CT-DQN: Control-Tutored Deep Reinforcement Learning

Francesco De Lellis
University of Naples Federico II, Italy
FRANCESCO.DELELLIS@UNINA.IT

Marco Coraggio
Scuola Superiore Meridionale, Italy
MARCO.CORAGGIO@UNINA.IT

Giovanni Russo
University of Salerno, Italy
GIOVARUSSO@UNISA.IT

Mirco Musolesi
University College London, UK, and University of Bologna, Italy
M.MUSOLESI@UCL.AC.UK

Mario di Bernardo
University of Naples Federico II, Italy, and Scuola Superiore Meridionale, Italy
MARIO.DIBERNARDO@UNINA.IT

Abstract

One of the major challenges in Deep Reinforcement Learning for control is the need for extensive training to learn a policy. Motivated by this, we present the design of the Control-Tutored Deep Q-Networks (CT-DQN) algorithm, a Deep Reinforcement Learning algorithm that leverages a control tutor, i.e., an exogenous control law, to reduce learning time. The tutor can be designed using an approximate model of the system, without any assumption about the knowledge of the system dynamics. There is no expectation that it will be able to achieve the control objective if used standalone. During learning, the tutor occasionally suggests an action, thus partially guiding exploration. We validate our approach on three scenarios from OpenAI Gym: the inverted pendulum, lunar lander, and car racing. We demonstrate that CT-DQN is able to achieve better or equivalent data efficiency with respect to the classic function approximation solutions.

Keywords: Reinforcement learning based control, deep reinforcement learning, feedback control.

1. Introduction

The design of controllers based on training from data via Reinforcement Learning (RL) is a fascinating area, which is also increasingly gaining popularity. This paradigm is particularly suitable for scenarios in which we do not have any prior knowledge of the system dynamics (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 2018; Nian et al., 2020). At the same time, in order to deal with large state spaces, neural approximators are now widely adopted. These solutions are usually referred to as Deep Reinforcement Learning (DRL) (Hornik et al., 1989; Mnih et al., 2015; Lillicrap et al., 2019). Control algorithms based on DRL have shown impressive performance in different application fields, including the control of plasma in nuclear fusion (Degrave et al., 2022) and that of microbial cultures in bioreactors (Treloar et al., 2020). However, a critical issue for these algorithms is that they typically require extensive training. To tackle these challenges, we propose a control framework combining DRL algorithms and feedback controllers.

Indeed, in recent years, classical control theoretical tools and RL solutions have been combined in a number of ways. For example, in Rathi et al. (2021), Model Predictive Control (MPC) is
used in state-space regions where a model of the dynamics is available, while tabular Q-learning was used in the other regions. In Zanon and Gros (2021), a RL algorithm is used to vary the parameters of the model and the objective function used by a MPC. The authors of Abbeel et al. (2006) propose a policy gradient algorithm that performs updates on the policy using data generated by an approximate Markov decision process model in combination and through exploration of the environment. In Gu et al. (2016), a variant of the Q-learning algorithm (normalized advantage functions) is discussed; the authors show that their solution is able to accelerate the learning process by using local linear models fitted iteratively with exploration data. More in general, model-based RL techniques have been developed to learn the system dynamics; the model is then used to perform simulated roll-outs, which generate new data for learning (Sutton, 1991). An example is Deisenroth and Rasmussen (2011), in which model fitting is performed using Gaussian processes. However, these model-based techniques may introduce biases in the learning process as part of the data is not generated by the actual system. Conversely, in De Lellis et al. (2021, 2022) the authors propose Control-Tutored Reinforcement Learning, which relies on the introduction of a beneficial bias in the exploration process to speed up learning. This bias has the form of “suggestions” from a control law—which we call tutor—based on approximate modeling of the system dynamics. Differently from Imitation Learning (Wu et al., 2019), we do not need to have access to a large dataset of expert demonstrations. Instead we leverage a control law based on a simplified model of the system dynamics which in principle does not guarantee to fulfill the designed task for the original complete dynamics. For example, in De Lellis et al. (2022) the authors integrate a tabular Q-learning with a simple tutor designed to capture a very limited description of the dynamics with the objective of stabilizing an inverted pendulum. This approach leads to a significant reduction in the learning time, without compromising the effectiveness of the learnt policy. However, the algorithm is based on tabular RL, which is suitable only for problems with limited state and action spaces.

The key contributions of this paper can be summarized as follows. We introduce Control-Tutored Deep Q-Networks (CT-DQN), an algorithm combining DQN (Mnih et al., 2015) and control tutors. We discuss its design showing its effectiveness via numerical validations. We test our algorithm in three representative OpenAI Gym scenarios of increasing complexity to stress the performance of our approach. Then, by comparing our results to a classical DQN solution, we show that, even if the tutor is implemented through a very simple control law and designed using a rough approximation of the underlying dynamics, learning time can be reduced significantly, thus improving data efficiency of the learning process. The code used to carry the numerical simulation is available at https://github.com/FrancescoDeLellis/Control-Tutored-Reinforcement-Learning.

2. Control-Tutored Deep Q-Networks

In the following, we denote random variables by capital letters and their realization with lower case letters; \( \mathbb{E} \) is the expectation operator.

2.1. Problem Formulation

Following De Lellis et al. (2022, 2021), we consider a discrete time dynamical system affected by noise, of the form

\[
X_{k+1} = f(X_k, U_k, W_k), \quad x_0 = \bar{x}_0, \tag{1}
\]
where $k \in \mathbb{N}_{\geq 0}$ is discrete time, $X_k \in \mathcal{X}$ is the state at time $k$, with $\mathcal{X}$ being the state space, $\tilde{x}_0 \in \mathcal{X}$ is the initial condition, $U_k \in \mathcal{U}$ is the control input (or action), and $\mathcal{U}$ is the set of feasible inputs; $W_k$ is a random variable representing noise, with values in a set $\mathcal{W}$, and $f : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \rightarrow \mathcal{X}$ is the system dynamics.

We consider the problem of learning a policy $\pi : \mathcal{X} \rightarrow \mathcal{U}$ to solve the following sequential decision making problem, see e.g., (Garrabé and Russo, 2022), with finite time horizon $N \in \mathbb{N}_{>0}$:

$$\max_{\pi} \mathbb{E}[J^\pi],$$

s.t. $X_{k+1} = f(X_k, U_k, W_k), \ k \in \{0, \ldots, N - 1\}$,

$U_k = \pi(X_k), \ k \in \{0, \ldots, N - 1\}$,

$x_0$ given,

where $J^\pi = r_N(X_N) + \sum_{k=1}^{N} r(X_k, X_{k-1}, U_{k-1})$ is the cumulative reward, with $r : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ being the reward received by the learning agent when entering the next state after taking the selected action and $r_N : \mathcal{X} \rightarrow \mathbb{R}$ being the final reward.

### 2.2. Policy Design

During the learning phase, the control input $U_k$ is chosen either as the value proposed by some RL policy ($\pi^{\text{rl}} : \mathcal{X} \rightarrow \mathcal{U}$), with probability $\beta \in (0, 1)$, or as the one proposed by a control law (i.e., tutor; $\pi^{\text{c}} : \mathcal{X} \rightarrow \mathcal{U}$). Hence, $\pi$ in (2) is given by

$$\pi(x) = \begin{cases} \pi^{\text{rl}}(x), & \text{with probability } \beta, \\ \pi^{\text{c}}(x), & \text{with probability } 1 - \beta. \end{cases}$$

Next, we explain how we selected $\pi^{\text{rl}}$ and $\pi^{\text{c}}$ in (3). Specifically, $\pi^{\text{rl}}$ is learnt through an $\epsilon$-greedy DQN policy (Mnih et al., 2015). Thus, we have

$$\pi^{\text{rl}}(x) = \begin{cases} \arg \max_{u \in \mathcal{U}} Q(x, u), & \text{with probability } 1 - \epsilon^{\text{rl}}, \\ u \sim \text{rand}(\mathcal{U}), & \text{with probability } \epsilon^{\text{rl}}, \end{cases}$$

with $\epsilon^{\text{rl}} \in (0, 1)$, and $Q : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ being the state-action value function (Sutton and Barto, 2018). DQN uses Deep Neural Networks to iteratively approximate the function $Q$; it is among the most popular implementations of DRL and can be used also for continuous state spaces $\mathcal{X}$. Differently from tabular $Q$-learning (Watkins and Dayan, 1992), there are currently no guarantees of convergence towards the optimal policy for DQN (Fan et al., 2020), although its effectiveness is supported by strong empirical evidence (Mnih et al., 2015, 2013).

To select the control tutor policy $\pi^{\text{c}}$ in (3), we assume to have a feedback controller $g : \mathcal{X} \rightarrow \mathcal{U}$, designed with limited information$^1$ about the dynamical system described by (1). Then, letting $\epsilon^{\text{c}} \in (0, 1)$, we select

$$\pi^{\text{c}}(x) = \begin{cases} g(x), & \text{with probability } 1 - \epsilon^{\text{c}}, \\ u \sim \text{rand}(\mathcal{U}), & \text{with probability } \epsilon^{\text{c}}. \end{cases}$$

In conclusion, combining (3), (4) and (5), we have that action (4a) is taken with probability $\beta(1 - \epsilon^{\text{rl}})$, action (5a) is taken with probability $\omega : = (1 - \beta)(1 - \epsilon^{\text{c}})$ and the random action with probability $\beta \epsilon^{\text{rl}} + (1 - \beta) \epsilon^{\text{c}}$.

---

$^1$ See Section 4 for practical examples.
3. Metrics

In all scenarios we consider, each study is repeated in S independent sessions, each composed of E episodes, which are simulations lasting N time steps. The weights of the neural networks in DQN are carried over from one episode to the next, and re-initialized at each session. An episode can end earlier if a (scenario-specific) terminal condition is met, and we denote by $J_e^{\pi}$ the cumulative reward (see §2.1) in episode $e$. As usual, maximizing $J_e^{\pi}$ in §2.1 amounts to fulfilling some problem-specific goal: we define the goal condition $c_g$ as a Boolean variable that is true if and only if the goal is achieved in an episode. Next, we define three metrics to assess learning performance.

Definition 1 (Learning metrics) (i) The average cumulative reward is $J_{avg}^{\pi} := \frac{1}{E} \sum_{e=1}^{E} J_e^{\pi}$. (ii) The terminal episode $E_t$ is the smallest episode such that $c_g$ is true for all $e \in \{E_t - 10, \ldots, E_t\}$. (iii) The average cumulative reward after terminal episode is $J_{avg,t}^{\pi} := \frac{1}{E_t} \sum_{e=E_t}^{E} J_e^{\pi}$.

$J_{avg}^{\pi}$ is often used in RL (Duan et al., 2016; Wang et al., 2019); $E_t$ is used to assess the effective duration of the learning phase, and consequently data efficiency; $J_{avg,t}^{\pi}$ quantifies the quality of the controller, once the learning phase is completed. Next, we define three metrics inspired by those commonly used in control theory, to assess the transient and steady-state performance. Let $\pi_{greedy}(x) = \arg \max_{u \in \mathcal{U}} Q(x, u)$ be the greedy policy. Moreover, when the goal is to reach some goal state $x^* \in \mathcal{X}$ (or region containing $x^*$), we refer to it as a regulation problem.

Definition 2 (Control metrics) (i) The cumulative reward (see §2.1) obtained following $\pi_{greedy}$ is $J_{\pi_{greedy}}^{\pi}$. (ii) In an episode, the settling time $k_s$ is a time instant such that the goal is achieved or a related task is completed (defined uniquely in each scenario, when possible); (iii) in regulation problems, the steady state error is $e_s := \frac{1}{N-k_s+1} \sum_{k=k_s}^{N} \|x - x^*\|$.

4. Evaluation

We assess the performance of the CT-DQN algorithm (3)-(4)-(5) on three representative case studies from the OpenAI gym suite (OpenAI, 2022a; Brockman et al., 2016), i.e., the inverted pendulum OpenAI (2022d), lunar lander OpenAI (2022c) and car racing OpenAI (2022b). The inverted pendulum was selected as it is a classical nonlinear benchmark problem in control theory. Lunar lander was chosen as it represents a harder control problem with multiple input and outputs (MIMO) and in which certain regions of the state space must be avoided. Car racing was selected as it is a tracking problem where the state is observable as a matrix of pixels, rather than measured physical quantities.

4.1. Inverted Pendulum

Environment description and control goal. A rigid pendulum, subject to gravity, must be stabilized to its upward position. The states are the pendulum’s angular position and velocity; $\mathcal{X} = [-\pi, \pi] \times [-8, 8]$; $[0\ 0]^T$ and $[\pi\ 0]^T$ correspond to the upward unstable position and the downward stable position, respectively; the initial position is always $x_0 = [\pi, 0]^T$. The control input is a torque at the joint, with $\mathcal{U}$ being discrete. Further details are reported in OpenAI (2022d) and omitted here for brevity. We set $S = 3$, $E = 100$, and $N = 400$ (see §3). The control goal is a regulation problem, with $x^* = [0\ 0]^T$. The goal condition $c_g$ is true in an episode if $\exists k \in [0, N - 100] : \|x_k - x^*\| \leq 0.05 \|[\pi\ 8]\|, \forall k \in [k, N]$; the settling time $k_s$ is the smallest of such $k$. 


Control tutor design. Assume we know a linearized dynamics of the pendulum, approximating $f$ in (1) close to the upward equilibrium position $x^*$, namely $	ilde{f}(x_k, v_k) = Ax_k + Bv_k$, where $A = \begin{bmatrix} 0 & 1 + \frac{T}{3} \frac{g}{2l} \\ 3Tg/2l & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 0 \\ T/l \end{bmatrix}$, with $T = 0.05$ s being the sampling time, $l = 1$ m being the rod length and $I = ml^2/3$ being the moment of inertia of the rod. From this model, using a pole placement technique, we synthesize the linear feedback controller $v_k = -[5.83 \ 1.83]x_k$, which can stabilize $x^*$ only locally. Then $g(x)$ in (5) is obtained by projecting $v_k \in \mathbb{R}$ in $\mathcal{U}$ (which is discrete). Note that this controller, if used on its own, is unable to swing up the pendulum from its downward asymptotically stable position.

Numerical results. Fig. 1 shows that CT-DQN (with different values of the switching probability $\omega$) and DQN have comparable performance during the learning phase. Indeed, in Tab. 1, a Welch’s t-test reveals no statistically significant difference between the two. In Tab. 2, we report the control metrics assessed after a training of 50 episodes (larger than $E_t$ for all cases, meaning learning is considered complete), and observe similar control performance, without statistically significant differences. Hence, in this scenario, under all metrics considered, CT-DQN and DQN have comparable performance. We believe this happens because the state and action spaces are small, and DQN is already able to learn quickly, not needing additional aid from the tutor. In Sections 4.2 and 4.3, we show how the tutor can improve learning performance when the state and action spaces are larger.

4.2. Lunar Lander

Environment description and control goal. In a 2-D space, a spaceship subject to gravity, in the absence of friction, must use its thrusters to land with reduced velocity on a landing pad. The states are the coordinates and orientation of the lander, the corresponding velocities, and two Boolean variables to determine contact of the two legs with the ground. The lander has three thrusters,
on the left, on the right, and on the bottom (main) of the spacecraft. The possible (four) control inputs are the following: use only the left thruster, only the right one, the main one or no activation of any thruster. The position of the landing pad and the initial position and orientation of the lander are fixed, while the initial linear speed is random, as well as the terrain topography aside from the landing pad. The spacecraft lands correctly if it impacts on the pad with its legs at a moderate velocity, while it crashes if its body touches the ground, or lands with a velocity that is too high. Further detail can be found in OpenAI (2022c). The agent obtains a high reward for landing correctly, a large negative one for crashing, and a small negative one for consuming fuel. Following OpenAI (2022c), we set the goal condition as achieving $J^\pi \geq 200$ (i.e., $c_g$ true if $J^\pi \geq 200$). It is worth noting that this might also be seen as a regulation problem, with the objective of reaching the center of the pad ($x^*$), in the origin of the reference frame. Thus, we define the settling time $k_s$ as the instant when the spacecraft lands correctly, if it happens. Moreover, we set $S = 3$, $E = 1000$, $N = 1000$, although an episode ends immediately if the lander lands correctly or if it crashes.

**Control tutor design.** In order to design the tutor, we assume the knowledge of a simplified dynamics of the center of mass of the lander, by neglecting gravity. Indeed, its magnitude might be unknown. Namely, we approximate $f$ in (2) with the reduced order model $\hat{f}(\chi_k, v_k) = A\chi_k + Bv_k$, where $\chi_k \in \mathbb{R}^4$ is the vector containing position and velocity on the x-axis followed by position and velocity on the y-axis (in this given order); $v_k \in \mathbb{R}^2$ are the x- and y- components of the force applied by a hypothetical swivelling thruster. Noting that $\chi = 0$ corresponds to the center of the landing pad, we exploit the state-feedback control law defined as $v_k = -K\chi_k$ to stabilize asymptotically the origin, where $K \in \mathbb{R}^{2 \times 4}$. The matrices of the reduced order model are defined as follows:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$E_t$</th>
<th>$J^\pi_{\text{avg}}$</th>
<th>$J^\pi_{\text{avg},t}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inverted pendulum</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DQN</td>
<td>38 ± 3</td>
<td>-837.8 ± 90.6</td>
<td>-422.8 ± 16.9</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.01$)</td>
<td>40 ± 4</td>
<td>-906 ± 110.4</td>
<td>-397.3 ± 8.9</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.05$)</td>
<td>39 ± 4</td>
<td>-856.5 ± 111.9</td>
<td>-403.1 ± 12.5</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.1$)</td>
<td>38 ± 6</td>
<td>-858.9 ± 43.3</td>
<td>-507.1 ± 68.6</td>
</tr>
<tr>
<td><strong>Lunar lander</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DQN</td>
<td>665 ± 28</td>
<td>60.1 ± 1.9</td>
<td>231.6 ± 5.2</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.01$)</td>
<td>456 ± 29</td>
<td>135.5 ± 25.4</td>
<td>232.5 ± 0.9</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.05$)</td>
<td>324 ± 90</td>
<td>155.7 ± 25.9</td>
<td>230 ± 9.4</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.1$)</td>
<td>N.A.</td>
<td>100 ± 13.6</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>Car racing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DQN</td>
<td>-</td>
<td>363.7 ± 13.7</td>
<td>-</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.05$)</td>
<td>-</td>
<td>451.7 ± 0.4</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Learning metrics (Def. 1) for the scenarios in § 4. Means and standard deviations across sessions are reported, when $S > 1$. Values that are statistically significantly different from those of DQN are in bold (according to Welch’s t-test with $p$-value less than 0.05).
CONTROL-TUTORED DQN

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$k_e$</th>
<th>$\epsilon_e$</th>
<th>$J_{\pi_{\text{greedy}}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverted pendulum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DQN</td>
<td>66 ± 3</td>
<td>0.11 ± 0.03</td>
<td>−366.6 ± 9.1</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.01$)</td>
<td>75 ± 17</td>
<td>0.15 ± 0.03</td>
<td>−407.5 ± 65.3</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.05$)</td>
<td>70 ± 3</td>
<td>0.16 ± 0.04</td>
<td>−370.3 ± 8.9</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.1$)</td>
<td>66 ± 0.4</td>
<td>0.15 ± 0.04</td>
<td>−370.2 ± 1.2</td>
</tr>
<tr>
<td>Lunar lander</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DQN</td>
<td>N.A.</td>
<td>0.18 ± 0.11</td>
<td>−82.5 ± 10.22</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.01$)</td>
<td>431 ± 55</td>
<td>0.09 ± 0.01</td>
<td>180.2 ± 11.7</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.05$)</td>
<td>311 ± 58</td>
<td>0.16 ± 0.07</td>
<td>189.8 ± 24.1</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.1$)</td>
<td>529 ± 82</td>
<td>0.14 ± 0.09</td>
<td>156.7 ± 28.4</td>
</tr>
<tr>
<td>Car racing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DQN</td>
<td>-</td>
<td>-</td>
<td>549.2 ± 290</td>
</tr>
<tr>
<td>CT-DQN ($\omega = 0.05$)</td>
<td>-</td>
<td>-</td>
<td>728 ± 294.5</td>
</tr>
</tbody>
</table>

Table 2: Control metrics (Def. 2) for the scenarios in § 4. Means and standard deviations across sessions are reported, when $S > 1$. Values that are statistically significantly different from those of DQN are in bold (according to Welch’s t-test with $p$-value less than 0.05).

$$A = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ T/m & 0 \\ 0 & 0 \\ 0 & T/m \end{bmatrix}, \quad K = \begin{bmatrix} 470 & 474.7 & 0 & 0 \\ 0 & 0 & 470 & 474.7 \end{bmatrix}, \quad (6)$$

where $T = 0.02$ is a sampling time and $m = 10$ kg is the mass of the lander. To obtain the control tutor’s input $g(x_k)$ in (5) from $v_k(\chi_k)$, we proceed as follows. If $v_y > 0$ and $|v_y| \geq |v_x|$ (the tutor mainly suggests moving upwards), we use the thruster on the bottom; if $|v_x| > |v_y|$ and $|v_x| > 0$ (the tutor mainly suggests moving right), the thruster on the left; if $|v_x| > |v_y|$ and $v_x < 0$ (the tutor mainly suggests moving left), the thruster on the right; in the other cases, no thruster. Note that this control tutor, by itself, is unable to make the spacecraft land correctly as it has access only to a very limited amount of information on the system dynamics.

**Numerical results.** Fig. 2 shows that CT-DQN improves the learning performance with respect to DQN, reducing learning times. Notably, as reported in Tab. 1, CT-DQN with $\omega = 0.05$ requires about half as many episodes as DQN to consistently achieve the goal (see $E_t$). Also, the average cumulative reward across all episodes $J^\pi_{\text{avg}}$ of CT-DQN is more than twice that of DQN, indicating a shorter learning time. Then, after both algorithms reach their terminal episode $E_t$, they exhibit comparable average cumulative reward $J^\pi_{\text{avg,t}}$. In Tab. 2 we compare the control strategies $\pi_{\text{greedy}}$ obtained from CT-DQN and DQN by halting their training at 500 episodes (after 500 episodes, CT-DQN already converged, as its $E_t < 500$, while DQN has not, as its $E_t > 500$). The DQN agent could not learn how to land yet, but keeps hovering over the landing pad, wasting fuel. This
Figure 2: Cumulative reward per episode $J_\pi^e$ for the lunar lander problem. The reward curves were averaged with a moving window of 100 samples taken on the left. Then mean (solid curves) and standard deviations (shaded areas) are calculated across sessions.

is captured by the negative cumulative reward $J_\pi^\text{greedy}$, coupled with a low steady state error $e_s$, and the settling time $k_s$ being not available. On the other hand, the CT-DQN agent has already learnt how to land, even with different values of $\omega$ (introduced in § 2.2), displaying a positive $J_\pi^\text{greedy}$, a finite $k_s$, and a low $e_s$. We note also that, as the tutor is synthesized with only a partial model of the system dynamics, performance might start to degrade when the tutor is used too often. As evidence, see the asymptotic value of the reward curves of CT-DQN with $\omega = 0.1$ in Figs. 1 and 2.

4.3. Car Racing

Environment description and control goal. In a 2-D space, a car must complete a random track as fast as possible. The state is composed of the pixel matrices of three consecutive image frames. The actions are the possible combinations of “steer left/right”, “accelerate”, and “brake”, all by a fixed amount. The agent is rewarded positively each time it covers an additional stretch of the road, and receives a small negative reward when time steps pass. Further details are reported in OpenAI (2022b). In this case, rather than defining a specific goal that can or cannot be satisfied in an episode, we deem more natural to consider the task just as that of maximizing the reward; therefore, the only metric we consider are the cumulative rewards $J_\pi^\text{avg}$ and $J_\pi^\text{greedy}$. We set $S = 2$, $E = 500$ and $N = 1000$, although an episode can end earlier if the car gets too far from the track or visits 95% of the track.

Control tutor design. The tutor regulates acceleration and steering separately. Steering is regulated as follows. First, we note that the car is still in each frame, is oriented upwards, and has its center of mass at position $p_c = [x_c \ y_c]^\top = [0 \ 0]^\top$ (in pixels) (Fig. 3). Moreover, we detect the margins of the road by processing each image frame with a Roberts operator (Davis, 1975). Next, we consider a point in front of the car, with position $p_h = [x_h \ y_h]^\top = [x_c \ y_c + l_p]^\top$, with $l_p \in \mathbb{R}_{>0}$ (see Fig. 3). We also consider two horizontal lines at $y_h + \Delta y$ and $y_h - \Delta y$, where $\Delta y \in \mathbb{R}_{>0}$. 
Normally, these lines will intersect the margins of the road in four points (see again Fig. 3), and we define $v_{\text{road}}$ as the vector between the intersection points on the side of the road closer to $p_h$. Let $\theta := \angle(v_{\text{car}} - v_{\text{road}})$ be the angle of road with respect to the car. Then, to align the car with the road, if $\theta < 0$ (resp. $\theta > 0$), the tutor suggests to steer left (resp. right). However, if all the intersection points are on one side with respect to $x_c$, or if less than four intersection points are found, it is inferred that the car is off the road, and $v_{\text{road}}$ is defined as the vector from $p_c$ to the closest intersection point, instead (see Fig. 3.(b)).\(^2\) To regulate the speed $s$, first we detect $s$ by measuring an indicator bar printed on the image frame. Then, setting some thresholds $\eta_{\text{speed}}^{\text{acc}}, \eta_{\text{angle}}^{\text{acc}}, \eta_{\text{angle}}^{\text{brk}} \in \mathbb{R}_{>0}$, the tutor suggests to accelerate if $s < \eta_{\text{speed}}^{\text{acc}}$ and $|\theta| < \eta_{\text{angle}}^{\text{acc}}$; conversely, it suggests to brake if $|\theta| > \eta_{\text{angle}}^{\text{brk}}$.

**Numerical results.** Fig. 4 shows a generally faster learning for CT-DQN, as the cumulative reward is higher for almost the entire session. This is also confirmed in Tab. 1 by the larger value of $J_{\pi}^{\text{avg}}$ for CT-DQN. Additionally, we test the greedy policies obtained after training for 250 episodes, on 30 tracks generated randomly (the same tracks for both algorithms). We find significantly higher rewards for CT-DQN (see $J_{\pi}^{\text{greedy}}$ in Tab. 2), showing the benefit of using the control tutor.

### 5. Conclusions

In this paper, we have presented Control-Tutored DQN (CT-DQN), a solution based on the integration of DRL algorithms with a tutor mechanism for aiding exploration based on control theory. In particular, we have discussed the design of an extended version of DQN, where actions are sometimes suggested by a controller (tutor). In order to evaluate our approach, we have considered three representative scenarios of increasing complexity, i.e, the stabilization of an inverted pendulum, the landing of a spacecraft (i.e., lunar lander), and the control of a racing car. In all the cases, we have considered tutors that rely on simple mechanisms and are designed with limited information about the systems dynamics.

We have shown that their addition always proved to be non-pejorative (with the inverted pendulum) or significantly beneficial (with the lunar lander and racing car) in terms of shorter learning time. We have also observed that we are able to obtain better policies with respect to classical DQN in the same number of episodes. Moreover, the better the tutor is at solving a problem (according

---

\(^2\) More complicated situations might exist, e.g., where less than four intersection points are found, but the car is on the road; however, these are typically infrequent. We also do not aim to build the best possible tutor, but a simple one that is able to demonstrate the potential of the approach. Tutors that are able to provide the learning process with more accurate suggestions will lead to better performance. In a sense, a simple tutor can be considered a baseline over which improvements are possible.
to case-specific metrics), the larger the improvement tends to be. Our future agenda includes the formal analysis of the design of the tutor mechanism for Deep Reinforcement Learning, e.g., the quantification of information and definition of bounds (e.g., regret bounds).

Appendix A. Hyperparameters Tuning

During training, we use a target neural network (Mnih et al., 2015) which is updated at the end of every episode. We also introduce a replay buffer with size $N_b$, which is used to randomly sample 64 data-points to update the network parameters at every step. Moreover, we set the learning rate $\alpha = 0.001$ and the discount factor $\gamma = 0.99$ (Sutton and Barto, 2018). With respect to the inverted pendulum described in Section 4.1, for the neural networks in DQN, we use 2 hidden layers with rectifier linear unit activation functions (ReLu), with 128 and 64 nodes, respectively. We set $N_b = 1,000,000$ and $\epsilon_{rl} = 0.02$.

As far as the lunar lander in Section 4.2 is concerned, for the neural networks in DQN, we use 2 hidden layers of 128 nodes with ReLu. Moreover, we set $N_b = 1,000,000$, $\alpha = 0.0001$, $\gamma = 0.99$, and $\epsilon_{rl} = 0.1$. Finally for the car racing discussed in Section 4.3, we use convolutional neural networks. The input has dimension $94 \times 94 \times 3$. A first hidden layer convolves 6 filters of $7 \times 7$ with stride 3 with the input image, with ReLu. A second hidden layer convolves 12 filters of $4 \times 4$ with stride 1, with ReLu. A third hidden layer is present, with 216 nodes and ReLu. The output layer is a fully-connected linear layer with a single output for each possible action. Finally, we set $N_b = 5,000$, $\alpha = 0.001$, $\gamma = 0.9999$, and $\epsilon_{rl} = 0.1$, and select $l_p = 10$ pixels, $\Delta y = 2$ pixels, $\eta_{\text{acc}} = 15^\circ$, $\eta_{\text{brk}} = 50^\circ$, and $\eta_{\text{speed}}$ as 40% of the maximum possible speed. These values are selected as a representative scenario for this type of games.
References


OpenAI. *OpenAI Gym online documentation*, 2022a. URL https://www.gymlibrary.dev/.


Acknowledgments

This work was in part supported by the Research Project SHARESPACE funded by the European Union (EU HORIZON-CL4-2022-HUMAN-01-14. SHARESPACE. GA 101092889 - http://sharespace.eu)