \sum_{t_i} \sum_{t_j} W(\Delta t),\end{gathered} \tag{3}$$

where $t_j$ and $t_i$ are the times of pre- and postsynaptic spikes, $A^+$ and $A^-$ are constants for potentiation and depression, and $\tau^+$ and $\tau^-$ are the time constants for the STDP update windows. The change in the synaptic weight $\Delta w_{ij}$ is calculated as the sum of function $W(\Delta t)$ over all pre- and postsynaptic spikes.

While STDP correlates spike times in short time frames, it does not account for the association of events over longer timescales. However, this is important for many use-cases, e.g. distal reward problems. One approach to tackle this problem is r-STDP [11, 16]. It is a three-factor learning rule, i.e. besides two hebbian factors (pre- and postsynaptic spike-times), it uses a third factor to trigger changes in synaptic strength [18]. In r-STDP, the co-activation of pre- and postsynaptic neurons modifies an eligibility trace. This trace is multiplied by a third signal, the reward, to calculate the change in synaptic strength at a later timestep.

Therefore, two traces $k^+_{ij}$ and $k^-_{ij}$ are computed at each timestep. They encode the amount and recency of pre- and postsynaptic spikes, respectively:

$$k^+_{ij}(t+1) = k^+_{ij}(t) \, e^{\left(-\frac{1}{\tau^+}\right)} + s_j(t) \tag{4}$$

$$k^-_{ij}(t+1) = k^-_{ij}(t) \, e^{\left(-\frac{1}{\tau^-}\right)} + s_i(t), \tag{5}$$

The values $k^+_{ij}(t)$ and $k^-_{ij}(t)$ are used to modify the eligibility trace:

$$\begin{gathered}\Delta c_{ij}(t) = -\frac{c_{ij}(t)}{\tau^{\mathrm{c}}} + (A^+ \, k^+_{ij}(t) \, s_i(t) - A^- \, k^-_{ij}(t) \, s_j(t)) \, C \\ c_{ij}(t+1) = c_{ij}(t) + \Delta c_{ij}(t),\end{gathered} \tag{6}$$

where $c_{ij}(t)$ denotes the value of the eligibility trace at timestep $t$, $\tau^{\mathrm{c}}$ is the time constant for the trace's decay, and $C$ is a constant. Note that presynaptic spikes increment $k^+_{ij}$, but $k^+_{ij}$ is added to $c_{ij}(t)$ upon the occurrence of postsynaptic spikes. The inverse is true for $k^-_{ij}$. Furthermore, even though we reset the network state after every inference step, we maintain the value of the eligibility trace (see Figure 2). This way, we can use the value of the eligibility trace when an episode terminates to update the weights. The change in synaptic strength is then calculated as:

$$\Delta w_{ij} = r(t) \cdot c_{ij}(t), \tag{7}$$

where $r(t)$ is the value of the reward signal at timestep $t$.

To apply r-STDP, we evaluate the DSQN models trained on CartPole-v0 and Acrobot-v1 on a randomized version of the environment. Upon episode termination, weight updates are applied according to equation 7. For that, we defined custom reward functions based on each environment's default reward such that the magnitude of the weight updates is inversely proportional to the model's performance. We defined the reward function for CartPole-v0 to be:

$$r_{cartpole} = 1 - \frac{reward}{max\_reward} \tag{8}$$

where *max_reward* is the maximum possible reward and is equal to 200. This reward formulation for CartPole-v0 restricts weight updates when the model performs well. For Acrobot-v1, we had to define the reward function differently since the maximum possible reward is not defined. Only the minimum possible reward is known, which is the maximum number of time steps in an episode and is set to 500. We defined the reward function for Acrobot-v1 to be:

$$r_{acrobot} = \frac{reward - 50}{min\_reward} \tag{9}$$

This reward definition for Acrobot-v1 also ensures that good-performing agents will have negligible weight updates. Additional non-linear modifications of the reward were not considered in order to stay close to the original rewards returned by the environments.

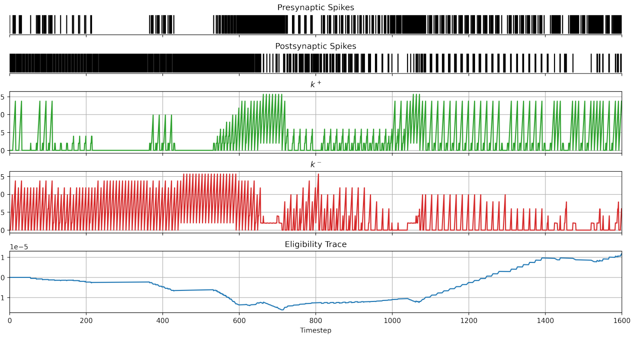


**Figure 2: Evolution of the eligibility trace in one synaptic connection during one CartPole-v0 episode. Horizontal axis: the number of timesteps of the episode (200) multiplied by the network simulation time for each observation (8). The eligibility trace is persistent, even though the network state (membrane potentials, synaptic currents, and spikes) is reset every 8 timesteps.**

**Table 1: SNN parameters. Membrane parameters used during DQN training (top), and r-STDP parameters used during fine-tuning (bottom).**

| Hyperparameter | Cartpole-v0 | Acrobot-v1 |
|---|---|---|
| $\alpha$ | 0.8 | 0.8 |
| $\beta$ | 0.8 | 0.8 |
| threshold | 0.5 | 1.0 |
| simulation time | 8 | 3 |
| network architecture | [8, 64, 64, 2] | [12, 256, 256 ,3] |
| $A^+$ | $\overline{w_{>0}}$ | $\overline{w_{>0}}$ |
| $A^-$ | $\lvert\overline{w_{<0}}\rvert$ | $\lvert\overline{w_{<0}}\rvert$ |
| $\tau^{+/-}$ | 5 | 30 |
| $\tau^c$ | 10 | 10 |
| C | 0.01 | 0.05 |

## 3 EXPERIMENTS AND RESULTS

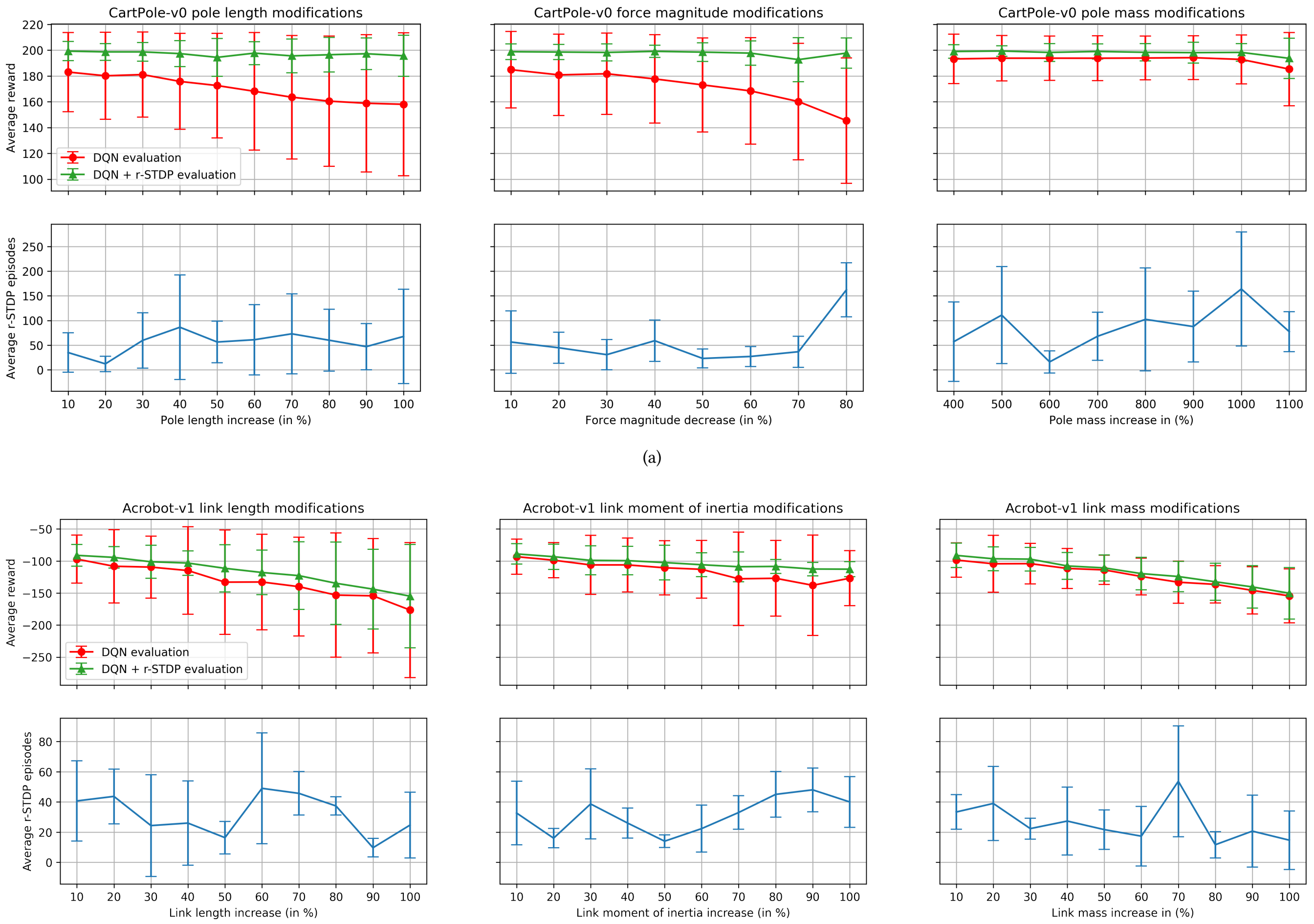
For each environment we trained three spiking DSQN models with the methods described in 2.3. The membrane parameters used for both experiments are listed in Table 1. We used slightly different membrane parameters than in our previous work [1], as we found that those parameters lead to faster and more stable learning. After training, we conducted two types of experiments on random environments. We fixed 100 random seeds for the test environments to able to conduct all experiments on the same initial conditions.

In a first set of experiments, we modified only one parameter in the environment at a time. When modifying the value of an environment's parameter, we can either increase or decrease its value. In our experiments, we chose to increase all the parameters' values, except for the force magnitude in CartPole. We observed that DSQNs perform worse when the force magnitude is decreased than when it is increased, which makes restoring the performance with r-STDP more challenging.

Overall, we observed that the DSQN performance is inversely proportional to the degree of randomness in the environment (see Figure 3). However, we chose the range of randomness for each parameter separately, as we observed that varying each parameter has different effects on the performance of the DSQN. For example, when increasing the pole's length in CartPole, we chose the range [10% - 100%]. However, applying the same range to the pole's mass did only have negligible effects on the network's performance. This is why we defined the range for the pole's mass increase to be [500% - 1100%]. Furthermore, we found that there is an upper limit for randomness, beyond which the DSQN performance collapses. For example, increasing the pole's mass in CartPole beyond 1100% makes the pole fall right away and renders the problem no longer solvable. The same behavior was observed when decreasing the force magnitude. The DSQN looses more than 60% of its performance when decreasing the force magnitude below 80%. This is also why we defined the range for the decrease in force magnitude to be [10% - 80%].

To compensate for the drop in performance, we apply r-STDP to the synapses connecting neurons in hidden layers one and two with the methods described in 2.4. We use a different set of random seeds during r-STDP training. To apply r-STDP, we evaluate the DSQN on the randomized environment for one episode, and only apply weight changes after an episode ends. The magnitude of the weight change depends on the value of the eligibility trace at the end of the episode (see Figure 2), as well as on the value of the reward achieved (see equations 8 and 9), and is applied according to equation 7. After each r-STDP training episode, we measure the performance of the new weights on the initial 100 randomized environments used for evaluation. We repeat this process for 250 episodes and keep the weights that achieved the highest reward on the evaluation environments. This is analogous to measuring accuracy on a validation set after completing an epoch in supervised learning, and performing early stopping [32]. The parameters we used for r-STDP are listed in Table 1. Since we are conducting the experiments using three DSQN models, we set the values for $A^+$ and $A^-$ for each DSQN model separately as we saw variations in mean weight values for the trained DSQNs. This allows the r-STDP induced weight change to be proportional to the existing weight values acquired through backpropagation. We set $A^+$ to the mean value of the positive weights, and $A^-$ to the absolute value of the mean of the negative weights.

The top panels in Figure 3 show the difference in performance before and after applying r-STDP on randomized environments. The CartPole results show that the drop in performance can be restored for each parameter within the specified range of randomness. The Acrobot results show that r-STDP always improves the DSQN performance, yet not fully restore the DSQN performance to that on the not modified environments. The performance on non-randomized environments can be seen on the blue bars in Figure 4. The Acrobot problem is inherently different than CartPole. In CartPole, the maximum reward (200) is known and achievable after modifying the parameters to a certain extent. Acrobot, on the other hand, does not have an established maximum possible reward and the maximal reward heavily depends on the initial configuration. Additionally, modifying the environment's parameters affects the number of time steps required to lift the lower link to the target height. To highlight the difference parameter modification has on



(a)

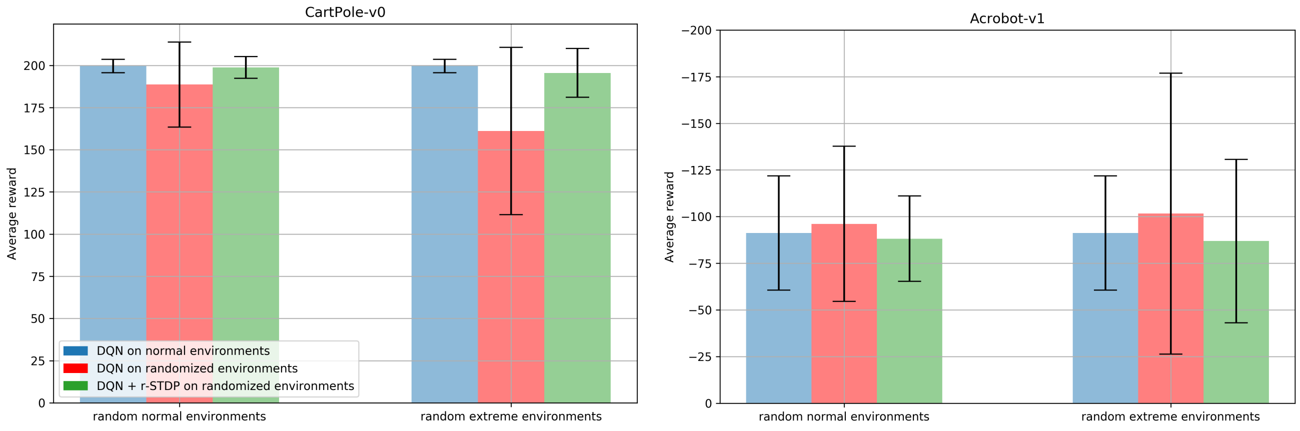
(b)

**Figure 3: r-STDP results on single parameter variation for CartPole (a) and Acrobot (b). Top panels show the performance of the trained DSQN on the randomized environment (in red) vs. the performance of the DSQN after applying r-STDP (in green) measured over three different DQN models. The dots indicate the mean value of the rewards measured for three models, each on 100 randomly initialized environments with fixed random seeds. Bottom panels show the average number of r-STDP episodes required to reach the best reward. All error bars represent the standard deviation.**

each environment, we trained DSQNs to solve CartPole-v0 and Acrobot-v1 with 50% and 100% pole and link lengths increase. The training curves can be seen in Figure 5. They show that DSQNs achieve the same reward on CartPole-v0 with different pole lengths, while the same parameter modifications in Acrobot-v1 yield different reward values. This distinction between both environments is important when interpreting the results.

The lower panels in Figure 3 show the average number of episodes required to reach the best r-STDP result. In some experiments we see that the number of episodes decreases as we increase the degree of randomness. This is because for smaller degrees of randomness, the DSQN's drop in performance is not considerable. This means that during r-STDP training, the agent will perform well on many episodes, which will result in either no or negligible weight updates. Consequently, r-STDP training might require more episodes, until it encounters an episode in which the agent does not perform well.

In a second set of experiments, we modified all the environments' physical parameters at once. The values for the random parameters were sampled uniformly from the ranges defined in Table 2. The classification of "random normal" and "random extreme" environments, as well as the range of values is inspired by a previous work on generalization in RL [29]. We sampled 10 sets of random normal and 10 sets of random extreme values for each environment. We then measured the performance of the DSQN on 100 randomized environments for each random setting using the same fixed random seeds, and applied r-STDP to compensate for the drop in performance. Figure 4 shows the results of the multiple parameter randomization experiment. For the random normal CartPole



(a) (b)

**Figure 4: r-STDP results on multiple parameter variation for CartPole-v0 and Acrobot-v1. The results are averaged over 3000 runs (3 DQN models x 10 random environments x 100 environment initializations). The values for the parameters in the random normal and random extreme environments were drawn from a uniform distribution, the details of which are specified in Table 2. All error bars represent the standard deviation.**

**Table 2: Random normal (R) and random extreme (E) environment parameter ranges, and default parameter values (D) for CartPole (top) and Acrobot (bottom)**

| Parameter | D | R | E |
|---|---|---|---|
| Force | 10 | [5, 15] | [1, 5] ∪ [15, 20] |
| Length | 0.5 | [0.25, 0.75] | [0.05, 0.25] ∪ [0.75, 1.0] |
| Mass | 0.1 | [0.05, 0.5] | [0.01, 0.05] ∪ [0.5, 1.0] |
| Length | 1 | [0.75, 1.25] | [0.5, 0.75] ∪ [1.25, 1.5] |
| Mass | 1 | [0.75, 1.25] | [0.5, 0.75] ∪ [1.25, 1.5] |
| MOI | 1 | [0.75, 1.25] | [0.5, 0.75] ∪ [1.25, 1.5] |

**Table 3: Significance values of multiple parameter variation experiments. P-values are calculated based on the dependent t-test for paired samples.**

| Environment | Random Normal | Random Extreme |
|---|---|---|
| CartPole-v0 | $t$=-22.16, $p$<.001 | $t$=-35.71, $p$<.001 |
| Acrobot-v1 | $t$=-10.13, $p$<.001 | $t$=-11.97, $p$<.001 |

experiment, the drop in performance was 5.47% and was reduced to 0.401% after applying r-STDP. In accordance with the previous results, the performance drop in the random extreme environments was higher and reached 19.28% and was reduced to 2.05% after applying r-STDP. Similarly, the random normal Acrobot experiment showed 5.34% drop in performance. After applying r-STDP, the reward received on the 100 evaluation environments surpassed that of the normal, non-randomized environments (-88.2 vs. -91.22). The random extreme acrobot environments showed a 11.4% drop in performance. After applying r-STDP, the evaluation on the randomized environments also surpassed that on the normal environments (-86.90 vs. -91.22).

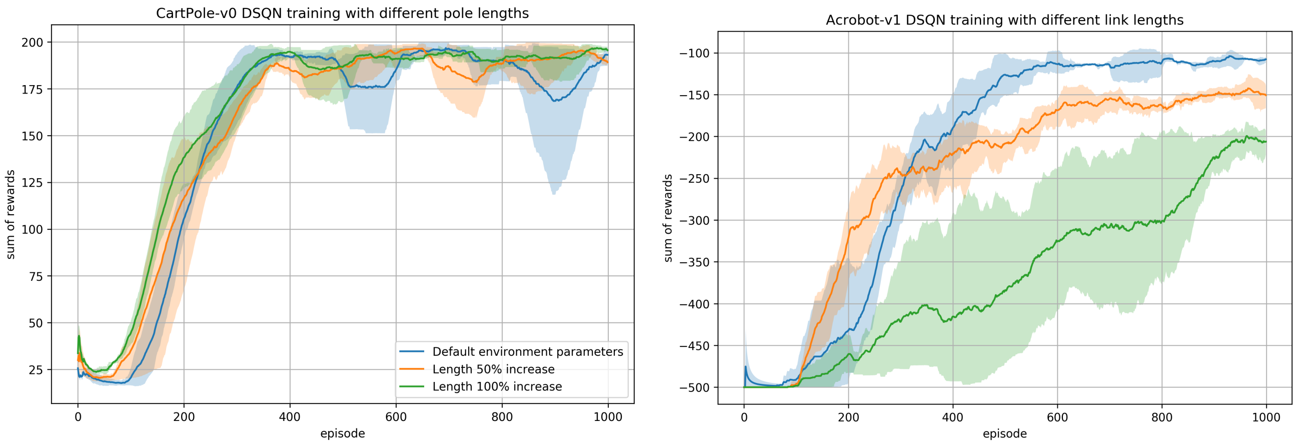
Unlike the single parameter modification experiments, random normal and random extreme environments end up with a combination of modified parameters. Some of which with an increased while others with a decreased value. Similar to the Acrobot-v1 training results shown in Figure 5 that show how increasing the link lengths results in lower rewards, decreasing the values of some parameters has the opposite effect. For example, decreasing the value of the links' length or mass will result in higher rewards than that on normal environments. Therefore, surpassing the normal environment reward after r-STDP fine tuning in the Acrobot-v1 problem is possible, depending on the sampled random environment parameters.

Following the recommendations from [14], we ran a statistical analysis on the results of the multiple parameter variation experiments. But we used the dependent t-test for paired samples since it better suits our type of experiments. The p-values for cartpole and acrobot on the random normal and random extreme environments are listed in table 3.

## 4 DISCUSSION AND CONCLUSION

Local learning rules like r-STDP have been applied in previous works to solve reinforcement learning [3, 22, 40] as well as supervised learning tasks [17, 27]. In this work, however, we propose r-STDP as a fine-tuning method for adapting DRL policies that were trained with backpropagation for environment variations. The method presented in this paper can be applied to directly trained SNNs, or to SNNs that were converted from ANNs. Additionally, this method is agnostic to the DRL algorithm used during training, and should be applicable to continuous control problems as well.

Our results on experiments with single parameter as well as multiple parameter variations reveal that r-STDP can help correct the behavior of the trained model after parameter change, depending on



(a)

(b)

**Figure 5: DSQN training curves with different pole and link lengths for Cartpole-v0 (a) and Acrobot-v1 (b), respectively. DSQNs achieve the same reward on CartPole-v0 with different pole lengths, while modifying the links' lengths in Acrobot-v1 yields different reward values. Solid lines are smoothed rewards (window size of 100 episodes) averaged over three runs with random initialization seeds. Shaded areas show the standard deviation.**

the nature of dependence of the reward function on the parameter change. If this dependence is monotonic, r-STDP in a single layer restores the behavior to that of the non-modified environment. If this dependence is more complex, it can only partially improve the behavior. Our chosen encoding and decoding methods restricted the application of r-STDP to the synapses connecting hidden layers one and two, since our input and output layers do not produce any spiking activity (see Figure 1). Even though applying r-STDP to those synaptic connections was sufficient to improve the network's performance on randomized environments, we believe that extending the application of r-STDP to other layers might improve the sampling efficiency. Further studies on what kind of local learning rule (and in how many layers) can deal with which complexity of the environment change and reward function dependence are required. Furthermore, we observed that fine-tuning with r-STDP is sensitive to the hyperparameters like eligibility trace time constant, amplitudes of weight change for facilitation and depression, and the time constant of the STDP time window. We found that the r-STDP parameters listed in Table 1 worked well for two different models with different membrane parameters. However, we conclude that an extensive hyperparameter search on the STDP parameters as well as the investigation of the dependency of the r-STDP parameters on the membrane parameters (for example a higher SNN simulation time might require larger time constants) could lead to better sample efficiency in neuromorphic fine-tuning of DRL policies.

Since such three-factor learning rules are available on neuromorphic devices [8, 25], we believe that combining backpropagation-based offline training with r-STDP online fine-tuning can provide a promising research direction for neuromorphic sim-to-real transfer, and could potentially open the door for more neuromorphic robotic applications.

## ACKNOWLEDGMENTS

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 945539 (Human Brain Project SGA3).

## REFERENCES

[1] 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 2021* (Knoxville, TN, USA) *(ICONS 2021)*. Association for Computing Machinery, New York, NY, USA, Article 13, 7 pages. https://doi.org/10.1145/3477145.3477159

[2] 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

[3] 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. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 4725–4732. https://doi.org/10.1109/ICRA.2018.8460482

[4] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. 2018. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 4243–4250. https://doi.org/10.1109/ICRA.2018.8460875

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[6] Natalia Caporale and Yang Dan. 2008. Spike Timing–Dependent Plasticity: A Hebbian Learning Rule. *Annual Review of Neuroscience* 31, 1 (July 2008), 25–46. https://doi.org/10.1146/annurev.neuro.31.060407.125639

[7] Erwin Coumans and Yunfei Bai. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

[8] 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

[9] Egidio Falotico, Lorenzo Vannucci, Alessandro Ambrosano, Ugo Albanese, Stefan Ulbrich, Juan Camilo Vasquez Tieck, Georg Hinkel, Jacques Kaiser, Igor Peric, Oliver Denninger, et al. 2017. Connecting artificial brains to robots in a comprehensive simulation framework: the neurorobotics platform. *Frontiers in Neurorobotics* 11 (2017), 2. https://doi.org/10.3389/fnbot.2017.00002

[10] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. 2018. SURREAL: Open-Source Reinforcement Learning Framework and Robot Manipulation Benchmark. In *Proceedings of The 2nd Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 87)*. PMLR, 767–782.

[11] Răzvan V. Florian. 2007. Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity. *Neural Computation* 19, 6 (June 2007), 1468–1502. https://doi.org/10.1162/neco.2007.19.6.1468 Conference Name: Neural Computation.

[12] Javier García, Fern, and o Fernández. 2015. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research* 16, 42 (2015), 1437–1480. http://jmlr.org/papers/v16/garcia15a.html

[13] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. 2019. Learning to Walk Via Deep Reinforcement Learning. https://doi.org/10.15607/RSS.2019.XV.011

[14] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep Reinforcement Learning That Matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence* (New Orleans, Louisiana, USA) *(AAAI'18/IAAI'18/EAAI'18)*. AAAI Press, Article 392, 8 pages.

[15] Dongsung Huh and Terrence J Sejnowski. 2018. Gradient Descent for Spiking Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 31. Curran Associates, Inc.

[16] Eugene M. Izhikevich. 2007. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex (New York, N.Y.: 1991)* 17, 10 (Oct. 2007), 2443–2452. https://doi.org/10.1093/cercor/bhl152

[17] Saeed Reza Kheradpisheh, 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.

[18] Łukasz Kuśmierz, Takuya Isomura, and Taro Toyoizumi. 2017. Learning with three factors: modulating Hebbian plasticity with errors. *Current Opinion in Neurobiology* 46 (Oct. 2017), 170–177. https://doi.org/10.1016/j.conb.2017.08.020

[19] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. 2020. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in neuroscience* (2020), 119.

[20] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. 2020. Learning quadrupedal locomotion over challenging terrain. *Science robotics* 5, 47 (2020), eabc5986.

[21] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. 2018. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research* 37, 4-5 (2018), 421–436.

[22] Hao Lu, Junxiu Liu, Yuling Luo, Yifan Hua, Senhui Qiu, and Yongchuang Huang. 2021. An autonomous learning mobile robot using biological reward modulate STDP. *Neurocomputing* 458 (2021), 308–318.

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

[24] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. 2021. Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[25] Mantas Mikaitis, Garibaldi Pineda García, James C Knight, and Steve B Furber. 2018. Neuromodulated synaptic plasticity on the SpiNNaker neuromorphic system. *Frontiers in neuroscience* 12 (2018), 105.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[27] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J Thorpe, and Timothée Masquelier. 2019. Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern recognition* 94 (2019), 87–95.

[28] 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.

[29] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. 2018. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282* (2018).

[30] 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.

[31] José Antonio Pérez-Carrasco, Bo Zhao, Carmen Serrano, Begona Acha, Teresa Serrano-Gotarredona, Shouchun Chen, and Bernabé Linares-Barranco. 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 (2013), 2706–2719.

[32] Lutz Prechelt. 1998. Early stopping-but when? In *Neural Networks: Tricks of the trade*. Springer, 55–69.

[33] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. 2018. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania. https://doi.org/10.15607/RSS.2018.XIV.049

[34] Sumit B Shrestha and Garrick Orchard. 2018. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems* 31 (2018).

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

[36] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[37] Weihao Tan, Robert Kozma, and Devdhar Patel. 2022. Optimization methods for improved efficiency and performance of Deep Q-Networks upon conversion to neuromorphic population platforms. *Knowledge-Based Systems* (2022), 108257.

[38] 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

[39] Guangzhi Tang, Neelesh Kumar, Raymond Yoo, and Konstantinos Michmizos. 2021. Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control. In *Conference on Robot Learning*. PMLR, 2016–2029.

[40] J Camilo Vasquez Tieck, Pascal Becker, Jacques Kaiser, Igor Peric, Mahmoud Akl, Daniel Reichard, Arne Roennau, and Rüdiger Dillmann. 2019. Learning target reaching motions with a robotic arm using brain-inspired dopamine modulated STDP. In *2019 IEEE 18th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE, 54–61.

[41] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. 2020. *Adapting Deep Visuomotor Representations with Weak Pairwise Constraints*. Springer International Publishing, Cham, 688–703. https://doi.org/10.1007/978-3-030-43089-4_44

[42] Chao Wang, Jian Wang, Yuan Shen, and Xudong Zhang. 2019. Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach. *IEEE Transactions on Vehicular Technology* 68, 3 (2019), 2124–2136. https://doi.org/10.1109/TVT.2018.2890773

[43] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. 2018. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience* 12 (2018), 331.

[44] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. 2017. Preparing for the Unknown: Learning a Universal Policy with Online System Identification. In *Proceedings of Robotics: Science and Systems*. Cambridge, Massachusetts. https://doi.org/10.15607/RSS.2017.XIII.048

[45] Friedemann Zenke. 2019. *SpyTorch*. https://doi.org/10.5281/zenodo.3724018

[46] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. 2015. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791* (2015).

[47] Wenrui Zhang and Peng Li. 2019. Spike-train level backpropagation for training deep recurrent spiking neural networks. *Advances in neural information processing systems* 32 (2019).

[48] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. 2020. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 737–744.

[49] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 3357–3364. https://doi.org/10.1109/ICRA.2017.7989381

[50] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, JÃ¡nos KramÃ¡r, Raia Hadsell, Nando de Freitas, and Nicolas Heess. 2018. Reinforcement and Imitation Learning for Diverse Visuomotor Skills. In *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania. https://doi.org/10.15607/RSS.2018.XIV.009