IDIAP RESEARCH REPORT

AN ATTENTION MECHANISM FOR DEEP Q-NETWORKS WITH APPLICATIONS IN ROBOTIC PUSHING

Marco Ewerton[a] Sylvain Calinon
Jean-Marc Odobez

Idiap-RR-03-2021

APRIL 2021

[a]Idiap

Centre du Parc, Centre du Parc, Rue Marconi 19, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

An Attention Mechanism for Deep Q-Networks with Applications in Robotic Pushing

Marco Ewerton, Sylvain Calinon, Jean-Marc Odobez

Abstract— **Humans effortlessly solve push tasks in everyday life but unlocking these capabilities remains a research challenge in robotics. Physical models are often inaccurate or unattainable. State-of-the-art data-driven approaches learn to compensate for these inaccuracies or get rid of the approximated physical models altogether. Nevertheless, data-driven approaches such as Deep Q-Networks (DQNs) get frequently stuck in local optima in large state-action spaces. We propose an attention mechanism for DQNs to improve their sampling efficiency and demonstrate in simulation experiments with a UR5 robot arm that such a mechanism helps the DQN learn faster and achieve higher performance in a push task involving objects with unknown dynamics.**

I. Introduction and Related Work

Pushing is a motion that is important for moving objects that are too large, too heavy, too cluttered or too distant to be grasped. Thus, several types of pushes can be involved in many manipulation tasks, to isolate object, reorient them, bring them closer, put them into a container, or even aid perception. However, deciding whether to push or not, which push to perform and how to do it remains a challenging task.

Related work and motivations. For doing pushing, robots often rely on physical models. However, these models are computable but are only approximations of physical phenomena. For instance, Yu et al. [1] present a dataset of planar pushing experiments to study how reliable these models are. The dataset can also be used for benchmarking motion prediction methods and for learning. It shows that pushing can be seen as a stochastic process even if a highly precise manipulator (accuracy of pusher position $= 0.1$ mm) is used to perform the pushes, highlighting the importance of learning to tackle such tasks.

Zeng et al. [2] use model-free deep reinforcement learning to synergistically perform pushing and grasping. Their learning architecture is comprised of two fully convolutional networks. One network maps RGB-D images to the utility of pushing 10 cm to the right at each pixel. The other maps RGB-D images to the utility of grasping at each pixel. By presenting these networks with the input RGB-D image rotated at 16 different angles, the networks can compute the utilities of pushes and grasps with different orientations. The heights of the pushes and grasps depend on the D channel. More specifically, Q-learning is used to train the networks. A reward of 1 is achieved when a grasp is successful and a reward of 0.5 when pushes produce detectable changes to the environment. Otherwise, the reward is 0.

The authors are with the Idiap Research Institute, CH-1920, Martigny, Switzerland {marco.ewerton, sylvain.calinon, jean-marc.odobez}@idiap.ch

Continuing to explore synergies between different kinds of movements, [3] shows a method to learn grasping and throwing of arbitrary objects. In that work, a perception module computes a feature representation of an RGB-D image, and a simple physics model is used to estimate the release velocity necessary to throw an object at a certain target position. The estimated release velocity is concatenated with a feature representation and fed into the grasping module and the throwing module. The grasping module computes the utility of a grasp at each pixel. The throwing module computes, for each possible grasp, the residual on top of the estimated velocity to account for phenomena not accounted for by the simple physics model like centripetal accelerations and aerodynamic drag. The perception, grasping and throwing modules are fully convolutional networks. The success of grasps is quantified based on the end distance between the parallel grippers or based on whether or not the thrown object landed in the correct box.

The methods proposed in [2] and [3] succeed in learning to push, grasp and throw objects in part due to the discretization of the action space, which simplifies to the some extent the reinforcement learning problem. In contrast, [4] uses Logic-Geometric Programming [5] and Multi-Bound Tree Search [6] to find and optimize a sequence of continuous actions to solve several tasks involving the usage of tools and the manipulation of different objects. Notwithstanding its success in finding sequences of continuous actions, a number of assumptions have been made in this work to make this optimization problem numerically solvable. It has been assumed, for example, that each object has a sphere-swept convex geometry, that its dynamics are known and that there are no uncertainties. Future work will need to make less assumptions, deal with unknown dynamics and uncertainty to succeed in real-world applications.

Hogan et al. [7] address the problem of pushing an object on a plane along a desired trajectory or to pass through a sequence of via points. The proposed method can deal with disturbances such as the ones introduced by a human pushing the object orthogonally to the desired trajectory. Their method consists of using Model Predictive Control and a family of m mode sequences that have been designed by the authors. Each mode has its own set of motion equations and constraints. In real-time, m convex optimization programs are solved and the mode sequence with the lowest cost is chosen. The main limitation of that work is the necessity of hand-designing a specific family of mode sequences for each task the robot has to solve.

In [8], the push dataset presented in [1] is used. The

work presented in [8], as the one in [3], uses the concept of Residual Physics, i.e., it augments analytical models with data-driven techniques to compensate for the imperfections of the models. In [8], the trained neural networks not only correct the model predictions but also provide distributions over possible outcomes of actions. This output in the form of a distribution can be used to estimate the uncertainty of the predictions, which can be useful for planning and control.

Strudel et al. [9] combine behavioral cloning (BC) [10] with reinforcement learning (RL) to solve manipulation tasks, potentially comprising several primitive motor skills. Examples of primitive skills are "grasp a cube" or "pour from a cup". Each primitive skill is learned through BC. The demonstrations are provided by an expert script with access to the full state of the environment. Once each skill has been learned through BC, a master policy is trained with the PPO [11] algorithm to select the skill that will be executed for the next n time steps, relying on sparse rewards. The proposed method is able to achieve higher success rates in the *FetchPickPlace* task from OpenAI Gym [12] than state of the art imitation or reinforcement learning methods. This work can potentially learn to combine skills with more variety than only pushing and grasping, without discretizing the action space such as in the method proposed by Zeng et al. [2]. On the other hand, it relies on predefining expert scripts for each skill, while the method proposed by Zeng et al. simply assumes a certain discretization of the action space, not the existence of expert pushing and grasping policies.

Suh and Tedrake [13] propose a switch-linear model to deal with the problem of pushing little carrot pieces towards a certain region. They do not deal with variable height and their images are binary, which is limiting when working with arbitrary heaps of objects. Nevertheless, the authors show that a switch-linear model can surpass the performance of deep learning models for this task because the tested deep models can get stuck in a loop, predicting actions that do not cause any change in the actual scene.

Approach and contribution. In this work, our aim is to learn when and how to push for different tasks, like reorienting objects to better grasp them, or pushing objects into a box on the border of a table. To do so and avoid hand crafting pushing rules, we have adopted the learning framework of Zeng et al. [2]. In this context, as in [13], we have observed that the tested deep models can get stuck in a loop, predicting actions that do not cause any change in the actual scene, and also spends a large amount of training time only to learn how to perform basic pushes that move objects. To address these two issues, our main contribution is an attention mechanism for Deep Q-Networks (DQNs) that improves their sampling efficiency by constraining the set of possible actions to actually lead to changes in the environment. Fig. 2 shows a workflow of our learning system with the attention mechanism, which will be explained in detail in Section III. We demonstrated in simulated experiments with a UR5 robot arm that our attention mechanism helps the DQN learn a push task faster and achieve better performance.

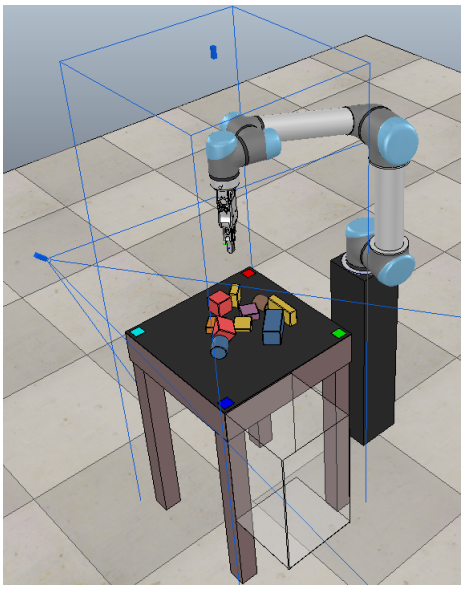


Fig. 1. Pushing into the box task.

II. Pushing Task and Learning Architecture

Fig. 1 depicts the task studied in this paper. The objective is to make the robot push all the objects on the table top into the transparent box juxtaposed with the table. The table top has dimensions 45×45 cm. A camera positioned in front of the robot captures RGB-D images from the workspace. The orthogonal projections of these images are fed as inputs to the learning architecture (Fig. 2). The robot can push at the (x, y, z) coordinates corresponding to any of the 224×224 pixels on the image. The pushing actions can be performed in any from 16 different orientations and are 10 cm long. Our approach and main contribution in this paper is to propose a learning system that combines Deep Q-Learning with an attention mechanism, improving its learning speed and performance on the pushing task.

We have defined the reward for taking action a_t at the state s_t and transitioning to the state s_{t+1} as

$$R_{a_t}(s_t, s_{t+1}) = \begin{cases} 0 \\ \text{if no change or object falls to ground,} \\ \max(0, \frac{1}{N}\sum_{i=1}^{N} d_i^t - \frac{1}{M}\sum_{i=1}^{M} d_i^{t+1}) \\ \text{if change,} \\ +10 \\ \text{if object falls into box,} \end{cases} \tag{1}$$

where change is detected as follows: The depth heightmaps before and after the push are binarized by setting 0 when the height at a given pixel is smaller than 1cm and 1 if that height is greater than or equal to 1 cm. Subsequently, the pixelwise distance between the two binary height maps is computed. If this distance is greater than 100, the scenario is deemed to have changed.

The variables d_i^t and d_i^{t+1} are, respectively, the distances before and after the push between the pixel i on an object and the target pixel defined to be in the middle of the side of the box juxtaposed with the table. The term $\frac{1}{N}\sum_{i=1}^{N} d_i^t$

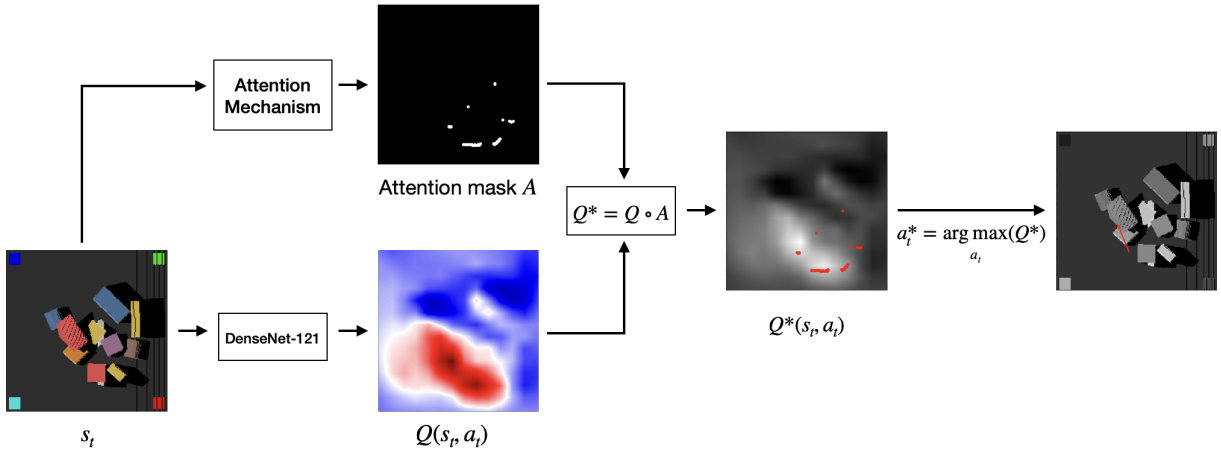


Fig. 2. Workflow of our learning system. A DenseNet predicts the state-action values $Q(s_t, a_t)$ of pushing at each pixel with a certain angle while our attention mechanism identifies push start position candidates that lead to moving objects. The mask A provided by this attention mechanism is combined via the Hadamard product with Q to produce the state-action values Q^*. The optimal push a_t^* is the one that maximizes Q^*. The attention mask A prevents trying pushes that do not lead to any changes, speeding up the learning process and reducing the chances of getting stuck in local optima.

is the average distance between the N pixels on objects in the image before the push and the target pixel. The term $\frac{1}{M}\sum_{i=1}^{M} d_i^{t+1}$ is the average distance between the M pixels on objects in the image after the push and the target pixel.

The workflow of our learning system is depicted in Fig. 2. The parameters of the DenseNet [14] are optimized via backpropagation to minimize the temporal difference error

$$\delta_t = \left|Q^*\left(s_t, a_t\right) - y_t\right| \tag{2}$$

of the predicted Q-value $Q^*\left(s_t, a_t\right)$ to a target value

$$y_t = R_{a_t}\left(s_t, s_{t+1}\right) + \gamma Q^*\left(s_{t+1}, \arg\max_{a'}\left(Q^*\left(s_{t+1}, a'\right)\right)\right), \tag{3}$$

where γ is the discount factor and a' is an action in the set of all available actions.

During training, the actions are sampled according to an ϵ-greedy approach with $\epsilon = 0.1$. A number n is sampled from a continuous uniform distribution over the half-open interval $[0, 1)$. If $n > \epsilon$, the action corresponding to the maximum predicted Q-value is selected. If $n \leq \epsilon$, an action corresponding to any of the next 99 highest Q-values is selected at random. During test, the selected actions are the ones that maximize Q^*.

III. Attention Mechanism

Our goal is to propose an attention mechanism that will improve sampling efficiency, reduce training time, and select better actions. It works by generating a mask A which is applied to the output Q of the DenseNet. The updated output which is effectively used to sample actions becomes

$$Q^* = Q \circ A, \tag{4}$$

where $\circ$ stands for the Hadamard product. For each orientation, the mask A has value 1 at pixels corresponding to suitable push candidates and 0 otherwise. See Fig. 3 for an example.

Algorithm 1 shows in detail how the attention mask A is computed. The two thresholds t_1 and t_2 are the parameters of the Canny edge detection algorithm. The offset l determines the desired distance between the start position of the push and the corresponding point on the edge of an object to be pushed. The angle θ is one of the possible 16 angles for pushing. The parameter Δx is used to compute edges that can be pushed from a given direction and the parameter δ defines the minimum distance between a push start position and any edge. This minimum distance is important due to the width of the fingers of the robot, which may prevent the gripper from reaching certain positions. Algorithm 1 uses the following equations:

$$\mathbf{p}_\theta = \mathbf{R}_\theta \mathbf{p}, \tag{5}$$

$$\mathbf{R}_\theta = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix}, \tag{6}$$

$$\mathbf{p}_\theta^{-\Delta x} = \mathbf{p}_\theta - (\Delta x, 0), \tag{7}$$

$$\mathbf{p}_\theta^{+\Delta x} = \mathbf{p}_\theta + (\Delta x, 0), \tag{8}$$

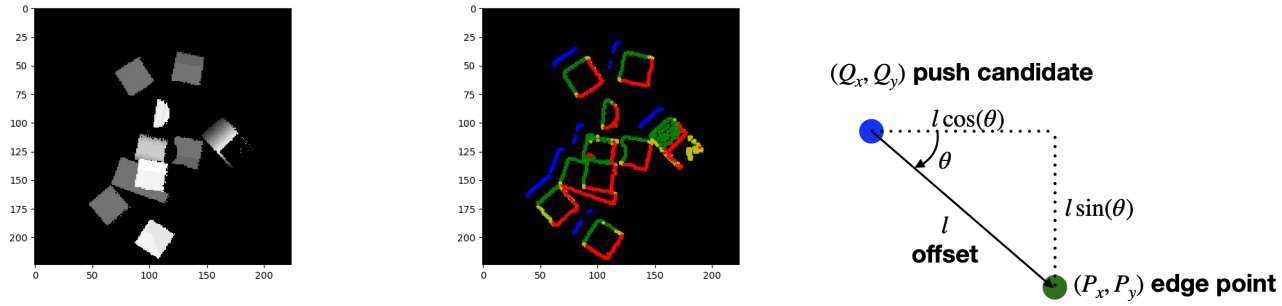
$$\mathbf{p}^{-\Delta x} = \mathbf{R}_\theta^{-1} \mathbf{p}_\theta^{-\Delta x}, \tag{9}$$

$$\mathbf{p}^{+\Delta x} = \mathbf{R}_\theta^{-1} \mathbf{p}_\theta^{+\Delta x}, \tag{10}$$

$$\Delta h = h\left(\mathbf{p}^{+\Delta x}\right) - h\left(\mathbf{p}^{-\Delta x}\right). \tag{11}$$

The value Δh is the difference between the height $h\left(\mathbf{p}^{+\Delta x}\right)$ at the point $\mathbf{p}^{+\Delta x}$ and the height $h\left(\mathbf{p}^{-\Delta x}\right)$ at the point $\mathbf{p}^{-\Delta x}$. If $\Delta h > 0.1$, the point $\mathbf{p}$ is an edge point of interest because it is on an edge with the height varying from low to high along the direction of the push, which means that this edge can be pushed along this direction. The corresponding start pushing position is then computed with $\mathbf{q} = \mathbf{p} - (l\cos(\theta), l\sin(\theta))^\top$, as depicted in Fig. 3c.

From line 15 on, the algorithm makes sure that the pushing start position candidates are far enough from any edge point such that the fingers of the gripper do not get unintentionally stuck on the top of an object when the gripper goes down to perform a push. To do so, for each point $\mathbf{p}$ on the edges, if the Euclidean distance between $\mathbf{p}$ and the pushing start position candidate $\mathbf{q}$ is less than or equal to the value d, update d



(a) (b) (c)

Fig. 3. (a) Depth heightmap. (b) The green points correspond to edges that can be pushed from a given angle, 45 degrees in this case. The blue points correspond to suitable pushing start positions given that the push should start at a distance l from the edges. The yellow edges keep approximately the same height along the direction of the push and the red edges drop in height along that direction. (c) Geometrical relation between edge points and suitable start pushing position candidates.

with that distance. If the smallest distance d between $\mathbf{q}$ and any edge point $\mathbf{p}$ is greater than or equal to δ, the pixel of the mask corresponding to the candidate $\mathbf{q}$ receives 1. In summary, a pushing start position candidate $\mathbf{q}$ is valid only if the distance between $\mathbf{q}$ and any edge point $\mathbf{p}$ is greater than or equal to δ (e.g. 10 pixels). This part of the algorithm

Algorithm 1 Attention mask based on edge detection

Inputs: $m \times n$ depth heightmap, threshold t_1, threshold t_2, offset l, angle θ, Δx, δ
Normalize heightmap such that (s.t.) its values are between 0 and 255
edges $\leftarrow$ Canny(heightmap, t_1, t_2)
Find all points $\mathbf{p}$ on the edges
Normalize heightmap s.t. its values are between 0 and 1
mask $\leftarrow$ zeros(m, n)
for each $\mathbf{p}$ **do**
    Compute the coordinates of $\mathbf{p}$ with respect to the frame rotated by $-\theta$ with (5) and (6)
    Compute $\mathbf{p}_\theta^{-\Delta x}$ with (7) and $\mathbf{p}_\theta^{+\Delta x}$ with (8)
    Compute $\mathbf{p}^{-\Delta x}$ with (9) and $\mathbf{p}^{+\Delta x}$ with (10)
    Compute Δh with (11)
    if $\Delta h > 0.1$ **then**
        $\mathbf{q} \leftarrow \mathbf{p} - (l\cos(\theta), l\sin(\theta))^\top$
        $d \leftarrow \infty$
        for each edge point $\mathbf{p}'$ **do**
            if $\|\mathbf{q} - \mathbf{p}'\|_2 \leq d$ **then**
                $d \leftarrow \|\mathbf{q} - \mathbf{p}'\|_2$
            end if
        end for
        if $d \geq \delta$ **then**
            mask(int($\mathbf{q}_y$), int($\mathbf{q}_x$)) $\leftarrow 1$
        end if
    end if
end for
return mask

IV. Experimental Results

1) Experimental Protocol: We performed experiments in simulation using a UR5 robot arm performing the task depicted in Fig. 1. The depth heightmaps in these experiments had dimensions 224×224. We have used $t_1 = 70$ and $t_2 = 100$ as parameters for the Canny edge detection. The offset l was 12 pixels. The values Δx and δ were 3 pixels and 10 pixels, respectively.

Our simulationa involved eight different object shapes and ten different object colors. Training and test scenarios were generated by sampling object shapes and colors at random and letting the objects fall from random poses around the center of the table.

The training and test procedures moved from one stage to the next if there were no objects remaining on the table top or five pushes have been performed without any change or 60 pushes have been performed in total. This upper limit of 60 pushes has been used to move on in case of any possible glitches such as an object getting stuck to the end effector.

2) Training Analysis: Figs. 4 and 5 show how the performance of a model with our attention mechanism and of a model without it evolved during training for scenarios with one and ten objects, respectively. The learning models only differed with respect to using or not the proposed attention mechanism. The discount factor γ has been set equal to zero since $\gamma = 0$ performed better than other choices (see Fig. 7). We hypothesize that the reward function defined in Equation 1 is already quite informative and that it is easier to learn the Q-values if $\gamma = 0$ because the DQN has basically to learn the reward function.

Each training stage in Figs. 4 and 5 corresponds to a scenario with a predefined object configuration. Notice that the model with the attention mechanism achieved a higher average number of objects in the box earlier than the model without it. The standard deviation of the number of objects in the box was lower for the model with the attention mechanism, meaning more consistent results. The models with the attention mechanism achieved most of the time a lower average number of objects remaining on the table top and a lower average number of objects on the ground. Moreover, the average reward achieved by the models with the attention mechanism was higher, as shown in Fig. 6.

3) Test Results: Finally, we performed tests using 100 pregenerated test scenarios with one object and with ten

Model	# objects in the box ($\mu \pm \sigma$)	# objects on the ground ($\mu \pm \sigma$)	# objects left on the table ($\mu \pm \sigma$)	# actions ($\mu \pm \sigma$)	$\frac{\text{avg. \# objects in the box}}{\text{avg. \# actions}}$
with mask	0.99 ± 0.10	0.01 ± 0.10	0.00 ± 0.00	3.19 ± 0.96	0.31
no mask	0.92 ± 0.27	0.07 ± 0.26	0.01 ± 0.10	3.67 ± 1.18	0.25

TABLE I

TESTS WITH ONE OBJECT

Model	# objects in the box ($\mu \pm \sigma$)	# objects on the ground ($\mu \pm \sigma$)	# objects left on the table ($\mu \pm \sigma$)	# actions ($\mu \pm \sigma$)	$\frac{\text{avg. \# objects in the box}}{\text{avg. \# actions}}$
with mask	7.39 ± 1.50	2.47 ± 1.29	0.14 ± 1.01	19.00 ± 6.57	0.39
no mask	4.20 ± 2.79	1.63 ± 1.43	4.17 ± 3.38	21.32 ± 6.66	0.20

TABLE II

TESTS WITH TEN OBJECTS

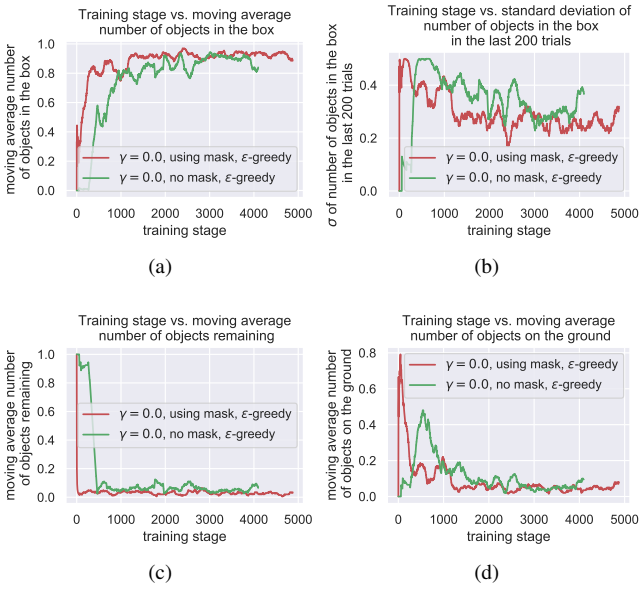


Fig. 4. Curves depicting the training of a model with our proposed attention mechanism (mask) and of a model without any attention mechanism. Each scenario used for training in this case consisted of one object with shape and color randomly chosen from a set of possible shapes and colors. The same sequence of scenarios has been used to train both models. The moving average has been computed based on the last 200 training stages. In the beginning of the training, less than 200 training stages have been used to compute the average.

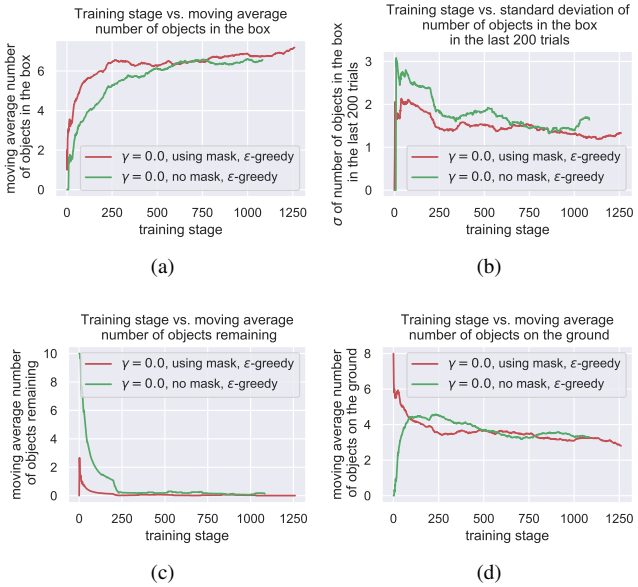


Fig. 5. Same as Fig. 4 for training with ten objects.

objects. The models tested with one object were trained in scenarios with one object and the models tested with ten objects, in scenarios with ten objects. The models tested with one object were achieved after 11800 training steps. The ones tested with ten objects, after 22350 training steps. The results presented in Tables I and II show that the model using the attention mechanism consistently outperformed the model without it.

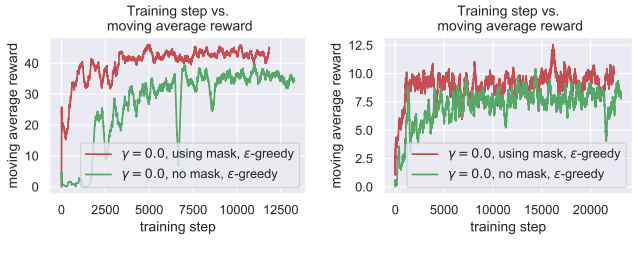


Fig. 6. Learning curves depicting how the moving average of the reward obtained by the robot changed during training. (a) Training in scenarios with one object. (b) Training in scenarios with ten objects. The moving average has been computed based on the last 200 training steps. In the beginning of the training, less than 200 training steps have been used to compute the average.

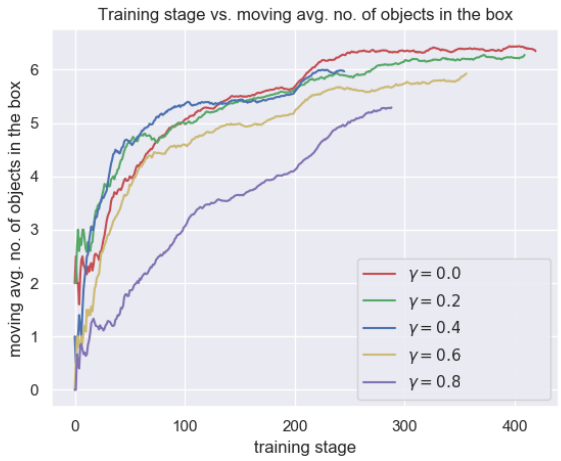


Fig. 7. Evolution of the moving average of the number of objects in the box for different choices of the discount factor γ. The model with $\gamma = 0$ achieved the highest plateau.

V. Conclusion

We proposed an attention mechanism for Deep Q-Networks (DQNs). In particular, we demonstrated that this mechanism helps a DQN to learn a push task faster and to achieve better performance. It improves the sampling efficiency of the DQN by constraining the set of possible actions to pushes that actually lead to changes in the environment.

Currently we are looking into whether such an attention mechanism can be learned by training a DQN simply to push such that there are changes in the environment. The predicted Q-values of such a DQN could then be used to mask the predicted Q-values of a DQN learning the actual task of pushing objects into the box, for example. Learning an attention mechanism in this way would be applicable to several different tasks without the necessity of designing a specific mechanism to each task. As future work, we also intend to improve the performance of the learning system with our proposed attention mechanism when pushing multiple objects into the box and to evaluate it in real-robot experiments

Acknowledgment

The research leading to these results has received funding from the Swiss National Science Foundation through the HEAP project (Human-Guided Learning and Benchmarking of Robotic Heap Sorting, ERA-net CHIST-ERA).

References

[1] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 30–37.

[2] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[3] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *arXiv preprint arXiv:1903.11239*, 2019.

[4] M. Toussaint, K. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning." in *Robotics: Science and Systems*, 2018.

[5] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[6] M. Toussaint and M. Lopes, "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4044–4051.

[7] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," *arXiv preprint arXiv:1611.08268*, 2016.

[8] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3066–3073.

[9] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, "Learning to combine primitive skills: A step towards versatile robotic manipulation," *arXiv preprint arXiv:1908.00722*, 2019.

[10] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[12] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, *et al.*, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," *arXiv preprint arXiv:1802.09464*, 2018.

[13] H. Suh and R. Tedrake, "The surprising effectiveness of linear models for visual foresight in object pile manipulation," *arXiv preprint arXiv:2002.09093*, 2020.

[14] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.