Deep Reinforcement Learning with Embedded LQR Controllers ⋆

Wouter Caarls *

* *Pontifical Catholic University, Rio de Janeiro, RJ 22451-900 Brazil (e-mail: wouter@ele.puc-rio.br).*

Abstract: Reinforcement learning is a model-free optimal control method that optimizes a control policy through direct interaction with the environment. For reaching tasks that end in regulation, popular discrete-action methods are not well suited due to chattering in the goal state. We compare three different ways to solve this problem through combining reinforcement learning with classical LQR control. In particular, we introduce a method that integrates LQR control into the action set, allowing generalization and avoiding fixing the computed control in the replay memory if it is based on learned dynamics. We also embed LQR control into a continuous-action method. In all cases, we show that adding LQR control can improve performance, although the effect is more profound if it can be used to augment a discrete action set.

Keywords: Reinforcement learning, actor-critic methods, learning control, linear quadratic regulator.

1. INTRODUCTION

Reinforcement Learning (RL, Sutton and Barto (2018)) is an optimal control method that estimates control policies from direct interaction with a (real or simulated) environment. It has been sucessfully applied in such diverse areas as robotics (Akkaya et al. (2019)), game playing (Vinyals et al. (2019)), and medication dosing (Nemati et al. (2016)). RL methods can be divided in two broad categories: those that directly optimize the parameters of an explicit control policy (called policy search methods), and those that estimate the reward-to-go (value) function and implicitly derive a control policy from that (value-based methods).

Value-based methods are better for tasks that inherently contain many local optima ("maze-like tasks"). However, they struggle in problems with continuous action spaces such as robotics, because the policy is defined as taking the action with the highest reward-to-go, which is usually implemented by iteration over a discrete set of possible actions. This is especially relevant for regulation or reaching tasks, because a controller using discrete actions can never maintain a stable dynamic position.The resulting chattering is undesirable and can lead to system damage (Meijdam et al. (2013)).

One solution is to use a lower-level (continuous-action) stabilizing controller at the goal state, while reinforcement learning optimizes the trajectory towards that goal. This can for example be implemented by defining a "capture region" in which the stabilizing controller is active (Randløv et al. (2000)), or by making it one of the possible discrete actions (Abramova et al. (2019)). The first case requires manual fine-tuning of this region, while the second requires learning the value of a new action. In this paper, we propose using the stabilizing controller's action one of the sampling points of the discrete action set, thus taking advantage of the action-space generalization provided by modern deep reinforcement learning methods.

In addition, we investigate the use of the same approach in an actor-critic setting. Actor-critic methods are a hybrid of policy-based and value-based RL methods. They use a value function to inform the adaptation of an explicitly represented policy. Although they do not suffer from a discrete action set, the intuition is that the inclusion of the action suggested by a (model-based) stabilizing controller could increase the learning speed, as it provides near-optimal actions in at least part of the state space.

The paper is organized as follows. Sections 2 and 3 introduce the basic theory and our specific methods of combining RL with LQR control, respectively. Sections 4 and 5 present the experiments, results and their analysis. Finally, section 6 concludes the paper and discusses future work.

2. THEORY

2.1 Reinforcement Learning

A Markov Decision Process (MDP) is a 4-tuple $< \mathcal{S}, \mathcal{A}, T, R >$ where $\mathcal{S}$ is the (possibly continuous) state space, $\mathcal{A}$ the (possibly continuous) action space, $T : \mathcal{S}, \mathcal{A}, \mathcal{S} \rightarrow \mathbb{R}$ a function specifying the state transition probabilities and $R : \mathcal{A}, \mathcal{S} \rightarrow \mathbb{R}$ a reward function. The goal of reinforcement learning is to find a control policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes, for every state $s \in \mathcal{S}$ the

⋆ ©2020 the authors. This work has been accepted to IFAC for publication under a Creative Commons Licence CC-BY-NC-ND. This research was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), grant number 304980/2018-8.

expected *discounted return* $\mathfrak{R}_t$ of executing that policy starting from s at the current timestep t:

$$\mathfrak{R}_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

where r_k is the reward received at timestep k and $\gamma \in [0, 1]$ is a discount factor that determines the optimization horizon.

In value-based reinforcement learning, the expected returns are stored in a state-value function $V^\pi : \mathcal{S} \to \mathbb{R}$, or an action-value function $Q^\pi : \mathcal{S}, \mathcal{A} \to \mathbb{R}$. In the latter case, the stored values $Q(s, a)$ indicate the return of taking action a in state s and following π afterwards. The optimal policy π^* always takes the action with the highest expected return, and its value functions are therefore related:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \tag{2}$$

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a). \tag{3}$$

Q^* is the unique solution to the Bellman optimality equation

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left(R(a, s') + \gamma V^*(s') \right). \tag{4}$$

Note that it is not necessary to define V^* in terms of Q^*, as Eqs. 3 and 4 may be combined into a single equation. However, in that case the transition probabilities T must be known in order to derive π^* in Eq. 2.

We consider model-free reinforcement learning, in which the transition probabilities T are unknown. In that case, the value functions are usually estimated by sampling experienced *transitions* (s_t, a_t, r_t, s_{t+1}). Each transition gives a sample of the Bellman optimality equation (4):

$$Q(s_t, a_t) = r_t + \gamma V(s_{t+1}) \tag{5}$$

$$= r_t + \gamma \arg\max_{a' \in \mathcal{A}} Q(s_{t+1}, a') \tag{6}$$

Deep Q learning In deep Q learning (DQN, Mnih et al. (2013)), the sampled transitions, gathered under an exploratory policy based on π, are stored in a *replay memory* which is used to approximate the action-value function with a deep neural network $\hat{Q}(s, a|\theta^Q)$ with parameters θ^Q. Iteratively (generally after every new transition), a *mini-batch* of N transitions (s_i, a_i, r_i, s'_i) is sampled randomly from this replay memory and used to minimize the L_2 loss between the left-hand and right-hand sides of Eq 6:

$$\mathcal{L} = \sum_i \left\| \hat{Q}(s_i, a_i|\theta^Q) - \left(r_i + \gamma \arg\max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a'|\theta^{Q'}) \right) \right\|_2. \tag{7}$$

Note the use of different weights $\theta^{Q'}$ in the right-hand side, used to stabilize the learning. These *target weights* are updated periodically from the learned weights θ^Q. The exploratory policy is usually ϵ-greedy with respect to Eq. 2:

$$\pi^\epsilon(s) = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s, a) & \text{with probability } \epsilon \\ \text{random}(\mathcal{A}) & \text{otherwise} \end{cases}. \tag{8}$$

Deep Deterministic Policy Gradient Actor-critic algorithms represent π explicitly instead of deriving it from Q. In the deep deterministic policy gradient algorithm (DDPG, Lillicrap et al. (2015)), the Q function is approximated as in deep Q learning, but the arg max in Eq. 7 is replaced by the action given by a separate policy network $\mu(s|\theta^\mu)$ trained to be the mean of a Gaussian policy

$$\pi(s|\theta^\mu) = \mathcal{N}(\mu(s|\theta^\mu), \sigma) \tag{9}$$

where σ is the standard deviation of the exploration distribution. After every Q update, the weights θ^μ of the policy network are updated in the direction of

$$\mathbb{E}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)} \right] \approx \frac{1}{N} \sum \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i}. \tag{10}$$

As in deep Q learning, separate target networks with weights $\theta^{Q'}$ and $\theta^{\mu'}$ are used to calculate the right hand side of Eq. 7. However, instead of periodic updates, they are updated continuously from θ^Q and θ^μ using a moving average filter. In addition, the Gaussian policy is usually replaced by one that has time-correlated noise to improve exploration (Lillicrap et al. (2015)).

2.2 Locally Linear Regression

We use locally linear regression (LLR, Atkeson et al. (1997)) to approximate the dynamics around the goal state s_d. LLR is well-suited to model approximation for LQR because the linear model it estimates can be directly split into the A and B matrices required by LQR control. In LLR, all transitions (s_i, a_i, s'_i) are stored and a linear model X is fit around the K nearest neighbors of the query point s_d

$$N_\mathrm{I} = \begin{bmatrix} \overline{s_1}, a_1, 1 \\ \overline{s_2}, a_2, 1 \\ \cdots \\ \overline{s_K}, a_K, 1 \end{bmatrix}, N_\mathrm{O} = \begin{bmatrix} \Delta s_1 \\ \Delta s_2 \\ \cdots \\ \Delta s_K \end{bmatrix}, \tag{11}$$

where $\overline{s_i} = s_i - s_\mathrm{d}$ and $\Delta s_i = s'_i - s_i$, by solving

$$(N_\mathrm{I}^\mathsf{T} N_\mathrm{I}) X = N_\mathrm{I}^\mathsf{T} N_\mathrm{O} \tag{12}$$

using the Cholesky decomposition with Tikhonov regularization. The A and B matrices of the discrete-time system

$$s_{t+1} = A s_t + B a_t + E \tag{13}$$

are then defined as follows:

$$A = \overline{X}_{1:|\mathcal{S}|, 1:|\mathcal{S}|} \tag{14}$$

$$B = \overline{X}_{1:|\mathcal{S}|, |\mathcal{S}|+1:|\mathcal{S}|+|\mathcal{A}|}, \tag{15}$$

where $\overline{X} = X + I$, and $|\mathcal{S}|$ and $|\mathcal{A}|$ are the dimensionalities of $\mathcal{S}$ and $\mathcal{A}$, respectively.

2.3 Linear Quadratic Regulator

The Linear Quadratic Regulator is an optimal solution to the Bellman equation (4) for linear systems with quadratic rewards (Mehrmann (1991)), in our case

$$R(s,a) = -\left(\overline{s}^{\mathsf{T}} C \overline{s} + a^{\mathsf{T}} D a\right), \tag{16}$$

where C and D are diagonal matrices specifying the costs for deviating from the goal state s_{d} and zero action, respectively. The solution is computed by solving the discrete time algebraic Riccati equation (Benner and Sima (2003))

$$P = A^{\mathsf{T}} P A - \left(A^{\mathsf{T}} P B\right)\left(D + B^{\mathsf{T}} P B\right)^{-1}\left(B^{\mathsf{T}} P A\right) + C \tag{17}$$

after which the control is given by

$$a_t = -F\overline{s_t} \tag{18}$$

$$F = \left(D + B^{\mathsf{T}} P B\right)^{-1}\left(B^{\mathsf{T}} P A\right). \tag{19}$$

To find the steady-state feedforward action a_{ff} to cancel the disregarded E term in Eq. 13, we solve

$$B a_{\mathrm{ff}} = A s_{t+1}|_{s_t = s_{\mathrm{d}}, a_t = 0} - s_{\mathrm{d}} \tag{20}$$

using the singular value decomposition and add it to the regulator output.

3. METHODS

We compare three different ways of integrating LQR control with reinforcement learning: LQR capture, LQR action and integrated LQR action.

3.1 LQR Capture

In LQR capture, the system is controlled by the reinforcement learning agent except in a region around the goal state, in which case control is taken over by the LQR controller (Randløv et al. (2000)). From the point of view of the RL agent, this turns the system into a semi-MDP (sMDP, Sutton et al. (1999)), where the action that led to the capture region is temporally extended until the system leaves the capture region, or until the end of the episode. In sMDPs, the target in Eq. 6 is replaced by

$$Q(s_t, a_t) = \sum_{k=0}^{\Delta t - 1} \gamma^k r_{t+k} + \gamma^{\Delta t} \arg\max_{a' \in \mathcal{A}} Q(s_{t+1}, a'), \tag{21}$$

where Δt is the number of time steps taken by action a_t. If the episode terminates with the system still within the capture region, we treat the state as *terminal absorbing* (Sutton and Barto (2018)), with the reward given by Eq. 21 and Δt equal to the number of steps the system remained in the capture region before the episode terminated. As such, the agent learns to enter the capture region in such a way as to maximize the performance of the LQR controller, instead of just entering the capture region itself such as in (Randløv et al. (2000)).

3.2 LQR Action

LQR capture requires the system architect to specify a capture region. If this region is not chosen optimally, the resulting controller will not be optimal. To avoid manual specification of the capture region, we may instead allow the reinforcement learning controller to select a special action which applies the action suggested by the LQR controller. Although this requires learning the return of this new action, it allows the controller to stabilize a continuous system.

For the LQR action algorithm, the action set is thus expanded to

$$\mathcal{A}^{\mathrm{A}} = \mathcal{A} \cup \{a_{\mathrm{LQR}}\}, \tag{22}$$

where the abstract action a_{LQR}, when selected, applies the LQR control. Due to the use of an abstract action, this method is only applicable to discrete action set methods such as DQN. The resulting *mixed cartesian-abstract* action set can be represented by adding an integer-valued abstract action dimension to the action space, where the value 0 indicates the use of the cartesian action space and any other value (in this case, only 1) indicates the index of the abstract action.

3.3 Integrated LQR Action

While the abstract LQR action does not share the cartesian action space with the other actions, the applied LQR control does. We may therefore instead add the LQR control as a sample point in the original action space by making it state-dependent

$$\mathcal{A}^{\mathrm{IA}}(s) = \mathcal{A} \cup \{-F\overline{s} + a_{\mathrm{ff}}\} \tag{23}$$

and taking the maxium over $\mathcal{A}^{\mathrm{IA}}(s)$ instead of over $\mathcal{A}$ in Eqs 7 and 8. The advantage of this method over an abstract LQR action is that its return may be generalized over by the Q-value representation. Taking similar (discrete) actions in the regular action space therefore also improves the estimate of the LQR action, decreasing the learning time.

In addition, we can embed the LQR action in the DDPG algorithm by taking the arg max in Eqs 7 and 8 over both the LQR action $-F\overline{s} + a_{\mathrm{ff}}$ and the DDPG action $\mu(s|\theta^{\mu'})$. While DDPG can already stabilize a dynamic system without chatter, adding a known-good (at least close to the goal state) solution might also decrease the learning time, or increase performance. To ensure sufficient exploration, we use ϵ-greedy exploration for the choice between DDPG and LQR action, and apply the exploration noise in Eq. 9 to both.

4. SIMULATIONS

We test the three described methods on three simulated testbeds: the pendulum swing-up, cart-pole swing-up and 2d flyer, using the Generic Reinforcement Learning Library, GRL [1]. The network architecture and other parameters are described in Appendix A.

[1] Code available at `https://github.com/wcaarls/grl`.

All experiments were performed 20 times to calculate the 95% confidence interval of the results, which are presented as the rise time and end performance. The rise time is defined as the first time the agent passes a system-defined cumulative episode reward consistently (3 times in a row), and the end performance is the mean of the cumulative episode reward over the last 10% of the episodes of each run.

4.1 Pendulum Swing-up

The pendulum swing-up is a classical control problem where a pendulum attached to a motor has to swing up from the stable equilibrium to the unstable equilibrium, but without enough torque to do so in one swing (Busoniu et al. (2010)). The system has two state dimensions $s = [\theta, \dot{\theta}]$ being the angle and angular velocity of the pendulum and one action dimension $a = u$ being the voltage applied to the motor, up to $3V$. To avoid the $0/2\pi$ nonlinearity, the angle θ is supplied to the networks in a sine-cosine representation. The cost matrices are $C = \mathrm{diag}([5, 0.01])$ and $D = 1$, with goal state $s_\mathrm{d} = [0, 0]$. The episode ends after 3s.

4.2 Cart-pole Swing-up

Another classical control set-up is the cart-pole swing-up (Barto et al. (1983)). In this case, the pendulum is mounted to a cart, which can be pushed along a track to perform the swing-up. As such, the system has four state dimensions $s = [x, \theta, \dot{x}, \dot{\theta}]$, now including the position and velocity of the cart. The action is still one-dimensional $a = F$, being the force applied to the cart, up to 15N. Again, the angle is presented in sine-cosine representation, and the cost matrices are $C = \mathrm{diag}(2, 1, 0.1, 0.1)$ and $D = \frac{1}{15}$, with goal state $s_\mathrm{d} = [0, 0, 0, 0]$. The episode ends after 10s.

4.3 2d Flyer

We introduce the 2d flyer as a very unstable regulation task, with additional nonlinearity in the form of an obstacle. The flyer is modeled as a rod with mass $m = 0.1$kg and length $l = 0.1$m, where two forces $[F_\mathrm{L}, F_\mathrm{R}]$ perpendicular to the rod may be applied at the tips. The equations of motion are

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} -\left(F_\mathrm{L} + F_\mathrm{R}\right) \sin\theta / m \\ \left(F_\mathrm{L} + F_\mathrm{R}\right) \cos\theta / m - g \\ \left(F_\mathrm{R} - F_\mathrm{L}\right) l / I \end{pmatrix}, \tag{24}$$

where $g = 9.81$ and the inertia $I = \frac{ml^2}{3}$. Therefore, the system has six state dimensions $s = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]$ and two action dimensions $a = [F_\mathrm{L}, F_\mathrm{R}] - [0.5, 0.5]$, up to 0.1N. The angle is represented by its sine and cosine once again, and the cost matrices are $C = \mathrm{diag}([1, 1, 1, 0, 0, 0])$ and $D = \mathrm{diag}([1, 1])$, with goal state $s_\mathrm{d} = [0, 0, 0, 0, 0, 0]$. The episode ends after 20s or if the flyer leaves the target area $[-1, -1] < [x, y] < [1, 1]$.

An obstacle occupies the region $[-0.4, -0.3] < [x, y] < [0.1, -0.2]$, which prohibits the flyer from reaching the target through simple regulation from the start location $[-0.4, -0.4]$.

5. RESULTS

The main results are presented in Table 1, with the respective learning curves given in Figure 3.

5.1 DQN

In the DQN case, the LQR integrated action algorithm (DQN-LQR-IA) consistently presented the best end performance, although it did not learn significantly faster than using an abstract action (DQN-LQR-A). All LQR-based algorithms outperform baseline DQN due to reduced chattering while maintaining the goal state, see Figure 1.

Using learned dynamics (*-LD) did not have a large effect on either rise time or end performance, indicating that the dynamics around the goal state are learned sufficiently quickly such as not to impede the learning process. In theory, learned dynamics makes the system nonstationary, as the actions of the LQR controller change during the run. The impact of this nonstationarity is different for each system. For DQN-LQR-LD, it changes the rewards received when entering the capture region (see Eq. 21), while for DNQ-LQR-A-LD it changes the controls applied by the abstract action, leading to a different next state. Finally, for DQN-LQR-IA-LD, it changes the action set used to calculate the target values. Note, however, that while these errors are stored permanently in the replay memory in the case of DQN-LQR-LD and DQN-LQR-A-LD, for DQN-LQR-IA-LD the loss in Eq. 7 is recalculated every time a new minibatch is sampled, and thus always reflects the current dynamics.

5.2 DDPG

Because DDPG already uses continuous actions, we did not expect a large performance gain from using LQR actions. Indeed, for the pendulum plain DDPG has the best end performance and is only slightly slower than the other variants. However, LQR capture (DDPG-LQR(-LD)) shows significantly improved rise time and end performance for the cart-pole swing-up problem, while LQR integrated action (DDPG-LQR-IA(-LD)) has the best rise time and end performance for the 2d flyer.

The improvements in rise time and end performance for LQR capture show that using a well-chosen capture region in which the LQR controller is optimal helps learning, even when using a continuous action algorithm such as DDPG. And the good results for LQR integrated action for the 2d flyer indicate that choosing between the LQR and DDPG actions might help guide the solution towards a better policy, although similar to (Gu et al. (2016)) the result is inconsistent across domains.

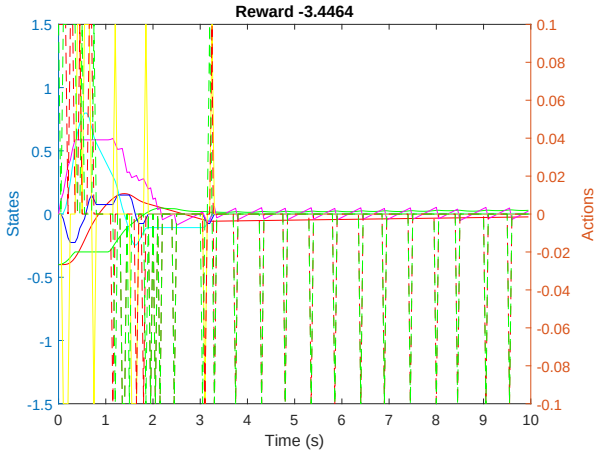
Inspecting the state evolution of individual episodes during a run (data not shown) shows that the best episode reward is comparable for all methods. Rather, it is the average end performance that is improved.

6. CONCLUSION

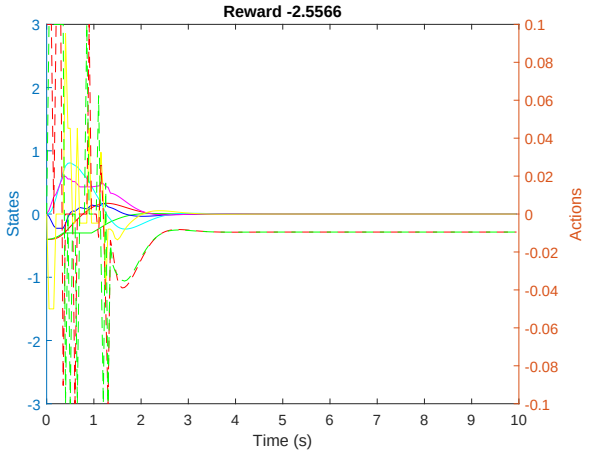
We presented a brief comparison between three different ways of using the action calculated by an optimal controller in reinforcement learning: LQR capture, LQR

Table 1. Mean and 95% confidence intervals over 20 runs for the rise time and end performance on the three testbeds. Values in bold are the best performance within that category (DQN or DDPG), or statistically equivalent to it.

	pendulum swing-up				cart-pole swing-up				2d flyer			
	rise (s)		end perf		rise (s)		end perf		rise (s)		end perf	
DQN	350	± 46	-802	± 3.5	2405	± 536	-270	± 1.7	7749	± 561	-5.4	± 0.8
DQN-LQR	313	± 40	-736	± 9.1	**985**	± 191	-232	± 5.0	7563	± 804	-6.9	± 2.0
DQN-LQR-A	285	± 31	**-714**	± 2.9	1625	± 401	-236	± 6.8	**6348**	± 747	**-4.3**	± 0.7
DQN-LQR-IA	**245**	± 36	**-713**	± 1.9	1370	± 352	**-224**	± 3.7	**7012**	± 700	**-4.3**	± 0.5
DQN-LQR-LD	308	± 42	-733	± 5.7	1210	± 337	-233	± 4.3	7103	± 442	**-4.5**	± 1.2
DQN-LQR-A-LD	320	± 52	-721	± 4.3	1900	± 410	-234	± 6.3	**6784**	± 609	-5.4	± 1.8
DQN-LQR-IA-LD	**233**	± 31	-716	± 8.3	1280	± 418	**-226**	± 5.1	7202	± 439	**-4.7**	± 0.7
DDPG	213	± 67	**-731**	± 10	1590	± 600	-244	± 8.8	10993	± 1493	-5.1	± 1.7
DDPG-LQR	**185**	± 20	**-728**	± 12	1025	± 351	-216	± 4.9	8840	± 1585	-4.6	± 1.7
DDPG-LQR-IA	**198**	± 15	**-737**	± 10	1755	± 559	-264	± 29	**5470**	± 841	**-3.4**	± 0.4
DDPG-LQR-LD	**183**	± 16	**-731**	± 16	**805**	± 124	**-214**	± 2.4	8314	± 1327	-4.9	± 1.0
DDPG-LQR-IA-LD	**198**	± 13	-754	± 22	1310	± 531	-249	± 15	**6140**	± 912	**-3.7**	± 0.7

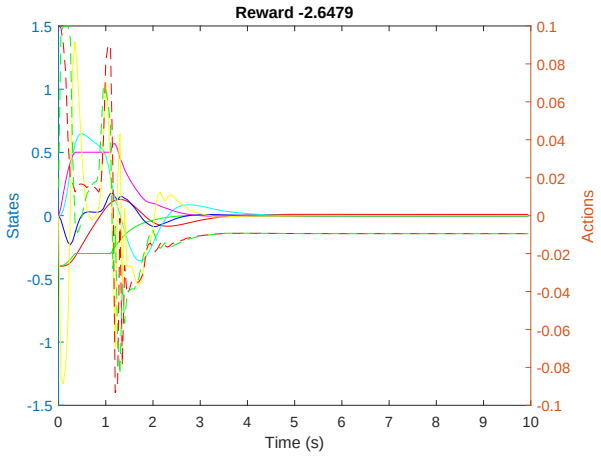


(a) 2d flyer state evolution, DQN

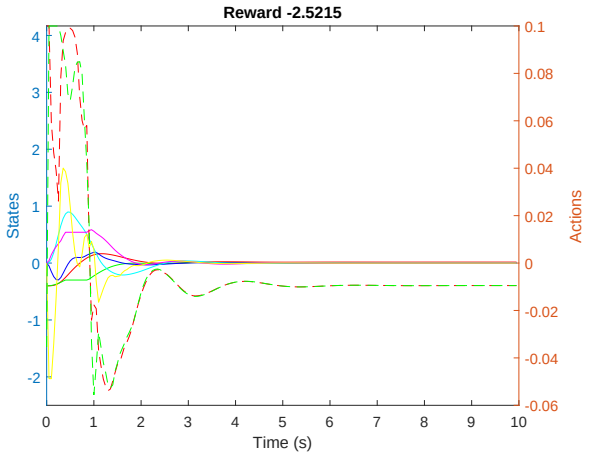


(b) 2d flyer state evolution, DQN with integrated LQR action

Fig. 1. State evolution of DQN and DQN-LQR-IA. Solid lines are state values, dashed lines are actions. DQN suffers from chattering due to its discrete action set, while DQN-LQR-IA maintains a chatter-free equilibrium.

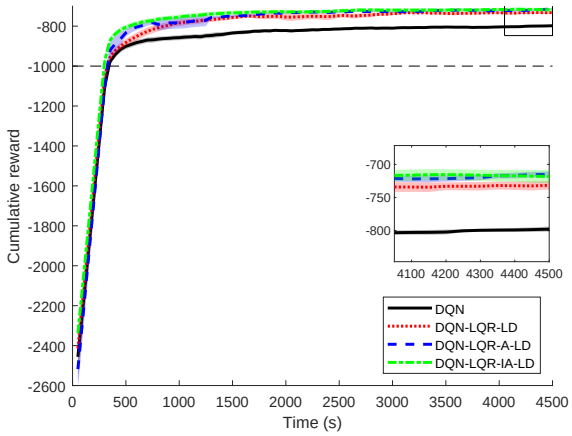


(a) 2d flyer state evolution, DDPG

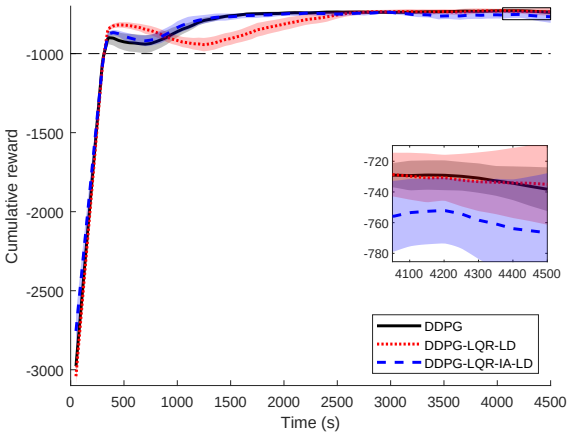


(b) 2d flyer state evolution, DDPG with integrated LQR action

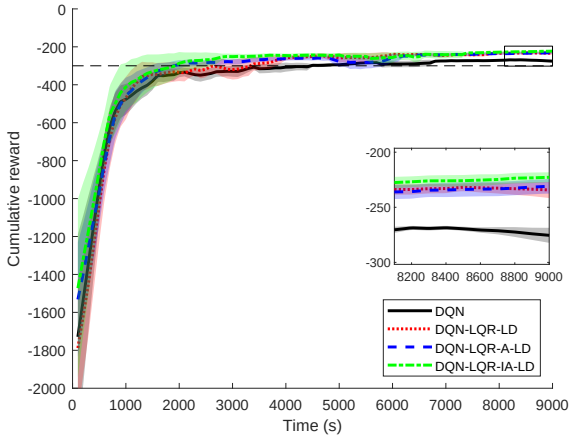
Fig. 2. State evolution of DDPG and DDPG-LQR-IA. Solid lines are state values, dashed lines are actions. DDPG-LQR-IA has a lower steady-state error.



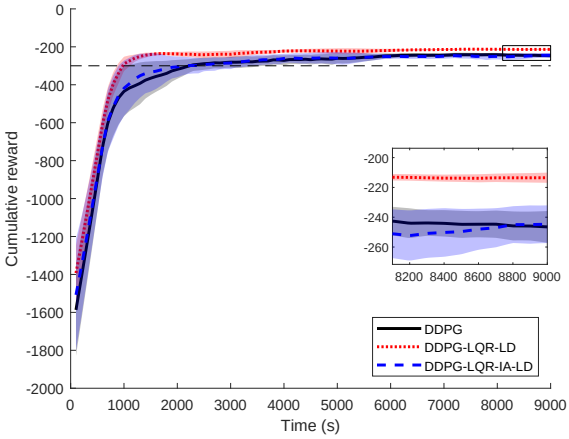
(a) Pendulum swing-up DQN



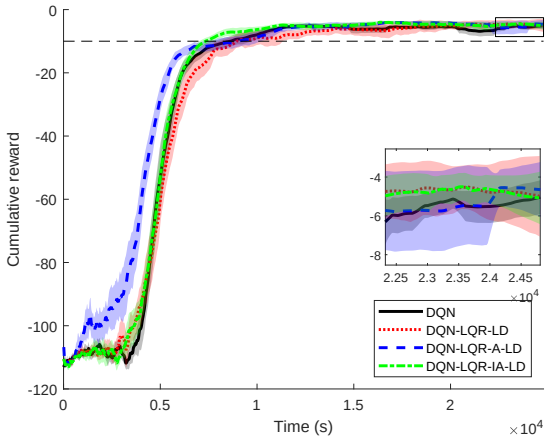
(b) Pendulum swing-up DDPG



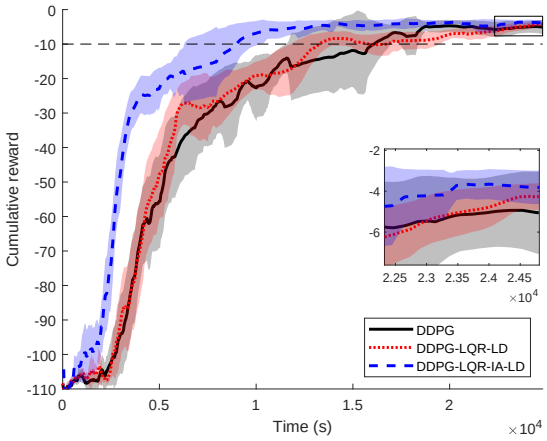
(c) Cart-pole swing-up DQN



(d) Cart-pole swing-up DDPG



(e) 2d flyer DQN



(f) 2d flyer DDPG

Fig. 3. Performance evaluation of LQR integration using learned dynamics for DQN (left column) and DDPG (right column). Shown are the mean and 95% confidence interval over 20 independent runs, plotted using a moving average filter of 10 episodes. The horizontal line is the point at which the rise time is measured.

action and LQR integrated action. When combined with DQN, all methods decreased rise time and increased end performance compared to the baseline, with the LQR integrated action algorithm having the best performance overall. For DDPG the results are less consistent, but both LQR capture and LQR integrated action showed improved performance in some test cases.

Future work includes using different (non-linear) optimal control techniques instead of LQR, which allows tackling non-regulation tasks. In this setting, it would also be interesting to compare our approach to methods that somehow combine the control output with the RL policy, such as by summing (Koryakovskiy et al. (2018)). The fact that LQR integrated action does not poison the replay memory may be particularly advantageous when integrating learning or adaptive controllers.

REFERENCES

Abramova, E., Dickens, L., Kuhn, D., and Faisal, A. (2019). Rloc: Neurobiologically inspired hierarchical reinforcement learning algorithm for continuous control of nonlinear dynamical systems. *arXiv preprint arXiv:1903.03064*.

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. (2019). Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*.

Atkeson, C.G., Moore, A.W., and Schaal, S. (1997). Locally weighted learning. *Artif. Intell. Rev.*, 11(1), 11–73.

Barto, A., Sutton, R., and Anderson, C. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.*, 13(5), 834–846.

Benner, P. and Sima, V. (2003). Solving algebraic riccati equations with slicot. In *Proceedings of the 11th Mediterranean Conference on Control & Automation MED*, volume 3, 18–20.

Busoniu, L., Babuska, R., de Schutter, B., and Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximation*. CRC Press, Boca Raton, FL.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, 2829–2838. JMLR.org. URL `http://dl.acm.org/citation.cfm?id=3045390.3045688`.

Koryakovskiy, I., Kudruss, M., Vallery, H., Babuska, R., and Caarls, W. (2018). Model-plant mismatch compensation using reinforcement learning. *IEEE Robotics and Automation Letters*, 3, 2471 – 2477.

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

Mehrmann, V.L. (1991). *The autonomous linear quadratic control problem: theory and numerical solution*, volume 163. Springer.

Meijdam, H., Plooij, M., and Caarls, W. (2013). Learning while preventing mechanical failure due to random motions. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst.*, 182–187. Tokyo, Japan.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M.A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Nemati, S., Ghassemi, M.M., and Clifford, G.D. (2016). Optimal medication dosing from suboptimal clinical examples: A deep reinforcement learning approach. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2978–2981. IEEE.

Randløv, J., Barto, A.G., and Rosenstein, M.T. (2000). Combining reinforcement learning with a local control algorithm. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, 775–782. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. URL `http://dl.acm.org/citation.cfm?id=645529.657804`.

Sutton, R.S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2), 181–211.

Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Uhlenbeck, G.E. and Ornstein, L.S. (1930). On the theory of the brownian motion. *Physical review*, 36, 823.

Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 1–5.

Appendix A. PARAMETERS

Table A.1 contains the environment-independent configuration of the DQN and DDPG algorithms, while Table A.2 contains the environment-specific parameters.

Table A.1. Algorithmic parameters

Parameter	DQN	DDPG
Hidden layers	[400, 300]	[400, 300]
Hidden layer activation	ReLU	ReLU
Output activation	linear	tanh
Exploration rate (ϵ)	0.05	
Uhlenbeck and Ornstein (1930) friction		0.15

Table A.2. Environment-specific parameters

Parameter	pendulum	cart-pole	2d flyer
State dimensions	2	4	6
Action dimensions	1	1	2
Action discretization	3	3	[3, 3]
Control time step (τ)	0.03	0.05	0.05
Timeout	3s	10s	20s
Discount rate (γ)	0.99	0.97	0.99
Exploration noise (σ)	1	5	0.01
Reward scale	0.1	0.1	1
Replay memory	∞	∞	∞
LLR neighbors (K)	64	64	64
LLR memory	10000	10000	10000