Transfer Reinforcement Learning across Environment Dynamics with Multiple Advisors

Hélène Plisnier[1], Denis Steckelmacher[1], Diederik M. Roijers[1], and Ann Nowé[1]

Vrije Universiteit Brussel, 1050 Brussels, Belgium
helene.plisnier@vub.be

Abstract. Sample-efficiency is crucial in reinforcement learning tasks, especially when a large number of similar yet distinct tasks have to be learned. For example, consider a smart wheelchair learning to exit many differently-furnished offices on a building floor. Sequentially learning each of these tasks from scratch would be highly inefficient. A step towards a satisfying solution is the use of transfer learning: exploiting the knowledge acquired in previous (or *source*) tasks to tackle new (or *target*) tasks. Existing work mainly focuses on exploiting only one source policy as an advisor for the fresh agent, even when there are several expert source policies available. However, using only one advisor requires artificial mechanisms to limit its influence in areas where the source task and the target task differ, in order for the advisee not to be misled. In this paper, we present a novel approach to transfer learning in which all available source policies are exploited to help learn several related new tasks. Moreover, our approach is compatible with tasks that differ by their *transition functions*, which is rarely considered in the transfer reinforcement learning literature. Our in-depth empirical evaluation demonstrates that our approach significantly improves sample-efficiency.

Keywords: Reinforcement Learning · Transfer Learning · Policy Shaping.

1 Introduction

One of the main goals pursued by reinforcement learning algorithms is high sample-efficiency. Especially on physical robots or slow simulators, an algorithm that requires too many interactions with the environment before learning a good policy can be impossible to apply. In this paper, we consider a setting where the agent needs to learn a large amount of similar yet distinct tasks. A simple example of this setting is a motorized wheelchair that has to learn how to go to any of the 22 offices (and a toilet) on a floor. A much more challenging example, on which we focus on this paper, is the wheelchair learning how to exit any of these offices from any location inside them. Offices have different shapes and furnishing, which makes them share a state representation, but *distinct transition*

dynamics. In this paper, we propose a method to reuse the knowledge acquired from learning several past tasks to tackle a new task, having the same state-space and action space but different dynamics and reward functions.

Transfer learning (Taylor and Stone, 2009) has the potential to make RL agents much faster at mastering new tasks (also called *target* tasks), by allowing the reuse of knowledge acquired in previous tasks (also called *source* tasks). An important challenge of transfer learning for an RL agent launched in a target task, is to determine which parts are shared by both the target task and the source task, and which parts differ. This information is critical; indeed one cannot just blindly follow the policy learned in the source task while tackling the target task, because a behavior that worked well in the past might not apply anymore. Therefore, it is necessary for a transfer learning algorithm to be capable of following the past policy only when suitable, and to otherwise diverge from it. To mitigate this problem, current techniques either artificially set a probability of following the advisor, hence limiting the potential damage of irrelevant advice (Fernández and Veloso, 2006); or let the agent learn when to follow advice (Taylor and Stone, 2007; Taylor et al., 2007; Parisotto et al., 2015). If several expert policies are available, then isolating one of them and only exploiting that one policy often becomes the focus, instead of exploiting a combination of them (Taylor and Stone, 2007; Fernández and Veloso, 2006). Hence, existing methods tend to add an extra learning problem, by either learning when to follow the advisor, or which advisor to follow.

In this paper, we consider the case where a large amount of similar, yet distinct tasks must be learned. Each of these tasks differs in its environment dynamics, and achieving success highly depends on learning these environment dynamics dissimilarities. We have at our disposal a number of expert source policies learned in the same environment, which we call *advisors*. Our main contribution, inspired by the Actor-Advisor (Plisnier et al., 2019), is a transfer learning approach combining all of the available expert source policies into one advisor. This advisor is confident in areas where the source tasks are similar, meaning that the advice probably applies in the target task as well, and is uncertain where the source tasks differ. The latter case allows the fresh agent to learn for itself what is best to do in situations unknown to its advisor. We implement our transfer learning method within Bootstrapped Dual Policy Iteration (BDPI) (Steckelmacher et al., 2019), an already extremely sample-efficient model-free actor-critic reinforcement learning algorithm, and we empirically show that, combined with our contribution, BDPI can achieve an even higher sample-efficiency.

2 Related Work in Transfer Learning

Reusing previously learned policies or skills to learn a new task is an intuitive approach to improving sample-efficiency (Taylor and Stone, 2009). Here, we consider that the knowledge to be transferred is represented by the policy π_{source} learned by a reinforcement learner in the source task (Brys, 2016, p. 34). Transferring only isolated parts of behavior (Andre and Russell, 2002; Ravindran and

Barto, 2003; Konidaris and Barto, 2007), such as skills or *options* (Sutton et al., 1999) is left out of the scope of this related work section.

Our method falls in the realm of transfer learning via exploration alteration (García and Fernández, 2015). A straightforward way of transferring a policy π_{source} is to leverage it in the agent's exploration strategy, letting it bias or determine the agent's actions at action selection time (Fernández and Veloso, 2006; Taylor and Stone, 2007; Plisnier et al., 2019). Such artificially altered exploration generally requires to be used with a value-based method, as the agent must be able to learn from *off-policy* experiences (except for Plisnier et al. (2019), which learns with Policy Gradient). In Fernández and Veloso (2006), 5 different π_{source}s, each learned in the same office-like gridworld but with a different goal location, are reused to learn a task with a new goal location. As the 5 π_{source}s are not all good advisors, the authors propose a method that allows the agent to learn which π_{source} is more relevant to the target task. However, they only consider dissimilarities in the reward function (by changing the goal location), and not in the environment dynamics. Taylor and Stone (2007) transfers rules learned in a discrete state-space environment to a continuous state-space environment. Similarly to Fernández and Veloso (2006), even though they learn several π_{source} in different source tasks, they end up only reusing one π_{source} at a time (the one that is the most relevant to the target task), instead of a combination of all π_{source}s. The Actor-Advisor (Plisnier et al., 2019) mixes π_{source} with the Policy Gradient actor's policy at action selection time, using the policy mixing formula in Griffith et al. (2013) (in Section 3.3). This way, the learning process is influenced by π_{source}, while π_{source} also guides the agent's exploration. In the transfer task in Plisnier et al. (2019), the doors of a maze are shifted, resulting in a change in the dynamics of the environment. We use a larger version of this environment in our experiments, and use a combination of multiple π_{source} as advisor.

Another approach to transfer is to either initialize the agent's policy with π_{source} (Taylor et al., 2007; Parisotto et al., 2015), and/or to make the agent actively learn to imitate π_{source} (Brys et al., 2015; Mohammedalamen et al., 2019). Multitasking is an RL subdomain close to transfer learning, which aims at generating one policy able to perform multiple distinct tasks. Several techniques exist to achieve this goal. The Actor-Mimic (Parisotto et al., 2015) uses several DQN policies (each expert in a different source task) to train a *single* multi-task student network, by minimizing the cross-entropy loss between the student and experts' policies. Transfer can then be done by initializing yet another DQN network with the resulting multi-task expert policy. Although our contribution reuses multiple π_{source}, it is distinct from multitasking in that it is not trying to generate one good policy able to perform several tasks, but to improve the sample-efficiency of a fresh agent learning one new task.

3 Background

In this section, we introduce the concepts at the basis of our work: Markov Decision Processes, Bootstrapped Dual Policy Iteration (BDPI), Policy Shaping and the Actor-Advisor.

3.1 Markov Decision Processes

A discrete-time Markov Decision Process (MDP) Bellman (1957) with discrete actions is defined by the tuple $\langle S, A, R, T \rangle$: a possibly-infinite set S of states; a finite set A of actions; a reward function $R(s_t, a_t, s_{t+1}) \in \mathbb{R}$ returning a scalar reward r_t for each state transition; and a transition function $T(s_{t+1}|s_t, a_t) \in [0, 1]$ taking as input a state-action pair (s_t, a_t) and returning a probability distribution over new states s_{t+1}.

A stochastic stationary policy $\pi(a_t|s_t) \in [0, 1]$ maps each state to a probability distribution over actions. At each time-step, the agent observes s_t, selects $a_t \sim \pi(s_t)$, then observes r_{t+1} and s_{t+1}. The $(s_t, a_t, r_{t+1}, s_{t+1})$ tuple is called an *experience* tuple. An optimal policy π^* maximizes the expected cumulative discounted reward $E_{\pi^*}[\sum_t \gamma^t r_t]$, where γ is a discount factor. The goal of the agent is to find π^* based on its experiences within the environment.

3.2 Bootstrapped Dual Policy Iteration

Bootstrapped Dual Policy Iteration (Steckelmacher et al., 2019, BDPI) is an actor-critic method, with one actor and $N_c > 1$ critics. The critics are trained using Aggressive Bootstrapped Clipped DQN (Steckelmacher et al., 2019), a version of Clipped DQN Fujimoto et al. (2018) that performs $N_t > 1$ training iterations per training epoch. Each critic maintains two Q-functions, Q^A and Q^B. Each training epoch, a batch b_i is sampled for each critic $i \in [1, N_c]$ from an experience buffer B. Then, for each training iteration, every critic i swaps its Q^A and Q^B functions, then Q^A is trained using Equation 1 on b_i.

$$\begin{aligned} Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) \; + \; \alpha \left(r_{t+1} + \gamma \, V(s_{t+1}) \right) \\ - \, Q_k(s_t, a_t) \\ V(s_{t+1})) = \min_{l=A,B} Q^l(s_{t+1}, \operatorname*{argmax}_{a'} Q^A(s_{t+1}, a')) \end{aligned} \tag{1}$$

The actor π is trained using a variant of Conservative Policy Iteration (Pirotta et al., 2013). Every training epoch, after the critics have been updated for a number N_t of times, the actor is trained towards the greedy policy of all its critics. This is achieved by sequentially applying Equation 2 N_c times, each iteration updating the actor based on a different critic.

$$\pi(s) \leftarrow (1 - \lambda) \, \pi(s) + \lambda \, \Gamma(Q^{A,i}_{k+1}(s, \cdot)) \qquad \forall s \in b_i \tag{2}$$

where $\lambda = 0.05$. A great asset of BDPI over other state-of-the-art actor-critic methods is its high sample-efficiency, due to the aggressiveness of its off-policy critics.

3.3 Policy Shaping and the Actor-Advisor

Policy Shaping (Kartoun et al., 2010; Griffith et al., 2013; MacGlashan et al., 2017; Harrison et al., 2018) generally aims at letting an external advisory policy π_{source} (we call it π_{source} since, in our case, it is learned in the *source* task) alter or determine the agent's behavior. The specific Policy Shaping formula we are considering in this paper is the one suggested by Griffith et al. (2013):

$$a_t \sim \underbrace{\frac{\pi(s_t)\pi_{\text{source}}(s_t)}{\pi(s_t) \cdot \pi_{\text{source}}(s_t)}}_{\sum_{a \in A} \pi(a|s_t)\pi_{\text{source}}(a|s_t)} \tag{3}$$

where $\pi(s_t)$ is the state-dependent policy learned by the agent, $\pi_{\text{source}}(s_t)$ is the state-dependent advice, and $\pi(s_t) \cdot \pi_{\text{source}}(s_t)$ is the dot product. The actions executed by the agent in the environment are sampled from a mixture of the agent's current learned policy π and an external advisory policy π_{source}. Executing actions from this mixture allows the advisor π_{source} to guide the agent's exploration and potentially improves its performance. This method not only allows the actor to benefit from the advisor's expertise; it also lets the actor eventually outperform its advisor. This way, the actor's performance is never bounded by its advisor's, and the advisor does not need to have a complete knowledge of the task to be solved.

The Actor-Advisor (Plisnier et al., 2019) is the first attempt made at using the policy mixing formula in Equation 3 to achieve transfer learning. Their main contribution is to directly influence a Policy Gradient agent's policy with some off-policy external advice π_{source}, without convergence issue. In the next section, we detail our contribution, which is in part an extension of BDPI inspired by the Actor-Advisor.

4 BDPI with Multiple Advisors

In this section, we introduce our contributions. The first one is an extension of the Bootstrapped Dual Policy Iteration (Steckelmacher et al., 2019) that allows advice to be given to the agent. The Actor-Advisor (Plisnier et al., 2019) having been originally designed for Policy Gradient algorithms, its application to an actor-critic algorithm that does not use Policy Gradient is far from trivial. Our second contribution is a training method that consists of training an agent on N tasks, then using these N actors to advise the $N + 1^{th}$ agent, then using the $N + 1$ agents to advise the $N + 2^{th}$, and so on.

4.1 Bootstrapped Dual Policy Iteration with Advice

In their original work, Plisnier et al. (2019) propose a Policy Gradient actor $\pi(a|s, \pi^E)$ that takes external advice π^E as input, in addition to a state. The actor, implemented as a neural network, computes a learned candidate policy π^L that is combined with π^E at the very end of the network, using the policy mixing formula of Griffith et al. (2013). This integration of the advice within the neural network allows the behavior of the agent to be influenced by external advice without convergence issues, despite Policy Gradient being a strong on-policy algorithm. Moreover, because support for advice is built directly into the neural network, the presence of advice influence both *acting* and *learning*. When extending BDPI with advice in a similar fashion as the Actor-Advisor, we must therefore define how its *actor* learns and acts, and how its *critics* learn.

Acting When acting, we consider that the actor of BDPI is $\pi^L(a|s)$, the learned actor. This actor does not observe any advice, and only produces the usual state-conditioned probability distribution over actions. We propose to introduce advice to the agent by simply using the formula introduced by Griffith et al. (2013):

$$\pi(a|s, \pi^E) = \frac{\pi^L(a|s)\pi^E(a|s)}{\sum_{a'} \pi^L(a'|s)\pi^E(a'|s)} \tag{4}$$

with $\pi^E(a|s)$ a normalized probability vector that encodes the advice given to the agent in state s. The agent directly observes $\pi^E(a|s)$ values, and has no direct knowledge of π^E. The advice vector received by the agent at each time-step is part of the experience, which leads to the agent storing $(s_t, a_t, r_t, s_{t+1}, \pi^E(\cdot|s_t))$ tuples in its experience buffer.

Training the actor As demonstrated in Steckelmacher et al. (2019), BDPI is an off-policy algorithm. The demonstration consists of forcing 20% of the actions being executed in the environment to be random, and showing that this does not impair learning speed. With advice, the situation is slightly different. Since BDPI is off-policy, in contrast to the Actor-Advisor, we theoretically do not need to incorporate the advice in the actor's learning rule. Indeed, a BDPI agent is perfectly able to learn optimal policies by using advice solely at acting time. However, it is believed by the authors of Plisnier et al. (2019) that, in the Actor-Advisor, the integration of the advice in the Policy Gradient's loss not only allows for Policy Gradient to learn without diverging, but also leads to an implicit learning correction, which contributes to the agent's performance. As BDPI does not have a loss similar to the Policy Gradient's one, we must explicitly define a similar learning correction in the actor's learning rule. Our correction consists of observing that the BDPI critics produce greedy policies $\Gamma(Q)$ that converge to the optimal policy for the task. This means that the *policy as executed by the agent*, that mixes the actor and the advisor, must pursue $\Gamma(Q)$:

$$\pi(s, \pi^E(s)) \leftarrow \Gamma(Q(s)) \qquad \text{converges to optimal policy} \tag{5}$$

Starting from Equation 5, we isolate π^L, the actor of BDPI:

$$\begin{aligned}
\pi(s, \pi^E(s)) &\leftarrow \Gamma(Q(s)) \\
\frac{\pi^L(s)\pi^E(s)}{|\pi^L(s) \cdot \pi^E(s)|} &\leftarrow \Gamma(Q(s)) \\
\pi^L(s)\pi^E(s) &\leftarrow \underbrace{\Gamma(Q(s)) \times \overbrace{|\pi^L(s) \cdot \pi^E(s)|}^{\text{a scalar}}}_{\text{a vector}} \\
\pi^L(s) &\leftarrow \frac{\Gamma(Q(s)) \times |\pi^L(s) \cdot \pi^E(s)|}{\pi^E(s) + \varepsilon}
\end{aligned} \tag{6}$$

with the fraction an element-wise division between two vectors, and ε a small positive value that prevents a division by zero if the advice contains any zero.

Intuitively, Equation 6, the actor learning rule that we use instead of the standard BDPI one, moves the actor in the direction of the greedy function of a critic, as described by Steckelmacher et al. (2019), with the addition of a *weight* that influences how much the greedy function is followed. The learning correction works as follows: the more the advice differs from the actor, the more the actor will follow the greedy function (and thus pull away from the advice). Such a pull allows the agent to compensate for bad advice, by learning an actor π^L that, when combined with the faulty advice, still leads to a good policy. The observation that the actor tries to differ from the advisor, that we explain in this paper, has also been made by Plisnier et al. (2019), when they discuss that the policy gradient forces the policy to move away from the advisor. In Plisnier et al. (2019), no explanation or justification for this pull-away behavior was given.

Training the critics The critics of BDPI being updated with a variant of (off-policy) Q-Learning, the fact that the behavior of the agent is altered by advice does not change any of the update rules. The greedy policies of the critics will automatically converge to the optimal policy for the task.

4.2 Receiving Advice from Many Advisors

An important challenge in transfer reinforcement learning is discovering when to follow the advisor (trained on the source task), and when to ignore it. Ideally, the advisor should be followed only in the states for which its policy is optimal in the target task. Unfortunately, knowing whether the advisory policy is optimal in a state is impossible until the agent has fully learned the target task.

Previous work learns in which states to use the advisor, or which advisor to use in which state (Fernández and Veloso, 2006; Taylor and Stone, 2007), or

relies on the uncertainty or entropy of the advisor to decide how much to follow it (Plisnier et al., 2019). The observation is that when the advisor is certain of what has to be done, it is probably right. We provide a counter-example for this statement: in our environment (see Section 5), passing through a (thin) door requires careful moves, leading to a policy of very low entropy in states near doors. However, if the source task and target task have doors at different locations, the advisor will very confidently make the agent bang against a wall, where it thinks a door is.

We argue that it is initially impossible to evaluate when to follow advice, and when to ignore it, based solely on the advice received from a single advisor. Building on our hypothesis that evaluating a single advisor in a sample-efficient way is impossible, we instead propose to use *several* advisors, and combine them in the following way:

$$\pi^E(s) = \frac{1}{N} \sum_i \left(\pi_i^E(s) + 1 - \rho \right) \tag{7}$$

with N the number of advisors, and $\rho \leq 1$ the weight of the advisor's influence, that does nothing when set to 1, and artificially increases the entropy of the advisors when set to a value smaller than 1. Equation 7 is a simple average of the advice given by all the advisors. Intuitively, if N is large enough (typically 5 to 10), the advisors will tend to agree in states in which the tasks being learned share a common structure, and disagree in states where the tasks differ. Even if every single advisor is highly confident in these differing states, the average will produce a probability distribution of high entropy. Moreover, the advisors may agree that a few actions are bad in a given state, while not agreeing on which ones are good. In our example of leaving a room, moving towards the window is always bad, regardless of the room, while moving towards the door will be more strongly advised.

Building on Equation 7, we propose the following training method for multiple tasks that share a common structure but different dynamics: train N actors on N tasks from scratch, with N between 5 and 10 depending on the complexity of the tasks. Then, combine the N actors in a single advisor, using Equation 7, and use it to significantly improve the sample-efficiency and safety of the $N+1^{th}$ actor. Then, add the $N+1^{th}$ actor to the pool of actors used to produce advice, and repeat for $N+2$, ... Our experimental results, that we now present, validate our approach and show that, as more advisors are available, the sample-efficiency of the agent increases on new tasks.

5 Experiments

We now evaluate our Advised BDPI algorithm in a representative environment for which 676 tasks have to be learned. We demonstrate that learning a small amount of tasks from scratch (even as low as 4) allows the next tasks to be solved significantly faster. Moreover, the more tasks have been learned, the more

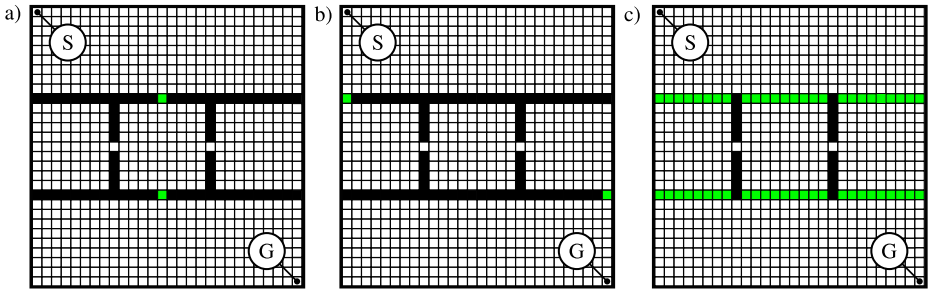


Fig. 1: The 29×28 Five Rooms environment used in our experiments. 'S' denotes the starting cell, 'G' the goal cell. a) The original door configuration. The first and last doors (in green) are the ones that are allowed to move; b) an example of an alternative configuration of the doors; c) all the potential door locations. For each of the two doors, there are 26 such locations, which results in $26 \times 26 = 676$ different environment configurations.

sample-efficient the agent becomes on new tasks. Combined with the already high sample-efficiency of BDPI, this demonstrates that reinforcement-learning can now be used to train full multi-task systems.

5.1 Environment

We evaluate our method on the Five Rooms environment (see Figure 1 introduced in Plisnier et al. (2019)). Five Rooms is a 29 cells high and 28 cells wide grid world, divided by walls into five rooms. Each of the rooms is accessible via one of the four one-cell-wide doors. The agent can move one cell up, down, left or right, unless the target cell is a wall (then the agent does not move). The agent must get to the goal cell, always located at the bottom right corner of the bottom room, starting from the top-left corner of the top room. Hence, in order to reach the goal, the agent imperatively has to go through at least two doors (the ones in green on Figure 1, a.). Reaching the goal results in a $+100$ reward, the agent otherwise receives -1 per time-step. The episode terminates either once the goal is reached, or after 500 unfruitful time-steps.

In our experiments, we vary the location of the doors on the two horizontal walls (see Figure 1). Each door can be located in any of 26 cells. Every combination allows the goal to be reached from the initial position, but some combinations lead to shorter or longer optimal paths (see, in Figure 2, the difference between the paths in examples a and g). Because we do not move the goal, and do not alter the reward function, but only vary the location of the doors across task, our environment isolates the impact of a varying transition function. Moving doors inside the environment is also a simple but relevant illustration of our example task: *learn to get out of many rooms, each room differing by its furniture*. Finally, moving a door is a localized change to the environment, but is still very challenging: directly applying a policy learned in a source configuration in a target configuration would lead the agent to get stuck against a wall

in the target environment (where there is a door in the source environment). The source policy, strongly used to see a door where it is not there anymore, would be highly confident in its bad action and mislead the fresh agent. We now describe our experimental setup, and show that our Advised BDPI successfully tackles this challenge.

5.2 Experimental Settings

We evaluate our Advised BDPI on the tasks described above. BDPI has been configured closely to what is recommended in Steckelmacher et al. (2019): 8 critics, all trained every time-step for 4 Clipped DQN iterations. The current state (the current cell in which the agent is) is one-hot encoded into a vector of 812 floats. The BDPI actor and critics are neural networks with a single hidden layer of 256 neurons, with the tanh activation function, and one output per action. Actor and critic networks are trained with the Adam optimizer, with a learning rate of 0.0001, for 10 gradient steps per Clipped DQN iteration. In order to produce our results, we trained many agent in this order:

1. 100 agents have been trained from scratch, each on a different random door configuration (from 676). This produces a pool of advisors.
2. The curves of Figure 3 have been produced by training around 20 agents (per curve), each on a randomly-selected configuration of doors, and using 50 of the advisors produced at step 1 for advice.
3. The curves of Figure 5 have been produced the same way as in step 2, but by using either more (100) or less (4 or 1) advisors for advice. Our experiments with $N = 100, 50, 4$ or 1 advisors measure the sample-efficiency gains that are obtained when training the $N + 1^{\text{th}}$ agent using N advisors.
4. The curves of Figure 9 have been produced the same way as in step 2 and 3, but means to illustrate the effect of implementing the learning correction (see Section 4). They are both advised by 100 advisors, and averaged over around 10 runs each.
5. For steps 2, 3 and 4, we vary the ρ parameter of Equation 7, to evaluate the impact of artificially increasing the entropy of the advisors. We generally show that artificially increasing the entropy of the advisors is not needed to obtain good results, which demonstrates the benefits of averaging advisors, and removes one tunable parameter from our algorithm.

Because each configuration of doors has an optimal policy that achieves a different return (as illustrated in Figure 2), we evaluate each agent on a large amount of runs, to average out the effect of the door positions. This allows us to produce curves with high confidence.

5.3 Results

Figure 3 shows that using advice generally improves performance, especially at the beginning of learning. In addition, it can be noticed that a high ρ (i.e., $\rho = 1$)

helps in the beginning of learning, but tends to prevent the agent from achieving the best performance towards the end. A lower ρ (i.e., $\rho = 0.7$), on the other hand, provides more freedom to the agent to reach that high performance at the end, but at the cost of lower performance in early learning stages.

We also evaluate the impact of either having a large amount of advisors (100, in our case) or only a few (4 or 1), while varying the advisors' weight ρ (0.8 or 1, see Figure 5). When $\rho = 0.8$ (see Figure 4a), i.e., the influence of the advisors is somewhat moderate, having a few or a large amount does not seem to matter in the long run. However, when the influence of the advisors is maximum (i.e., $\rho = 1.0$, in Figure 4b), then increasing the amount of advisors increases sample-efficiency. Averaging over a large number of advisors naturally exhibits uncertainty in areas where the target task and the source tasks might differ, allowing the agent to be less dependent on the ρ parameter. Even having only 4 advisors instead of 1 dramatically improves performance.

We compared leveraging several advisors to leveraging only one carefully selected advisor. This advisor is selected based on its doors configuration; the location of the doors in its source task is the closest to the location of the doors in the target task. Hence, as suggested by Fernández and Veloso (2006), out of all advisors from the pool, this advisor should be the most suited to provide quality advice to the fresh agent. However, even though there is a significant improvement between Figure 5 (i.e., a randomly selected advisor) and Figure 7 (i.e., the best advisor), averaging over 4 randomly chosen advisors can still provide better results in the long run, in both cases where $\rho = 0.8$ and $\rho = 1.0$. Moreover, having only one advisor, albeit the best, still makes the fresh agent rely on a low value of ρ.

Finally, we assess the influence of the learning correction (see Section 4) on learning while being advised (see Figure 9). When we do not artificially increase the entropy of the advisors (i.e., when $\rho = 1$), learning the task while being advised is harder without the learning correction. The importance of the learning correction is lesser when $\rho < 1$, though. In contrast, the performance of the agents with the learning correction is similar in both settings of ρ (0.8 or 1), which demonstrates that our learning correction positively impacts the robustness of the agent.

6 Conclusion

In this paper, we present a transfer learning method exploiting multiple advisors to tackle new tasks. The source tasks and the target tasks take place in the same state-space, but present crucial differences in their environment dynamics. Our motivation example is a smart wheelchair having to exit several different offices. Even though it is likely that all offices of a building floor share a common layout, these offices might also be furnished differently, which makes navigation a unique experience in each of them for a reinforcement learner. We contribute a transfer learning method consisting in averaging the advice coming from multiple advisors, and providing this averaged advice to the fresh agent tackling a

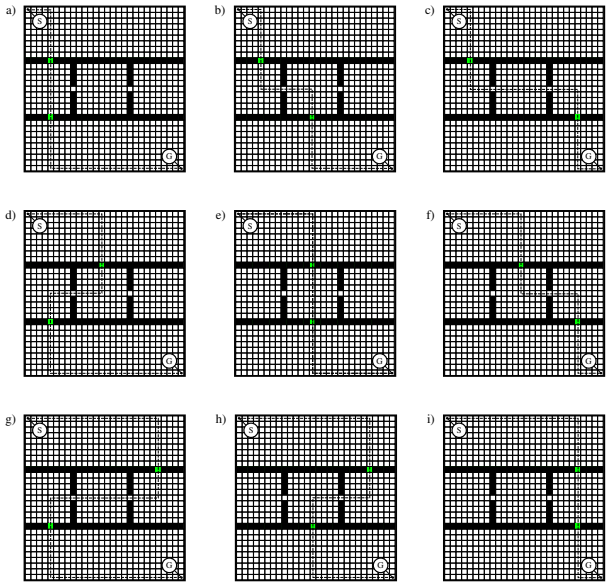


Fig. 2: These 9 examples of configurations show how the difficulty of the task (i.e., reaching the goal from the initial cell) can strongly vary from configuration to configuration. The path from the initial cell to the goal cell is much longer and convoluted in configuration g) than in configuration a), for instance.

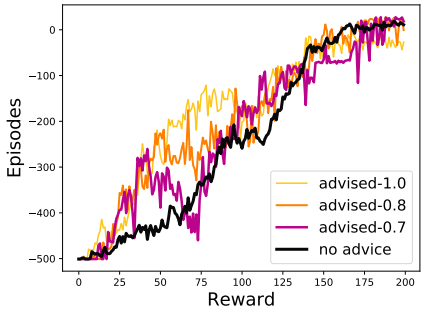
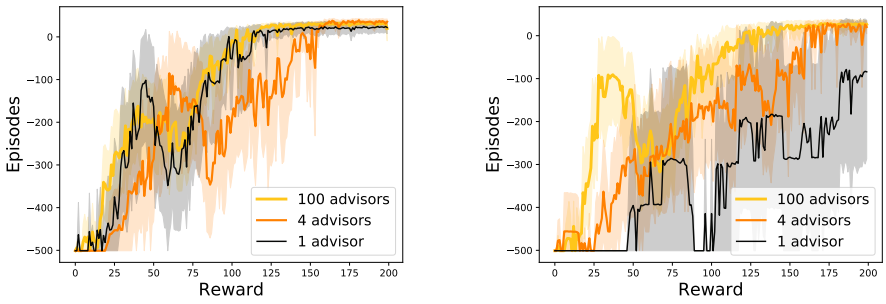


Fig. 3: Comparison between learning the task while using advice from 50 advisors (with $\rho = 1.0$, 0.8 or 0.7) and learning the task without advice. These curves are averaged over multiple runs: around 25 runs for the "advised" curves, and 100 runs for the "no advice" curve. A high ρ greatly helps the agent at the beginning of learning but slightly decreases performance in the long run, while a lower ρ allows the agent to reach a better policy at the end of learning, but provides a weaker jumpstart.



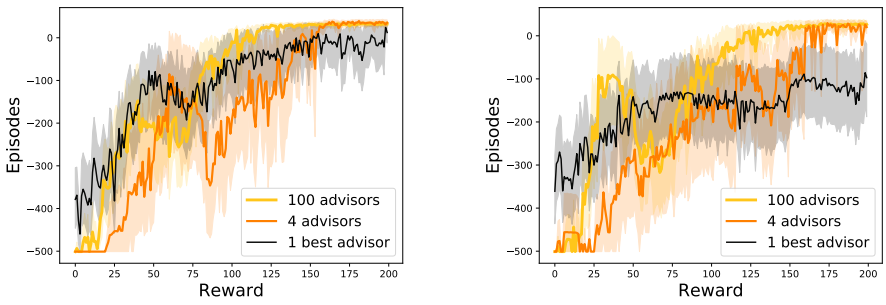
$\rho = 0.8$ $\rho = 1.0$

Fig. 5: Comparison between learning the task while advised by either 100 advisors, 4 advisors or only 1 advisor. When having a large amount of advisors to exploit, the performance of our method seems to remain stable, whether $\rho = 1.0$ or $\rho = 0.8$. Being advised by only one advisor, on the other hand, makes the agent rely more on a low ρ (i.e., artificially increasing the entropy of the advisors) to not get stuck with a suboptimal policy.

target task. This approach naturally balances advice versus *tabula-rasa* learning, depending on where the tasks are similar or not. We perform a thorough empirical evaluation of our method by: i) assessing the increase in performance gained thanks to the use of advice compared to none; ii) evaluating the benefit of having multiple advisors to average over instead of only one; iii) assessing the importance of implementing our learning correction to ensure stable learning. We saw that our contribution allows BDPI, an already highly sample-efficient algorithm, to be even more sample-efficient in a multi-task setting. This opens multi-task reinforcement-learning to areas, such as robotics, where many tasks have to be learned quickly.

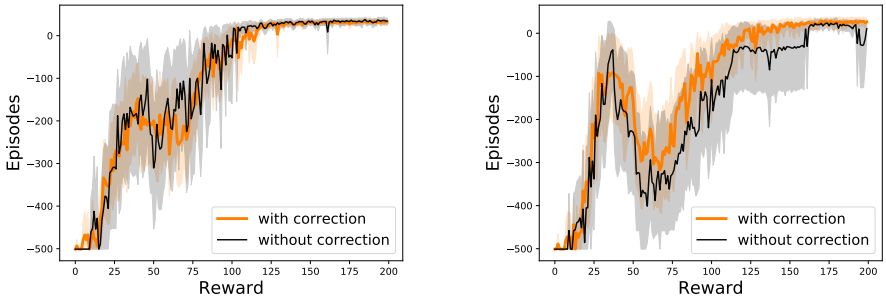
Acknowledgments

The first and second authors are funded by the Science Foundation of Flanders (FWO, Belgium), respectively as 1SA6619N Applied Researcher, and 1129319N Aspirant.



$\rho = 0.8$ $\rho = 1.0$

Fig. 7: Comparison between learning the task while advised by either 100 advisors, 4 advisors or the best advisor for the target task. The "best advisor" is the advisor which source task has the most similar doors configuration to that of the target task. Even though one carefully chosen advisor gives better advice than a randomly chosen one, the performance it can achieve in the long run is still below that of 4 randomly chosen advisors, whether $\rho = 0.8$ or $\rho = 1.0$. Additionally, being advised by only advisor, albeit the most suited one for the target task, still leads to an agent highly dependent on ρ.



$\rho = 0.8$ $\rho = 1.0$

Fig. 9: In both plots, the two curves are averages over around 20 runs of agents advised by 100 advisors. We compare between using advice with the learning correction and without the learning correction. The learning correction ensures a more stable performance regardless of whether $\rho = 0.8$ or 1.0 than when it is not implemented.

Bibliography

Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, pages 119–125.

Bellman, R. (1957). A Markovian decision process. *Journal Of Mathematics And Mechanics*.

Brys, T. (2016). *Reinforcement Learning with Heuristic Information*. PhD thesis, PhD thesis, Vrije Universitet Brussel.

Brys, T., Harutyunyan, A., Taylor, M. E., and Nowé, A. (2015). Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 181–188. International Foundation for Autonomous Agents and Multiagent Systems.

Fernández, F. and Veloso, M. M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *International Conference on Autonomous Agents and Multiagent Systems*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*.

García, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*.

Griffith, S., Subramanian, K., Scholz, J., Isbell, C. L., and Thomaz, A. L. (2013). Policy shaping: Integrating human feedback with reinforcement learning. In *Neural Information Processing Systems*.

Harrison, B., Ehsan, U., and Riedl, M. O. (2018). Guiding reinforcement learning exploration using natural language. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1956–1958. International Foundation for Autonomous Agents and Multiagent Systems.

Kartoun, U., Stern, H., and Edan, Y. (2010). A human-robot collaborative reinforcement learning algorithm. *Journal of Intelligent & Robotic Systems*, 60(2):217–239.

Konidaris, G. and Barto, A. G. (2007). Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900.

MacGlashan, J., Ho, M. K., Loftin, R., Peng, B., Wang, G., Roberts, D. L., Taylor, M. E., and Littman, M. L. (2017). Interactive learning from policy-dependent human feedback. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2285–2294. JMLR. org.

Mohammedalamen, M., Khamies, W. D., and Rosman, B. (2019). Transfer Learning for Prosthetics Using Imitation Learning. *arXiv e-prints*, page arXiv:1901.04772.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

Pirotta, M., Restelli, M., Pecorino, A., and Calandriello, D. (2013). Safe policy iteration. In *International Conference on Machine Learning*, pages 307–315.

Plisnier, H., Steckelmacher, D., Roijers, D. M., and Nowé, A. (2019). The Actor-Advisor: Policy Gradient With Off-Policy Advice. *arXiv e-prints*, page arXiv:1902.02556.

Ravindran, B. and Barto, A. G. (2003). Relativized options: Choosing the right transformation. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 608–615.

Steckelmacher, D., Plisnier, H., Roijers, D. M., and Nowé, A. (2019). Sample-Efficient Model-Free Reinforcement Learning with Off-Policy Critics. *arXiv e-prints*, page arXiv:1903.04193.

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

Taylor, M. E. and Stone, P. (2007). Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886. ACM.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*.

Taylor, M. E., Stone, P., and Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(Sep):2125–2167.