Teacher-Student Framework: a Reinforcement Learning Approach

Matthieu Zimmer, Paolo Viappiani, Paul Weng

Teacher-Student Framework: A Reinforcement Learning Approach

Matthieu Zimmer[1,2], Paolo Viappiani[1,2], and Paul Weng[1,2]

(1) Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6
(2) CNRS, UMR 7606, LIP6, F-75005, Paris, France
{matthieu.zimmer,paolo.viappiani,paul.weng}@lip6.fr

Abstract. We propose a reinforcement learning approach to learning to teach. Following Torrey and Taylor's framework [18], an agent (the "teacher") advises another one (the "student") by suggesting actions the latter should take while learning a specific task in a sequential decision problem; the teacher is limited by a "budget" (the number of times such advice can be given). Our approach assumes a principled decision-theoretic setting; both the student and the teacher are modeled as reinforcement learning agents. We provide experimental results with the Mountain car domain, showing how our approach outperforms the heuristics proposed by Torrey and Taylor [18]. Moreover, we propose a technique for a student to take into account advice more efficiently and we experimentally show that performances are improved in Torrey and Taylor's setting.

Keywords: reinforcement learning, reinforcement learning with rich feedback, teaching on a budget

1 Introduction

Reinforcement learning (RL) [10] is a framework for solving sequential decision problems where an agent interacts with the environment and adapt her policy taking into account a numerical reward signal. RL agents can autonomously learn somewhat difficult tasks, like navigating a maze or playing a video game. While the basic setting of RL is now well established, recently a number of researchers have been studying variants where agents have access to demonstrations of successful task completion (learning by demonstration or by imitation [3, 4]), or where the knowledge acquired for a particular task needs to be used to solve a similar related task (transfer learning [16]) or other settings where agents interacts with each other. The common ground of these works is that learning can be faster by exploiting additional information for the task at hand; this intuition goes back to the idea behind *reward shaping* [13, 11].

In this paper, we consider an agent charged with the task of teaching another agent how to perform a particular task. The idea of integrating teaching in RL is not new (e.g., [12, 17, 9]). Recently, a number of researchers have proposed similar settings but with different assumptions and communication protocols [6, 1, 7,

2]. However, in this literature, the teacher is always assumed to be a human. We assume here that the teacher is an artificial agent. This teaching problem was studied by Torrey and Taylor [18] who introduced a teacher-student framework for reinforcement learning and proposed a number of simple heuristic methods for choosing in which situations to provide advice (the best action) to the student. An important assumption in this framework is that a "budget" (maximum number of suggestions that can be given) is fixed, and therefore it is really important to carefully decide when (and which) advice to give. The constraint of a limited budget for teaching is justified in two situations [18]. In some problems, communication is constrained and limited (e.g., cost of communication), even if all agents are artificial. This framework could also be extended to human students and in this case, obviously the number of advice has to be small.

The setting in this paper assumes that the teacher (who knows the optimal policy) observes the student (her state) but cannot monitor nor change anything internal to the student. Communications rely on a very simple protocol: at each step of an episode, the student announces her intended action and the teacher can decide whether to provide some advice. A piece of advice consists in suggesting the action that the student should do (according to the optimal policy, known to the teacher but not to the student). In this paper, we try to answer two questions. From the teacher's point of view, how advice can be given efficiently? From the students' point of view, how can a piece of advice be efficiently exploited (in the RL setting)?

In this paper we assume a principled decision-theoretic approach where teaching is itself formulated as a reinforcement learning problem. Our intuition is that in this way we can provide efficient techniques for choosing the right moment at which advice should be given. For instance, the teacher should be able to autonomously learn that a student who chooses a good enough action does not need advice, or that most of the advice (but not all) will be needed in the first episodes of training, and that in some states it is more critical to perform a good action than in others. Moreover, with our approach, the teacher could adapt to particular types of learners and overcome differences in state representation.

2 Reinforcement Learning

Reinforcement learning (RL) [15] is a framework that models sequential decision problems where an agent learns to make better decisions while interacting with the environment. After an agent performs an action, the state changes and the agent receives a numerical value, called *reward*, that encodes "local" information about the quality of the action just taken. The goal of the agent is to maximize her long-term expected total reward. Because it is possible that actions associated with low reward will allow the agent to reach high-reward states, the agent needs to estimate somehow (by trial-and-error) the value of making an action in a given state.

The underlying formalism of RL is that of *Markov Decision Processes* (MDPs). An MDP is formally defined as a tuple $\langle S, A, T, R \rangle$ where S is a set of states, A

a set of actions, $T : S \times A \times S \rightarrow [0, 1]$ are transition probabilities between states ($T(s, a, s') = p(s'|a, s)$ is the probability of reaching state s' from state s after executing action a) and $R : S \times A \rightarrow \mathbb{R}$ is a reward signal. A (deterministic) *policy* $\pi : S \rightarrow A$ is a mapping from states to action (the action to be taken in each state) and encodes how the agent will behave. The *optimal* policy π^* is the policy maximizing the expected total reward, that is (for episodic reinforcement learning with a horizon T_{max}):

$$\pi^* = \arg\max_{\pi} E[\sum_{t=0}^{T_{max}} R(s_t, \pi_t(s_t))] \quad (1)$$

where t denotes a time step and T_{max} the maximum number of steps. In MDPs, since T and R are given, an optimal policy π^* can be computed offline using dynamic programming (e.g., value iteration, policy iteration) [8] for instance.

In the RL setting, the agent however does not know the transition probabilities T and the reward function R. There are two main approaches for learning an optimal policy in RL: model-based approach and model-free approach. While *model-based* methods aim at learning T and R, *model-free methods* are computationally more viable techniques that perform online learning storing an estimation of $Q : S \times A \rightarrow \mathbb{R}$, the (sequential) values of actions in each state. Whenever an action is taken and a reward is observed, the agent updates the value of $Q(s, a)$ using the following formula:

$$Q(s,a) = Q(s,a) + \alpha[\underbrace{R(s,a) - Q(s,a)}_{\text{adjustment given observed reward}} + \underbrace{\gamma Q(s_{t+1}, a_{t+1})}_{\text{estimation of future rewards}}] \quad (2)$$

where $\alpha \in]0, 1[$ is the learning rate (empirically, if the task is highly stochastic α needs to be low, alternatively it can be set to a high value and then the agent might be able to learn faster) and γ the discount rate.

In RL the agent faces the *exploitation versus exploration* dilemma, that means choosing whether to focus on high reward actions (but at the cost of ignoring potentially more valuable regions of the world) or performing actions with the intent of exploring other regions (but at the cost of giving up reward in the short term). One simple strategy to solve this tension is ϵ-greedy, that exploits most of the time, but with a small probability performs a (random) exploration move, i.e., $\pi(s) = \arg\max_a Q(s, a)$ with probability $1-\epsilon$ and $\pi(s)$ is a random uniformly drawn action in A with probability ϵ. To determine a good value for ϵ one often relies on experimental observations. Among the most common algorithms for reinforcement learning are Sarsa and Q-learning [15].

Notice however that in many tasks, the number of states and actions are too large so that the agent cannot store Q-values in a table. The problem of continuous states can be tackled with discretization; however the number of states will often explode. Even with the memory and computational power to handle very large tables, reinforcement learning with Q values represented in the tabular form suffers from the lack of generalization, meaning that what is learned in a state does not impact our knowledge about the value of similar states. A

solution to these problems is to use an approximation function, that provide a means for both efficient computation and support for generalization. The Q values is approximated by a function $\psi_\theta : S \times A \to \mathbb{R}$, i.e., $Q(s,a) \approx \psi_\theta(s,a)$ parametrized by a vector θ whose values are updated whenever receiving reward. In this way, we can represent Q-values for continuous domains. Moreover this allows generalizing what it is learned in a state to a different, but similar, state (notice that ψ_θ can take many forms of supervised learning methods: neural networks, linear regression, mixture of Gaussians, ...). In this paper, we assume that the Q-values are expressed as a linear approximation function

$$Q(s,a) \approx \psi_\theta(s,a) = \sum_i \theta_i \times f_i(s,a) \tag{3}$$

where the functions $f_i : S \times A \to \mathbb{R}$ are given basis functions.

3 Teaching on a Budget

Torrey and Taylor [18] introduced a teacher-student framework for reinforcement learning. In their framework, a student agent learns to perform a task through reinforcement learning, while a teacher agent may suggest an action in order to help the student learn faster. A few assumptions are made in this framework. First of all, the teacher is also a RL learner and has already learned an optimal policy. Moreover, the teacher and the student share a common action set. Besides, the teacher can only give a limited number of advice, called *budget*. Finally, the only means of communication between the two agents consists in the possibility for one agent to communicate an action to the other. More specifically, the student announces the action that it is about to perform (before actually performing it) and then the teacher may decide to give advice, in the form of an action that the student should perform instead (an action better than the announced one).

Formally, the framework can be described as follows. Given a task to be learned by the student agent, let $\mathcal{M}_{task} = \langle S_{task}, A_{task}, T_{task}, R_{task} \rangle$ be an MDP representing it. The state representation of the student is denoted by $\mathcal{R}_{student}$. The state representation of the teacher, which is not necessarily identical to $\mathcal{R}_{student}$, is denoted by $\mathcal{R}_{teacher}$. Typically, $\mathcal{R}_{student}$ and $\mathcal{R}_{teacher}$ are defined by the shape of the tiles and their numbers; they can be viewed as "how the agents perceive the environment".

The student starts in an initial state and learns through trial and error (i.e., with any RL algorithm). We call *episode* a sequence of time steps from an initial state to a goal state if the student manages to reach it or when a fixed maximum number of steps is reached. When an episode ends, the student starts over in a new initial state. We call *session* the sequence of episodes before the student learns a good enough policy.

During a student learning session, the teacher may choose to advise the student (i.e., communicate the best action according to the teacher's optimal policy to take in the current state of the student). The teacher can decide to spend her

budget (i.e., the limited number of advice) through the whole learning session, not only on one episode. Torrey and Taylor propose the following heuristic methods (where advice is given if a condition is satisfied) for the teacher to decide to give advice or not. We review them below:

Early advising This very simple strategy consists in the teacher spending all her budget at the very beginning of the session, providing advice at each step until the budget is over. In this strategy, the student agent does not need to announce the action that it is going to perform.

Importance advising This strategy consists in giving advice only in states that are deemed "important". For evaluating the importance $I(s)$ of a state s, Torrey and Taylor [18] adopt the following measure (initially proposed by Clouse [5]): $I(s) = \max_a Q(s,a) - \min_a Q(s,a)$. Then, at any timestep of a session, until the whole budget is spent, advice is given only if the importance $I(s)$, of the state s the student is currently in, is greater than a given (empirically optimized) threshold. In this strategy, the student does not announce her action.

Mistake Correcting In the previous heuristics, the teacher wastes her budget on situations where the student would have chosen on her own the recommended action. Denote π^* the optimal policy learned by the teacher. Mistake correcting consists in giving advice only if $\pi^*(s) \neq a_{student}$ and the state importance $I(s)$ is greater than a threshold.

Actually, Torrey and Taylor proposed another heuristic, Predictive Advising, extending Mistake Correcting, where the student does not need to announce her action. Using a classifier (in their implementation, a support vector machine) the teacher learns to predict what action the student is likely to do next.

While these heuristics have proved successful in experiments, in the following we show that by modeling the decision of when to give advice in a principled decision-theoretic way (as a sequential decision problem), we can provide more effective teaching agents. In the next section, we present how we model the teaching task as a RL problem.

4 Learning to Teach

There are two main components in our approach. First, we model the teaching task as a reinforcement learning problem (Section 4.1), i.e., the teacher agent needs to learn a policy for deciding when to give advice. Doing so, we expect that the teacher can learn a better policy for giving advice than those based on the previous heuristics. Second, the student can also exploit more efficiently the pieces of advice given by the teacher. We present a technique for improving the estimation of the learner's Q-values in order to be consistent with the information provided by the teacher (Section 4.2). Note however that such a technique can be used independently from the policy used by the teacher.

4.1 The Teacher Model

In our approach, the teacher learns how to teach RL students efficiently, without resorting to the previous heuristics, using reinforcement learning as well. As in the teacher-student framework proposed by Torrey and Taylor [18], the teacher first learns an optimal policy π^* for $\mathcal{M}_{task}$ in her state representation $\mathcal{R}_{teacher}$. Then, the teacher tries to learn a teaching policy, which would solve the same problem as the heuristics methods recalled in Section 3.

Let $\mathcal{M}_{teacher} = \langle S_{teacher}, A_{teacher}, T_{teacher}, R_{teacher} \rangle$ be a MDP describing the teaching task. The set of states $S_{teacher}$ is defined by $S_{task} \times A_{task} \times \{0, 1, \ldots, b_{\max}\} \times \mathbb{N}$ where $b_{\max}$ is the initial budget: the current state in S_{task} of the student, the action in A_{task} announced by the student, a remaining budget and a learning episode number of the student. Note that the state of the student is observed in the teacher's state representation $\mathcal{R}_{teacher}$. Recall that it may differ from that of the student $\mathcal{R}_{student}$. An action is defined as an element of $A_{teacher} = \{true, false\}$, telling whether an advice is given or not. If the teacher decides to give advice for a state s, he sends $\pi^*(s)$ to the student. The reward function defined for $\mathcal{M}_{teacher}$ is

$$R_{teacher}(s, a) = \begin{cases} r_{\max} - \frac{n_{step}}{r_d} & \text{if student reached goal} \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

where n_{step} is the number of time steps the student needed to reach a goal state, positive constant $r_{\max}$ is the greatest reward obtainable in $\mathcal{M}_{task}$ and r_d is a positive constant such that the maximum number of time steps of an episode divided by r_d is greater than $r_{\max}$. The definition of this reward function is motivated as follows. The longer it takes for the student to reach her goal, the less the teacher is rewarded. Thus, this implies that the teacher will try to minimize the number of learning time steps for the student. Note that, due to r_d, the teacher is always better off when the student reaches a goal state, however long it takes.

We call *session* one episode of this MDP, i.e., a full learning session of one student over $\mathcal{M}_{task}$. Through several sessions, the teacher experiments on multiple learners.

4.2 Effect of Advice

In their paper, Torrey and Taylor [18] considers a piece of advice as an *informed exploration* : the recommended action is performed (instead of the announced action) as if it was chosen by the ϵ-greedy strategy, and the Q-value is updated using the usual formula. This simple strategy has the advantage that the modification of the student's RL algorithm is minimal. However, to make it effective, all $Q(s, a)$'s need to be initialized in a pessimistic way, i.e. at a very low negative value. Doing so guarantees that a recommended action performed once has a greater chance of being chosen again as the best action when visiting the same state in the future.

In this section, we propose a more sophisticated strategy for the student to take into account a recommended action. This new approach will change more the student RL algorithm, but at the benefit of better estimations of Q-values. In this approach, called *max update*, we want the student to exploit a piece of advice not only once, but several times if needed. To that aim, we endow the student with a memory of the previous recommended actions and the corresponding states (of course, in the student state representation). We present the approach in the tabular version first to make it easier to understand, then we extend it to the case where approximation functions are used.

The first time an action a_r is recommended in a state s_r, the student performs it like in the informed exploration approach. Although the student does not know yet the correct value for $Q(s_r, a_r)$, it nonetheless knows that the following property shall be kept true:

$$\forall a \in A, Q(s_r, a_r) \geq Q(s_r, a) \tag{5}$$

If this property is not satisfied for s_r, then there exists an action $a_{\max}$, different from a_r, such that $a_{\max} = \arg\max_a Q(s_r, a)$. The naïve update where $Q(s_r, a_r)$ is set to the value of $Q(s_r, a_{\max})$ may not work as there is no guarantee that $Q(s_r, a_{\max})$ has already converged and that it may not be a good approximation of the true Q-value. Instead, we propose to make the following update:

$$Q(s_r, a_r) = Q(s_r, a_r) + \alpha \times (Q(s_r, a_{\max}) - Q(s_r, a_r)) \tag{6}$$

at every time steps. These updates help $Q(s_r, a_r)$ converge faster towards the target value $Q(s_r, a_{\max})$ as their aim is to minimize the quadratic difference of the two Q-values. Besides, as after each time step in the student RL algorithm, the estimation of $Q(s_r, a_{\max})$ may improve, the updates of Equation 6 may use better and better approximations. If at any time, the property described by Equation 5 is satisfied, these updates are not needed anymore and they are stopped.

For the version using a function approximation (Equation 3), the update needs to be performed on parameter θ. A component i of θ can be updated with the following formula

$$\theta_i = \theta_i + \alpha \sum_j \theta_j (f_j(s_r, a_{\max}) - f_j(s_r, a_r))(f_i(s_r, a_{\max}) - f_i(s_r, a_r)). \tag{7}$$

This is a standard gradient descent for minimizing the quadratic difference of $Q(s_r, a_r)$ and $Q(s_r, a_{\max})$.

We believe that this strategy is efficient in both settings (tabular RL and function approximations) and in Section 5.4 we indeed show that can boost the performance of a teacher. Another advantage of this new method is that we do not need to initialize the $Q(s, a)$ to a very low value (pessimistic initialization), since the property of Equation 5 is ensured by a series of specific updates (Equation 6 and Equation 13); however this comes at the expenses of additional computation at each step of the execution.

5 Experiments

We evaluate our two strategies comparing it to the heuristic methods proposed by Torrey and Taylor [18]. In the next three subsections, we recall the mountain car domain [14], a popular benchmark for reinforcement learning and give some details about the implementations of our two approaches. Then, we present our experimental results in the last subsection.

5.1 Mountain Car

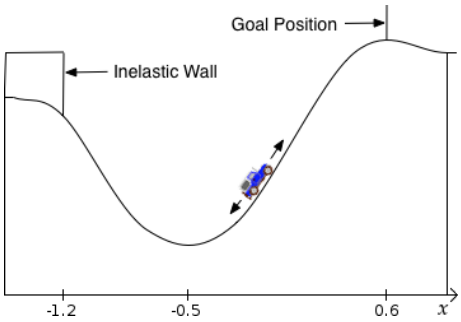

Fig. 1. The Mountain Car Problem

A standard testing domain in reinforcement learning is the mountain car problem [14]: an under-powered car must drive up a steep hill (Figure 1). The gravity is stronger than the car's engine so, even at full throttle, the car cannot simply accelerate up the steep slope; the agent must learn to gain momentum as on a swing. The state is an element of a 2 dimensional continuous space:

$$(x, y) \in Position \times Velocity \; with \begin{cases} Position = [-1.2, 0.6] \\ Velocity = [-0.07, 0.07] \end{cases} \tag{8}$$

The actions are valued in a one dimensional discrete space: $a \in \{-1, 0, +1\}$ corresponding to the directions of the acceleration, i.e., respectively backward, no acceleration and forward. The reward for every step is -1 except when the goal is reached (reward is 0). The transitions are defined as follows:

$$\begin{cases} y_{t+1} = y_t + a \times 0.001 + cos(3 \times x_t) \times -0.0025 \\ x_{t+1} = x_t + y_t \end{cases} \tag{9}$$

If the position or the velocity go out of the bounds, their values are set at the extreme bound of the respective domains. For the position, this corresponds to an inelastic wall on the left hand side and the goal on the right hand side. The

car begins each episode with zero velocity at the bottom of the mountain (initial position $(x_0, y_0) = (-0.5, 0)$), and the episode ends when it reaches the goal at the top (whenever $x \geq 0.6$), or after 500 steps.

Since the state space is continuous, we rely on a function approximation of Q, with the common approach of using a linear approximation function with a binary tile coding:

$$Q(s,a) \approx \psi_\theta(s,a) = \sum_i \theta_i \times f_i(s,a) \tag{10}$$

where $f_i : S \times A \to \{0,1\}$ with the semantics that a feature f_i is "active" only in some states (and therefore parameter θ_i only impacts the computation of Q for some regions of the world). We denote by F the set of features that are active for a state-action pair, i.e., $F(s,a) = \{i \,|\, f_i(s,a) = 1\}$. In our implementation, we assume a common tiling structure (of rectangular shape) for all actions and, moreover, a feature can only depend on one action at a time (i.e., for a given state s and two different actions a and a', the number of active features is the same, while their intersection is empty). More formally, the two following properties are satisfied:

$$(i) \quad \forall s \in S, \forall a, a' \in A, a \neq a' \Rightarrow \underbrace{(F(s,a) \cap F(s,a') = \emptyset)}_{\text{independence}} \wedge (|F(s,a)| = |F(s,a')|)$$

$$(ii) \quad \forall a, a' \in A, \forall f_i, \exists f_j, \{s \in S \,|\, f_i(s,a) = 1\} = \{s \in S \,\|\, f_j(s,a') = 1\}.$$

A common way of increasing the learning speed is to keep a trace of the last updated pairs (s,a). In our implementation we use an accumulative trace:

$$e_t(s,a) = \begin{cases} \gamma \times \lambda \times e_{t-1}(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t \\ 1 + e_{t-1}(s,a) & \text{otherwise} \end{cases} \tag{11}$$

where $\lambda \in [0,1]$ is the size of the trace, γ is the discount factor. The update step will adjust the Q-value of all pairs (s,a) for which $e_t(s,a) \neq 0$, using the formula:

$$Q(s,a) = Q(s,a) + \alpha \times [R(s,a) - Q(s,a) + \gamma \times \ Q(s_{t+1}, a_{t+1})] \times e_t(s,a) \tag{12}$$

The algorithm used for students and teachers is Sarsa(λ). In our simulations, the different reinforcement learning parameters for the student are $\epsilon = 0.05$, $\alpha = 0.08$, $\lambda = 0.9$, $\gamma = 1$ and those for the teacher are $\epsilon = 0.001$, $\alpha = 0.02$, $\lambda = 0.99$, $\gamma = 0.99$. For $\mathcal{R}_{student}$, the number of tilings is 8 and the number of segments in one tiling is 16×16.

5.2 Implementation of the RL teacher

The state of the teacher is represented by families of independent features. Recall that the teacher state $s_{teacher} = (s_{student}, a_{student}, b, n_e)$ is a tuple containing the student's current state $s_{student}$ in S_{task}, the student's declared action $a_{student}$ in A_{task}, the remaining budget b and the learning episode number n_e. The number of tilings is set to 8 for each component of $s_{teacher}$. The families of features are:

- A set of features $g_i(s_{student}, a_{teacher})$ jointly representing the student's position and her velocity and a teacher's action $a_{teacher}$ corresponding to the decision of advising or not. The tiling structure corresponding to the student's state is qualitatively similar to the student's state representation (but the actual positions of the tilings differ from those of the student's own representation). The number of g_i's is therefore $n_g = 16 \times 16 \times 8 \times 2 = 4096$.
- A set of features $h_i(a_{student}, a_{teacher})$ representing the action $a_{student}$ announced by the student (backward, no acceleration,forward). There are therefore $n_h = 3 \times 8 \times 2 = 48$ of features of this type.
- A set of features $m_i(b, a_{teacher})$ representing the available budget (the maximum allowed budget $b_{\max}$ is discretized into $\lfloor b_{\max}/3 \rfloor$ slots). For the m_i's, the number of features is $n_m = \lfloor b_{\max}/3 \rfloor \times 8 \times 2$.
- A set of features $l_i(n_e, a_{teacher})$ that discretize the number of episodes into $\lfloor n_{\max}/3 \rfloor$ subintervals (to have a fine-grained representation of the temporal position of the current episode in the training session) where $n_{\max}$ is the maximum allowed number of episodes of a session. There are $n_l = \lfloor n_{\max}/3 \rfloor \times 8 \times 2$ features of type l_i.

Thus, the functional approximation of $Q(s, a)$ for the teacher is computed as

$$Q(s_{teacher}, a_{teacher}) \approx \psi_\theta(s_{teacher}, a_{teacher})$$

where

$$\psi_\theta(s_{teacher}, a_{teacher}) = \sum_{i=1}^{n_g} g_i(s_{student}, a_{teacher})\theta_i^g + \sum_{i=1}^{n_h} h_i(a_{student}, a_{teacher})\theta_i^h + \sum_{i=1}^{n_m} m_i(b, a_{teacher})\theta_i^m + \sum_{i=1}^{n_l} l_i(n_e, a_{teacher})\theta_i^l$$

with the parameter vector θ decomposed as $(\theta^g, \theta^h, \theta^m, \theta^l)$.

5.3 Implementation of max-update

Recall the set of indices of active features in state-action pair (s, a) is denoted $F(s, a) = \{i \mid f_i(s, a) = 1\}$. We denote with $\theta(F(s, a))$ the tuple of the corresponding components of parameter θ. Thanks to the two assumptions (i) and (ii) of the tiling structure, the update (Equation 7) for our max-update approach can be simplified and written as follows:

$$\theta(F(s_r, a_r)) = \theta(F(s_r, a_r)) + \alpha \times [\theta(F(s_r, a_{max})) - \theta(F(s_r, a_r))] \tag{13}$$

Vector $\theta(F(s_r, a_r))$ can be seen as the output vector of a neural networks, and $\theta(F(s_r, a_{max}))$ as the desired vector. This update is a well-known gradient descent and helps make $\theta(F(s_r, a_r))$ converge to $\theta(F(s_r, a_{max}))$.

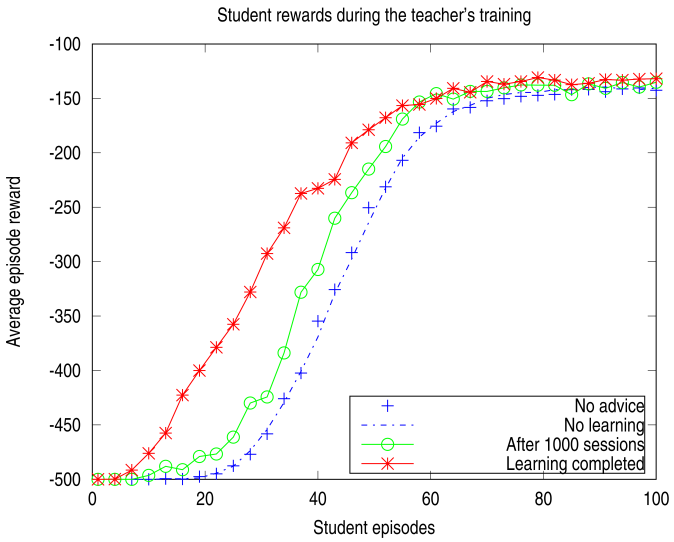

Fig. 2. Average reward of students at different levels of training of a RL teacher.

5.4 Experimental Results

In the experiments, we compare the heuristics proposed by Torrey and Taylor [18] and our approach based on representing the teacher as an RL agent. All plots are averaged over several runs, and the threshold of mistake correcting has been optimized to obtain the best performance (indeed if the threshold is too high, the heuristic will perform suboptimally as it will not spend all the available budget; conversely, if it is too low it will perform poorly as advice will be given also when not particularly relevant). We do not display the importance advising strategy because it is always outperformed by mistake correcting, since we assume that we are in a setting in which the learner agent announces her intended action before the teacher has the possibility of providing advice (importance sampling ignores this piece of information).

The RL teacher needs to interact with several learners before it learns a near-optimal teaching policy. In Figure 2 we plot the student's reward in function of the number of episodes of training *for a teacher at different levels of training* (different number of sessions with students). As expected, the student's performance improves when the teacher is more experienced. The teacher finishes to learn after approximately 8000 sessions (learning ends when Q-values converge).

In the following experiments we consider the performance of a fully trained RL teacher (meaning that the agent has already completed a number of sessions with different students, each composed of several episodes, until the teaching policy has converged). In Figures 3-6 we compare our RL teacher to the heuristics presented in Section 3 for a variety of values of the total available budget. We

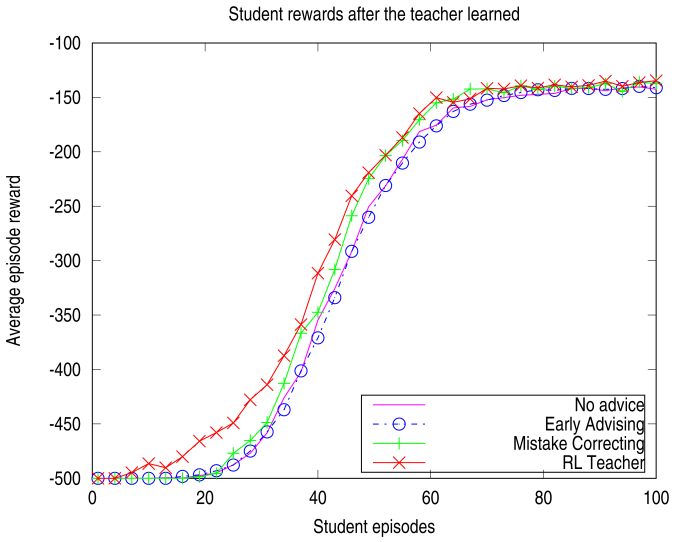

Fig. 3. Average reward of students with the fully trained RL teacher, compared to early advising and mistake correcting (budget=25; threshold for mistake correcting=76)

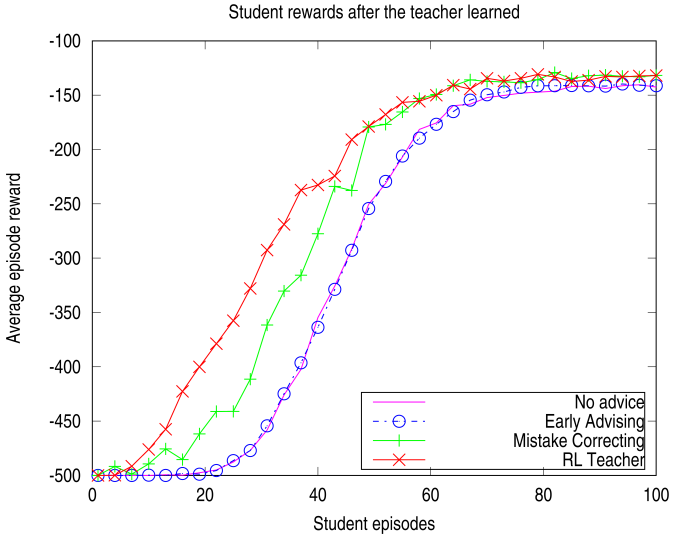

Fig. 4. Average reward of students with the fully trained RL teacher, compared to early advising and mistake correcting (budget=50; threshold for mistake correcting=148)

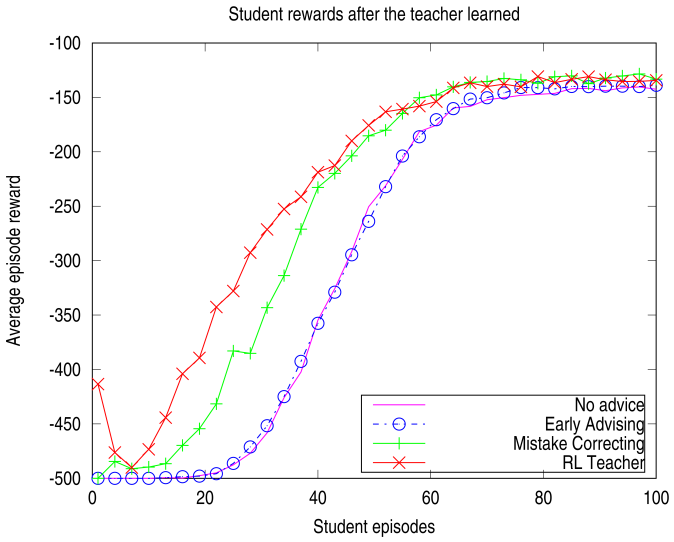

Fig. 5. Average reward of students with the fully trained RL teacher, compared to early advising and mistake correcting (budget=75; threshold for mistake correcting=128)

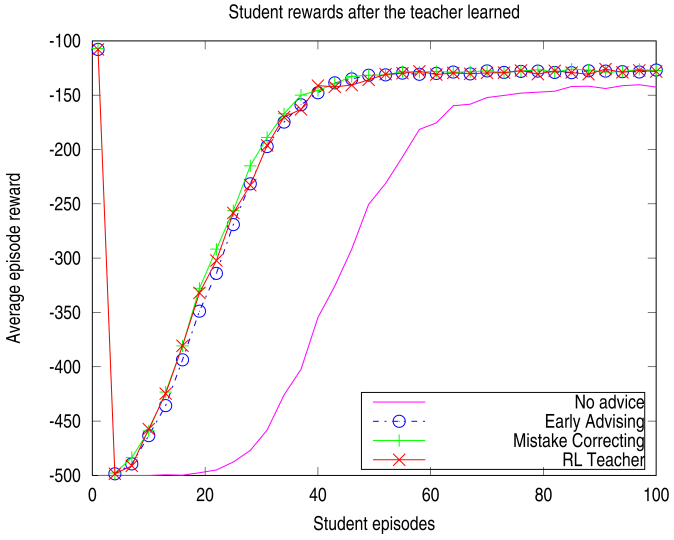

Fig. 6. Average reward of student with the fully trained RL teacher, compared to early advising and mistake correcting (budget=100; threshold for mistake correcting=4)

can notice that our RL teacher performs very well, outperforming the other strategies in all but one scenario (the high budget setting where all strategies perform roughly the same). Indeed, the bigger is the budget, the better are the performances of all teachers. The simplest strategy, early advising, performs poorly in most cases, but when the budget is high it becomes successful: the student will complete the task following the teacher's advice in each step of the first episode; given the pessimistic initialization of Q-values, the student will subsequently follow the same trajectory in successive episodes (except in some exploratory moves occurring with low probability).

As mentioned above, the threshold for mistake correcting has been tuned for best performance. It is interesting to notice that there is not a simple correlation between this threshold and the available advice budget. Indeed, the best threshold is somewhat low with a budget of 25 or 100 (Figures 3 and 6), and high for a budget of 50 or 75 (Figures 4 and 5). An explanation of this may be that if there is not enough budget, mistake correcting cannot be of too much help, on the other end, if one can give advice often, the problem becomes trivial because the student can reach the goal in the first episode just following the teacher's advice all the time.

These experimental results show that our RL teacher (when fully trained) is effective in advising students; moreover we do not have to rely on manual parameter optimization as mistake correcting does. The superior performance of our RL teacher can be explained in many ways. Intuitively, we postulate that the RL teacher is capable of autonomously discovering which states are more important, also in relation to the point of the session. Since actions are represented as features, the teacher can also learn that putting the motor in neutral position is a less severe error than moving forward when instead one should move backwards (so in the first case giving advice might not be necessary, while in the second case it will be). The availability of information about the episode in the teacher's state is crucial for learning an effective teaching policy. Indeed, for a given pair (state, action) in two different episodes of the session, we do not necessarily give advice in both.

In a final experiment, we compare the two alternative ways of using a piece of advice given by the teacher: informed exploration and max-update. In Figures 7-8 we show (for different values of the budget) how our proposed method (max-update) consistently enhances the heuristic strategies presented in Section 3. We are currently working on integrating this technique into our RL teaching agent.

6 Conclusions

In this paper we introduced a reinforcement learning approach to learning how to give advice to another RL agent (the student) in order to optimize the use of advice since only a limited number of suggestions can be given. The agent acting as a teacher observes the student's actions but cannot monitor nor change anything internal to the student.

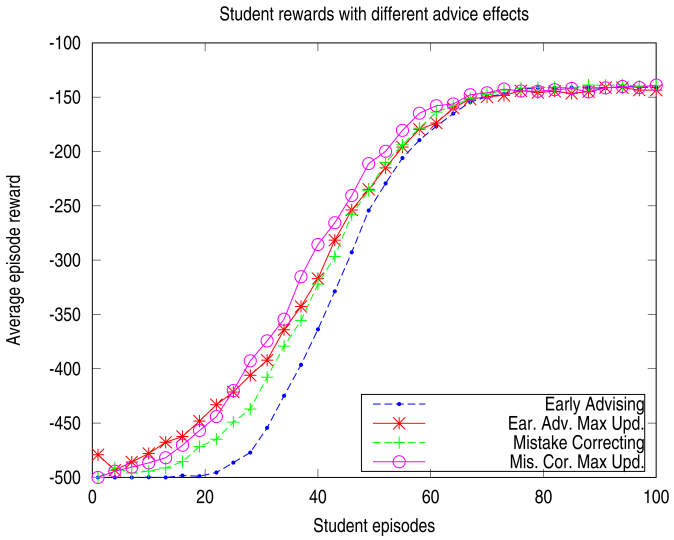

Fig. 7. Student's reward considering different heuristics with either informed-exploration or max-update (budget = 50; threshold for mistake correcting with informed exploration: 148; threshold for mistake correcting with max update: 160)

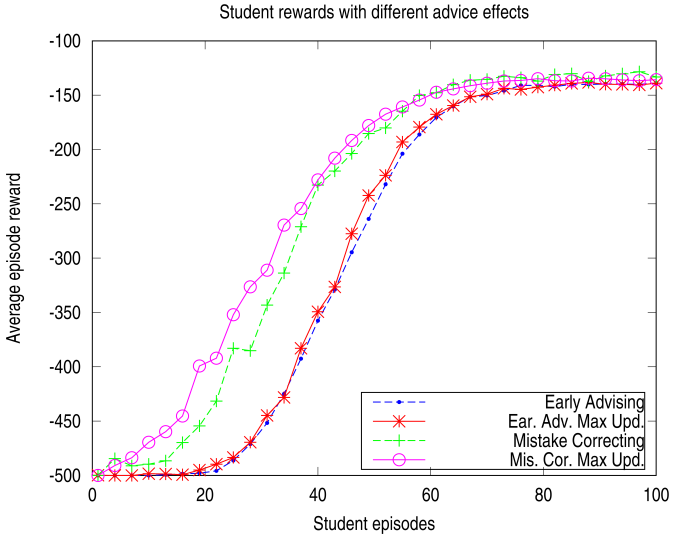

Fig. 8. Student's reward considering different heuristics with either informed-exploration or max-update (budget = 75; threshold for mistake correcting with informed exploration: 128; threshold for mistake correcting with max update: 140)

This "teaching on a budget" setting was introduced by Torrey and Taylor [18], with the specific goal of studying "the idea of how an RL agent can best teach another RL agent" but the authors did not pursue this vision all the way through, as in fact their teacher agent follows very simple *if-then-else* heuristic rules to decide whether to give advice or not.

In our work the teaching task is modeled as an RL problem whose state, in addition to the state of the student, includes information about the current teaching session (the action that the student is intended to perform, the available advice budget, the number of the episode); her possible actions represent the decision of whether giving advice or not in the current state of the student. With simulations, we showed that our techniques are competitive with respect to previously proposed heuristic techniques. Our RL teacher is effective because it can adaptively learn in which states and in which moment of the training session advice is more needed, it can learn how to spend well her budget when the budget is critically low and it is able to learn that a student agent who chooses a good enough action does not need advice.

We also considered an improved way on how the student can exploit the advice it receives from the teacher. Our max-update strategy manipulates the estimated Q-values in order to be consistent with the information that the suggested action is the best in that particular state. This come at the expense of a considerable modification of the student's architecture and of additional computation. We experimentally proved this idea to be successful in improving the mistake-correcting heuristics, but we did not yet implement it in our RL teacher.

We can extend our approach empowering our RL teacher with a classifier able to predict what the agent will do next (in the same way as the predictive advising strategy of Torrey and Taylor [18]) so that the communication between the two agents can be simplified, removing the assumption that the learner agent pre-announces her intended action to the teacher.

Other possible directions for future works include experimentation in other domains, evaluating how effective advising is when the teacher's policy is near-optimal, a setting with multiple teachers and/or multiple learners, and different types of feedbacks (including pairwise comparisons of actions as in the work of Cohn et al. [6], advice requested by the student, noisy advice).

Acknowledgments

Funded by the French National Research Agency under grant ANR-10-BLAN-0215.

References

1. Riad Akrour, Marc Schoenauer, and Michèle Sébag. April: Active preference learning-based reinforcement learning. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *ECML/PKDD (2)*, volume 7524 of *Lecture Notes in Computer Science*, pages 116–131. Springer, 2012.

2. Riad Akrour, Marc Schoenauer, and Michèle Sébag. Interactive robot education. In *ECML/PKDD Workshop Reinforcement Learning with Generalized Feedback: Beyond Numeric Rewards*, 2013.
3. Brenna Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.
4. Aude Billard and Maja J. Mataric. Learning human arm movements by imitation: : Evaluation of a biologically inspired connectionist architecture. *Robotics and Autonomous Systems*, 37(2-3):145–160, 2001.
5. J.A. Clouse. *On integrating apprentice learning and reinforcement learning.* PhD thesis, University of Massachussets, 1996.
6. Robert Cohn, Edmund H. Durfee, and Satinder P. Singh. Comparing action-query strategies in semi-autonomous agents. In Wolfram Burgard and Dan Roth, editors, *AAAI*. AAAI Press, 2011.
7. Johannes Fürnkranz, Eyke Hüllermeier, Weiwei Cheng, and Sang-Hyeun Park. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine Learning*, 89(1-2):123–156, 2012.
8. Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.
9. W.B. Knox, M.E. Taylor, and P. Stone, editors. *Understanding Human Teaching Modalities in Reinforcement Learning Environments: A Preliminary Report*, 2011.
10. J. Kober, J.A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, July 2013.
11. Adam Laud and Gerald DeJong. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 440–447. AAAI Press, 2003.
12. L.J. Lin, editor. *Programming Robots Using Reinforcement Learning and Teaching*, 1991.
13. A.Y. Ng, D. Harada, and S. Russell, editors. *Policy in variance under reward transformations: Theory and application to reward shaping*, 1999.
14. Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.
15. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, March 1998.
16. Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.
17. A.L. Thomaz and C. Breazeal, editors. *Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance*, 2006.
18. Lisa Torrey and Matthew E. Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2013.