Lazy Action Value Models for Q-Learning in Robotics*

Ingmar Schubert[1] and Marc Toussaint[2]

***Abstract*— Value-based algorithms for reinforcement learning such as Q-learning offer state-of-the-art performance in many settings with discrete action domains. However, plain value-based algorithms are limited in large or continuous action domains, which are needed for applications in robotics. Here, we propose to approximate the action-dependent part of the Q-function locally and to learn it on-the-fly from a data buffer. This gives rise to a simple framework for learning universal Q-functions in continuous action domains from off-policy data. We demonstrate the feasibility of our approach using a goal-conditioned robot pushing task.**

I. Introduction

Reinforcement learning provides the field of robotics with a powerful toolbox enabling the robot to learn complex tasks only from rewards [1]. This combination has proven to be very successful with many applications such as autonomous vehicle control [2], manipulation [3], or direct learning of visuomotor policies [4]. Reinforcement learning in robotics presents an intriguing avenue of current research due to its potential to lead the way to truly intelligent self-learning machines that adapt to dynamical environments and improve over time.

One of the specific challenges to the application of reinforcement learning to robotics is that typically, state and action spaces are both high-dimensional and continuous. In this work, we aim at providing an additional point of attack to the challenges posed by continuous action spaces. We propose a framework that combines value-based off-policy reinforcement learning with a lazy learning algorithm for the action value model to make it applicable to continuous action domains. To the best of our knowledge, this is the first work proposing such an approach. Further, we introduce a modification to stabilize the learning process, and discuss the combination of our approach with popular algorithms in deep reinforcement learning, such as Deep Q Networks (DQN) [5] and Double DQN [6].

II. Related Work

A. Q-learning

Value function methods for reinforcement learning such as Q-learning [7], [8] aim at learning the Q-function from the interaction with the environment which then implicitly defines a policy. In many real-world problems, the state and action spaces are too large to represent the Q-function in a tabular form. In this case, it can be parameterized, e.g. by a deep neural network [9]. This has allowed for great successes e.g. for the task of playing complex Atari games [5], [6].

While Q-learning approaches are very successful in discrete action spaces, traditional approaches are limited in continuous or large action spaces, as for example needed in robotics. There exist different approaches to extend Q-learning to continuous action domains, which include learning Q-values for discrete actions and interpolating between them [10], and, more recently, learning a representation of the Q-function that allows for an analytical maximization step with respect to the action [11]. In this work, we instead propose to combine an eagerly learned state value function $V(s)$ with a local action value model that is lazily learned directly from data. This is different from [10] in that we do not rely on learning Q-values for a discrete set of actions, and it is different from [11] in that no specific representation of $V(s)$ has to be learned, but our approach is in fact compatible with any off-the-shelf function approximator.

B. Policy Gradient Methods

In domains with continuous action spaces, policy-gradient based methods are a popular choice [12], [13], [14], [15]. These approaches have in common that they require explicitly learning the policy in addition to the Q-function. In contrast to that, we avoid the additional complexity of learning an explicit policy. Instead of training a policy to choose optimal actions, the optimal action in our approach is found using the lazy action value model.

C. Lazy Q-Learning

Lazy learning [16] has been combined with Q-learning before. In [17], actors in a pursuit game use lazy learning to predict action values for state-action pairs directly from neighboring state-action pairs in the database. A similar approach is used in [18] in a cooperative multi-robot setting. Our approach differs from both of the aforementioned in that the state value function $V(s)$ is not lazily but indeed eagerly learned, and combined with a lazily learned local action value model in order to obtain the Q-function $Q(s, a)$. This approach allows us to make use of the good generalization properties of deep neural networks for the value function, while being able to avoid the limitations of classical Q-learning in continuous action spaces by lazily creating local action value models. In other words, we propose to extend Q-learning to continuous action spaces by expanding the global information in the value function with local information on the dynamics, the latter being obtained directly from

*This work was supported in part by the *German Academic Scholarship Foundation*

[1]Technische Universität Berlin, 10623 Berlin, Germany ingmar.schubert@tu-berlin.de

[2]Technische Universität Berlin, 10623 Berlin, Germany and Max Planck Institute for Intelligent Systems, 70569 Stuttgart, Germany toussaint@tu-berlin.de

data by lazy learning. Considering the example of a robot manipulation task, the value function contains the global information on which end effector trajectory will result in a successful manipulation, and the lazy action value model uses local experience data at each step to decide which joint movements will move the end effector towards or further along this trajectory.

III. Q-Learning and Markov Decision Processes

We consider setups that can be described as a discrete-time deterministic *Markov decision process* (MDP) [19] with states $s \in \mathcal{S}$, continuous actions $a \in \mathcal{A} \subseteq \mathbb{R}^{n_a}$, and goals $g \in \mathcal{G}$. The deterministic transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ assigns each combination of state and action to the resulting next state. The agent interacts with the environment following its deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$, collects data tuples of the form $d_t = (s_t, a_t, s'_t, r_t, g_t)$ at each discrete time step $t \in \{0, 1, ...\}$, and stores it into its database $\mathcal{D} = \{d_0, d_1, ...\}$. Here, s_t is the state at time t, a_t is the action taken, and s'_t is the resulting state. Dependent on the goal g_t, the agent receives the reward r_t that follows the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{G}$, i.e. $r_t = R(s_t, a_t, g_t)$.

The Q-function quantifies the value of choosing action a in state s, assuming that the given policy π is followed thereafter.

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s,\ a_t = a, \pi\right] \quad (1)$$

Here, $\gamma \in [0, 1)$ is the discount factor trading off immediate reward with future reward. $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ is the Q-function of the optimal policy π^*, fulfills the Bellman equation

$$Q^*(s, a, g) = R(s, a, g) + \gamma V^*(T(s, a), g) \quad , \quad (2)$$

and can be learned using Q-learning [7], [8]. Here,

$$V^*(s, g) = \max_{a \in \mathcal{A}} [Q^*(s, a, g)] \quad (3)$$

is the state value function.

Universal or goal-conditioned value functions [20] capture the value of a state-action pair under different goals $g \in \mathcal{G}$ within the otherwise same MDP. If the reward function is known, universal value functions can be learned in a sample-efficient way using Hindsight Experience Replay [21].

IV. Lazy Action Value Models for Q-Learning

In the following we introduce our approach combining value-based reinforcement learning with a lazy action value model. We also discuss its limitations, investigate its stability, and touch on the combination of our approach with popular algorithms in deep reinforcement learning. Our aim is to learn universal Q-functions from off-policy data collected by the agent in continuous action spaces.

A. Lazy Q-function estimate

Instead of explicitly learning the Q-function $Q^*(s, a, g)$, we estimate the action values lazily. Specifically, we assume a linear dependency on the actions a in the sense that

$$Q^*(s, a, g) \approx \alpha(s, g) + \beta(s, g)^T a \quad . \quad (4)$$

This linear Taylor approximation works well in robotics applications with small joint movements per time step. In the limit of infinitesimally small actions, i.e. $||a|| \to 0$, it becomes exact. Please also notice that while we use an action value model that is linear in a here, our framework does not rely on this. Other local function approximations, for example higher-order Taylor expansions in a, could be used as well.

The parameters $\alpha(s, g)$ and $\beta(s, g)$ are learned lazily from the data set by minimizing

$$(\alpha(s, g), \beta(s, g)) = \underset{\alpha, \beta}{\operatorname{argmin}} \left[\sum_{(s_i, g_i) \in B(s,g)} \Delta_i^2(\alpha, \beta)\right] \quad (5)$$

with

$$\Delta_i(\alpha, \beta) = \left[\alpha + \beta^T a_i\right] - \left[r_i + \gamma V^*(s'_i, g_i)\right] \quad , \quad (6)$$

where $B(s, g) \subseteq \mathcal{S} \times \mathcal{G}$ is a neighborhood around (s, g) in state-goal space.

B. Self-consistent state value function

Using the local action value model, the Bellman update (3) takes the form

$$V^*(s, g) = \alpha(s, g) + \max_{a \in \mathcal{A}} \beta(s, g)^T a \quad , \quad (7)$$

where the maximization step can be done analytically in many cases. For example, if $\mathcal{A} = \{\mathbb{R}^{n_a} \mid ||a|| \leq c\}$, then $\max_{a \in \mathcal{A}} \beta(s, g)^T a = c\,||\beta||$. The update (7) relies on the applicability of the local model to the Q-function (4), i.e. on the approximately linear dependency of $Q(s, a, g)$ on a in our case. Violations of this assumption can lead to instabilities, and therefore have to be treated carefully.

C. Stability

The neighborhood $B(s_0, g_0) \subseteq \mathcal{S} \times \mathcal{G}$ is a sufficiently large volume around (s_0, g_0) in which (4) still holds true approximately for all $(s, g) \in B(s_0, g_0)$ with the same parameters $\alpha(s_0, g_0)$ and $\beta(s_0, g_0)$. The neighborhood is sufficiently large if it covers enough sample points to capture all relevant information on the local dynamics.

Is is not always possible to find such a $B(s_0, g_0)$ that has nonzero volume in $\mathcal{S} \times \mathcal{G}$. For example, consider an environment that gives negative rewards as long as the agent is in a "forbidden" area. At the border of the "forbidden" area to the "neutral" area, the agent's actions will either result in a negative reward or no reward, resulting in a discontinuity of $Q^*(s, a, g)$. Regardless of how small $B(s_0, g_0)$ is chosen, such a discontinuity will always violate (4) and thus lead to instabilities during the Bellman update step (7) as illustrated in figure 1.

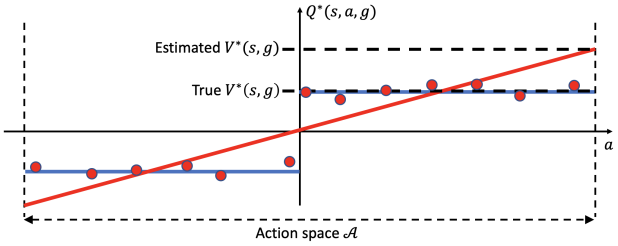


Fig. 1. A Q-function $Q^*(s,a,g)$ (in blue) exhibits a sharp step in the action space $\mathcal{A}$, thus violating the assumption (4). The linear model fitted to the data (red dots) is shown in red. Using the linear-model Bellman update step (7), the linear model will lead to a systematic overestimation of $V^*(s,g)$, which will prevent convergence. Using the maximum-bound version (8) instead avoids this issue.

We account for this by refining the Bellman update (7) to

$$V^*(s,g) = \min\Bigg\{\alpha(s,g) + \max_{a\in\mathcal{A}} \beta(s,g)^T a, \quad \max_{i:\ (s_i,g_i)\in B(s,g)} \left[r_i + \gamma V(s'_i, g_i)\right]\Bigg\} \quad . \tag{8}$$

In words, we clip the prediction of the lazy model if its predicted value of the next state s' is higher than the value of the actually recorded next states. This ensures that discontinuities as in figure 1 do not result in instabilities.

D. Optimal Policy

The optimal policy π^* acts greedily with respect to $Q^*(s,a,g)$. Again using our lazy estimate of the Q function, this results in the policy

$$\pi^*(s) = \operatorname*{argmax}_{a\in\mathcal{A}} Q^*(s,a,g) = \operatorname*{argmax}_{a\in\mathcal{A}} \beta(s,g)^T a \quad . \tag{9}$$

E. Function Approximators

Our approach is agnostic to how the state value function $V^*(s,g)$ is represented. In discrete state-goal spaces, $V^*(s,g)$ could be represented in tabular form, in higher-dimensional or continuous spaces however, function approximators such as neural networks are more suitable. Suppose that $V^*(s,g)$ is represented by a neural network parameterized by θ. The neural network is trained using the loss function

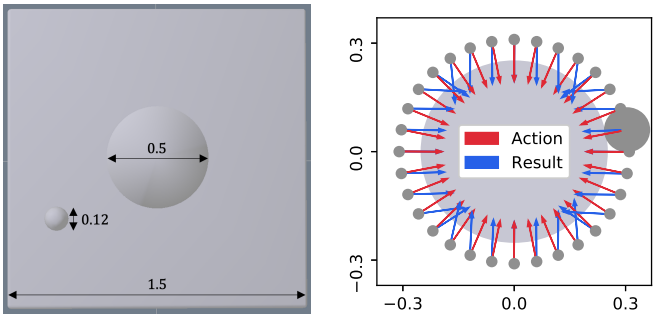
$$L(\theta) = \sum_i \left(Y_i - V^*(s_i,g_i,\theta)\right)^2 \quad , \tag{10}$$

with the targets

$$Y_i = \min\Bigg\{\alpha(s_i,g_i,\theta) + \max_{a\in\mathcal{A}} \beta(s_i,g_i,\theta)^T a, \quad \max_{j:\ (s_j,g_j)\in B(s_i,g_i)} \left[r_j + \gamma V(s'_j, g_j, \theta)\right]\Bigg\} \tag{11}$$

and

$$(\alpha(s,g,\theta), \beta(s,g,\theta)) = \operatorname*{argmin}_{\alpha,\beta} \left[\sum_{(s_i,g_i)\in B(s,g)} \Delta_i^2(\alpha,\beta,\theta)\right]$$
$$\Delta_i(\alpha,\beta,\theta) = \left[\alpha + \beta^T a_i\right] - \left[r_i + \gamma V^*(s'_i, g'_i, \theta)\right] \quad . \tag{12}$$



(a) (b)

Fig. 2. (a) Experimental Setup: The disk is initialized at the origin in the middle of the table. The spherical finger is controlled by the agent and is supposed to push the disk in a certain direction specified by the goal $g \in \mathcal{G}$. (b) Resulting movement (blue arrows) of the disk (light grey) when pushed by the finger (dark grey) in normal direction (red arrows). An anisotropic friction term results in a deflection of the disk's movement when pushed by the finger in diagonal direction.

Popular Q-learning algorithms such as DQN [5] and Double DQN [6] use a separate target network to obtain the target values Y_i from. Analogously, the DQN version of our algorithm can be obtained by replacing θ with the target network's parameters θ_t in (11) and, consequently, also for the calculation of the parameters α and β (12). The Double DQN algorithm on the other hand relies on estimating the optimal action from the primary network and then quantifying its value with the target network. The analogous Double DQN target in our framework is

$$Y_i^{\mathrm{DDQN}} = \min\Bigg\{\alpha(s_i,g_i,\theta_\mathrm{t}) + \beta(s_i,g_i,\theta_\mathrm{t})^T a_{\mathrm{opt}}(\theta), \quad \max_{j:\ (s_j,g_j)\in B(s_i,g_i)} \left[r_j + \gamma V(s'_j, g_j, \theta_\mathrm{t})\right]\Bigg\} \tag{13}$$

with the optimal action being estimated from the primary network

$$a_{\mathrm{opt}}(\theta) = \operatorname*{argmax}_{a\in\mathcal{A}} \left(\alpha(s_i,g_i,\theta) + \beta(s_i,g_i,\theta)^T a\right) \quad . \tag{14}$$

V. Experiments

To provide preliminary results on the feasibility of our framework, we apply it to a simulated robot pushing task with realistic physics. We describe the experimental setup and our algorithm briefly in the following.

A. Setup

1) Simulated environment: We use the NVIDIA PhysX engine [22] to simulate a disk of radius 0.25 lying on a table of size 1.5×1.5, as shown in figure 2a. The state of the environment is given by the 2D position of a spherical "finger" of radius 0.06 that is supposed to move the disk in a way specified by a goal g. A goal g contains the intended movement direction of the disk's coordinates, given as the angle to the x-direction. The possible actions are movements of the finger of constant length $|a| = 0.1$ into any direction. They are performed sufficiently slow so that the system is always quasistatic. In the simulation, we include

an anisotropic friction term which results in a lateral drift when the disk is pushed diagonally, as illustrated in figure 2b. Thus, diagonal pushes are more sensitive to the contact point and the direction of the push and therefore more difficult to perform successfully.

In summary, the system has the state space $\mathcal{S} = \mathbb{R}^2$, action space $\mathcal{A} = \{|a|(\cos(\phi), \sin(\phi)) \mid \phi \in [-\pi, \pi)\}$, and goal space $\mathcal{G} = [-\pi, \pi)$. The transition function reads $T(s, a) = s + a$. The reward function depends on the disk's position after the push, therefore it depends on the outcome of the black-box simulation. If the disk has been moved by δ_{disk}, the reward reads

$$R(s, a, g) = \begin{cases} 0 & \text{if } ||\delta_{\text{disk}}|| < 0.01 \\ -1 & \text{if } ||\delta_{\text{disk}}|| \geq 0.01 \text{ but wrong direction} \\ 1 & \text{if } ||\delta_{\text{disk}}|| \geq 0.01 \text{ and correct direction.} \end{cases} \tag{15}$$

Here, the direction is correct if the cosine of the angle $\angle(\delta_{\text{disk}}, g)$ between δ_{disk} and $(\cos(g), \sin(g))$ is larger than 0.9, and otherwise wrong. Once a nonzero reward has been received, i.e. the disk was moved, the rollout episode is stopped and the disk is reset to the origin.

2) Calculation of the nearest neighbors: Given a point (s_0, g_0), we define its neighborhood in the following way: $(s, g) \in B(s_0, g_0)$, iff

- The euclidean distance between the states is smaller than $||s - s_0|| < 0.06$, and
- the goal angle's true distance is smaller than 0.1, i.e. $\cos(g_{\text{disk}} - g_{0,\text{disk}}) > \cos(0.1)$.

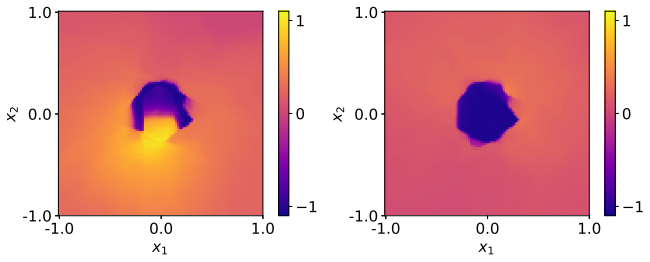
If $B(s_0, g_0)$ does not contain at least 20 points, we increase all boundaries by the factor $2^{1/3}$, in order to approximately double the volume. This is done until the required number is reached. We then calculate (5) using 20 points that are randomly sampled from $B(s_0, g_0)$ without replacement.

3) Data collection: To collect the off-policy dataset $\mathcal{D}$, we initialize the finger's state at 5000 random positions in $[-1, 1]^2$, but outside the disk. From each state, we perform 2 actions into the same randomly selected direction. The reward is recorded with respect to 100 different goals that are sampled uniformly from $\mathcal{G}$ for each step. This results in a comprehensive static dataset that allows us to test our algorithm in an isolated way.

4) Agent architecture and training procedure: We represent $V^*(s, g, \theta)$ as a neural network with 9 fully connected layers and batch normalization layers [23] in alternating order, implemented in Tensorflow [24]. We train the architecture on the static dataset, using targets as in (11) with discount factor $\gamma = 0.9$. We use ADAM [25] optimization steps. After this, the targets are updated and the entire process is repeated until convergence. In total, we use 10 such cycles with 10 training epochs each.

B. Results

Figure 3 shows the learned $V^*(s, g, \theta)$ as function of the state s for two different goals g. We test the performance of the agent by initializing the disk at the origin and initializing



(a) (b)

Fig. 3. Learned optimal state value function $V^*(s, g, \theta)$ as a function of the state $s = (x_1, x_2)$, shown for the goals (a) $g = \pi/2$, for which positive rewards have been collected and (b) $g = 5\pi/4$, for which the naive random exploration scheme did not obtain sufficient positive rewards.

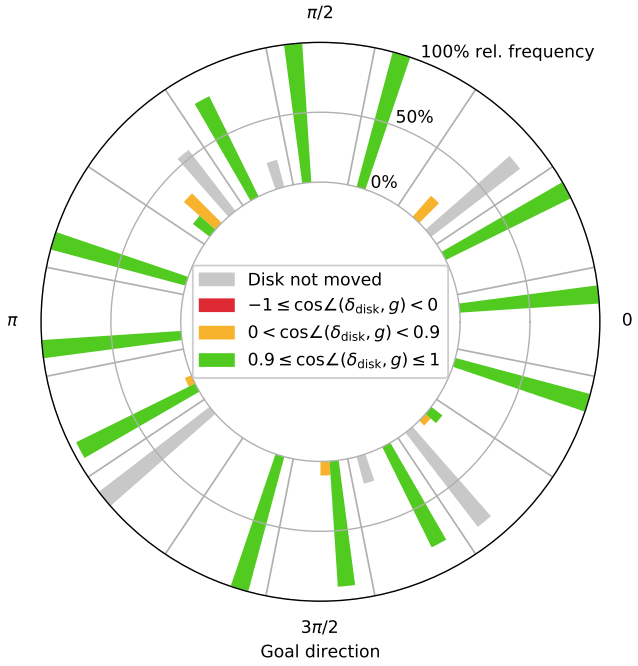


Fig. 4. Pushing task results for a trained agent. For each of 16 different goal directions g, 20 episodes are rolled out for 50 steps or until the disk is moved, i.e. a nonzero reward is received. The 16 histograms show relative frequencies of the outcome after each episode.

the finger at random positions in $[-1, 1]^2$ that are on the opposite side of the goal direction. For 16 different goals, we perform 20 rollouts each that are stopped once the disk has been moved, i.e. a nonzero reward is received, or after 50 steps. We measure if the disk has been pushed in the correct direction ($0.9 \leq \cos\angle(\delta_{\text{disk}}, g)$), if it has been pushed but into a wrong direction ($-1 \leq \cos\angle(\delta_{\text{disk}}, g) < 0$ or $0 < \cos\angle(\delta_{\text{disk}}, g) < 0.9$), or if has not been pushed at all after 50 steps.

Figure 4 shows the results. Due to the anisotropic friction term, the disk's movement along diagonal directions is very sensitive to the contact point and the direction of the push. In these cases, positive rewards are very unlikely to be obtained with our naive random data collection method. Based on this data, the agent generally learns the correct behavior, namely

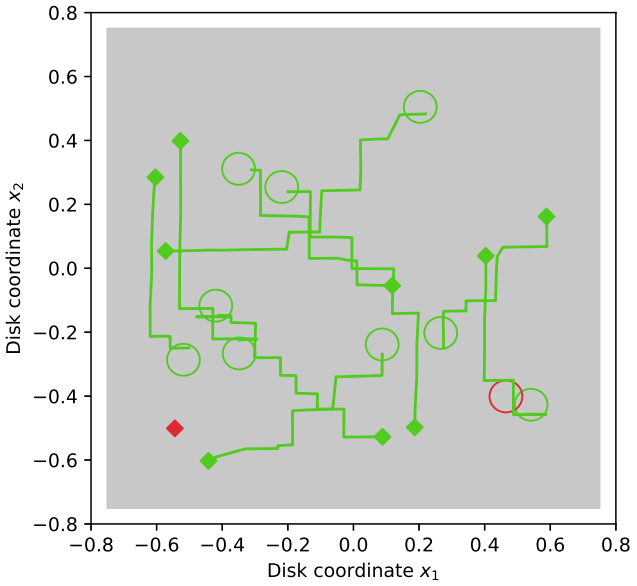


Fig. 5. The disk's trajectories during 10 randomly chosen rollouts on the table (grey). The starting points are diamond-shaped, and the goal areas are indicated by circles. The successful rollouts are shown in green, and an unsuccessful rollout is shown in red. Since we only use the goal directions 0, $\pi/2$, π, and $3\pi/2$ in this experiment, the disk is pushed along diagonal trajectories in a zigzag path.

not touching the disk at all. This can also be seen from the value function $V^*(s, g = 5\pi/4)$ shown in figure 3b. For the vertical and horizontal goal directions however, positive rewards can be found even by random exploration, allowing the agent to learn from this data and perform the pushing task into these directions reliably.

In a second experiment, we tested the resulting goal-conditioned policy in conjunction with a higher-level controller in order to fulfill a long-horizon pushing task. The high-level controller specifies the direction in which the disk is supposed to be moved during the next step. This is expressed in the form of a goal to the goal-conditioned policy that was described before. Let $\mathbf{x}$ be the 2D position of the disk, and let $\mathbf{x}^*$ be its intended target state. At each step, the high-level controller sets the direction $\mathbf{x}^* - \mathbf{x}$ as the new goal. We only use the goal directions 0, $\pi/2$, π, and $3\pi/2$ that the agent can fulfill reliably, and follow the one that is best aligned with the direction of $\mathbf{x}^* - \mathbf{x}$. We let the agent act for 300 steps at most, and count the run as successful if the agent succeeds to push the disk within $||\mathbf{x}^* - \mathbf{x}|| < 0.05$ before running out of time.

Both the disk and its target position are initialized at random positions within $[-0.6, 0.6]^2$, i.e. on the table. The robot finger is initialized at random positions outside the disk that are within a box of size 1.2×1.2 around the disk's position. In total, we conducted 100 rollouts, of which 95 succeeded. During 5 of the rollouts, the agent became stuck in a local minimum or pushed the disk off the table. Figure 5 shows 10 of the rollouts.

VI. Conclusions

We introduced a framework that combines an eagerly learned state value function with a lazily learned action value model to obtain goal-conditioned policies for continuous action spaces. The core idea is that the value information, being global in nature, is learned eagerly, while the dynamics information, being local in nature, is learned lazily in a separate step. We demonstrated the feasibility of our approach using a goal-conditioned robot pushing task. Compared to value-based approaches like plain Q-learning, our approach can be applied to continuous action spaces as well. Compared to policy-gradient approaches like actor-critic, our approach is less complex, since we only have to learn a single value function. We believe that especially for applications in robotics, where the actions per time step are small and local action value models work well, this could be an interesting avenue for future research.

Acknowledgment

I.S. would like to thank Jung-Su Ha for insightful discussions.

References

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in neural information processing systems*, 2007, pp. 1–8.

[3] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant manipulation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4639–4644.

[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[7] R. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 08 1988.

[8] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[9] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning*. Springer, 2005, pp. 317–328.

[10] J. D. R. Millán, D. Posenato, and E. Dedieu, "Continuous-action q-learning," *Machine Learning*, vol. 49, no. 2-3, pp. 247–265, 2002.

[11] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[12] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[13] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine learning*, vol. 84, no. 1-2, pp. 137–169, 2011.

[14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[17] J. W. Sheppard and S. L. Salzberg, "A teaching strategy for memory-based control," in *Lazy Learning*. Springer, 1997, pp. 343–370.

[18] C. F. Touzet, "Distributed lazy q-learning for cooperative mobile robots," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 1, 2004.

[19] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[20] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning*, 2015, pp. 1312–1320.

[21] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in neural information processing systems*, 2017, pp. 5048–5058.

[22] (2020, Mar.) Nvidia physx product site. [Online]. Available: https://developer.nvidia.com/gameworks-physx-overview

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.