Self-Transfer Reinforcement Learning for Continuous Control Tasks

Hélène Plisnier
Vrije Universiteit Brussel
Brussels
helene.plisnier@vub.be

Denis Steckelmacher
Vrije Universiteit Brussel
Brussels
denis.steckelmacher@vub.be

Ann Nowé
Vrije Universiteit Brussel
Brussels
ann.nowe@vub.be

ABSTRACT

Policy Intersection is a Policy Shaping technique that allows an external policy from any source to influence (*advise*) the action selection of a Reinforcement Learning agent. This method can be used to transfer policies between similar yet distinct tasks, by having the policy learned on the first task advise the agent that now learns the second task. In this paper, we introduce a Self-Transfer setting using Policy Intersection: we let an RL agent learn a task for a short period of time, then freeze it and use it as an advisor for a fresh agent, that learns the same task from scratch. Although transferring a policy from a task to itself might seem redundant, we present preliminary results empirically showing that this approach brings a non-negligible gain in sample-efficiency on three challenging continuous control environments. We suspect that Self-Transfer positively impacts exploration, both in policy space (exploring more states) and in parameter space (the fresh agent's policy is initialized used a fresh random seed).

KEYWORDS

Reinforcement Learning, Sample-Efficiency, Transfer Learning

1 INTRODUCTION

Policy Intersection (PI) [14] [1] is a Policy Shaping method that guides the exploration of the agent [12]. Policy Shaping methods allow an external policy to alter or determine the policy of a Reinforcement Learning agent. In PI, the external policy advising the actor can be from any source; this allowed its application to a variety of RL problems, amongst which Transfer Learning [26, 32]. A Transfer Learning (TL) setting often involves a *source* task and a *target* task; first, the agent learns the source task, then the knowledge acquired in the source task is intelligently leveraged by the agent while tackling the target task, using a Transfer Learning method [32, 36]. The aim of TL is generally to make an RL agent learn new tasks faster by allowing it to reuse previously learned knowledge efficiently.

In this paper, we train an RL agent for a small amount of episodes on a task and freeze the resulting policy. Then, we reuse this frozen policy as an advisor to assist a fresh agent learning until reaching a good policy on the same task, using a novel Policy Intersection algorithm designed for continuous actions. In our setting, the source and target tasks are the same, but the training time allocated to learning the frozen policy is a fraction of the time allocated to the fresh policy. We observe that an RL agent performing this "self-transfer" significantly improves its total sample-efficiency, that includes training both the frozen and fresh policies, compared to what it would have achieved by learning the task with no Self-Transfer. We also observe that the final performance of the agent with Self-Transfer is higher than the final performance of an agent without Self-Transfer.

[1]This specific algorithm was originally named Policy Shaping by [14], but we rename it Policy Intersection in this paper, as we believe that "Policy Shaping" can serve more accurately as an umbrella term for multiple algorithms (such as Policy Intersection).

In addition to showing the benefits to be obtained from our implementation of Self-Transfer, our empirical results in Section 6 also show that our particular implementation of the Policy Intersection idea outperforms two other algorithms that can be used in a Self-Transfer setting: Probabilistic Policy Reuse (PPR) [11], and Dual Policy Distillation (DPD) [19], that we review in Sections 2.3 and 5 respectively.

While DPD has originally been tested on environments with continuous actions [19], and PPR can easily accommodate continuous control tasks [13], Policy Intersection has until now been restricted to environments with discrete actions [26]. In this paper, we extend the Policy Intersection framework to the continuous actions case. As a result, the two main contributions of this paper are the following:

- We extend the Policy Intersection framework, which was previously restricted to environments with discrete actions, to ones with continuous actions (see Section 3).
- We empirically demonstrate the sample-efficiency gain achieved by performing Self-Transfer in continuous-action settings, using our continuous-actions Policy Intersection algorithm (see Section 6).

2 BACKGROUND

In this section, we introduce the concepts at the basis of our work: Markov Decision Processes, the general inner-workings of Reinforcement Learning algorithms with continuous actions, and two methods to achieve Policy Shaping, Probabilistic Policy Reuse and Policy Intersection.

2.1 Markov Decision Processes

We consider reinforcement learning problems in continuous action spaces. A Markov Decision Process (MDP) with continuous actions is defined by the tuple $\langle S, A, p, R \rangle$: a possibly-infinite set S of states; an infinite set A of d-dimensional continuous actions in $\mathbb{R}^d$; an unknown probability density transition function $p : S \times A \times S \rightarrow [0, \infty)$, and a reward function $R : S \times A \times S \rightarrow r_t \in \mathbb{R}$.

In a continuous action space, a stochastic stationary policy $\pi(s)$ defines, for each state s, a continuous probability distribution from which actions can be sampled. At each time-step, the agent observes s_t, samples an action a_t from $\pi(s_t)$, then observes r_t and s_{t+1}. 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.

2.2 Continuous-Action Reinforcement Learning

In the discrete actions case, the agent's policy π in state s_t is a discrete probability distribution over actions, a vector of $|A|$ real values between 0 and 1, and that sum to 1. Such a discrete policy can be explicitly learned, with Policy Gradient for instance [30], or computed on-the-fly based on learned Q-Values, using an exploration strategy.

In the continuous case on the other hand, enumerating the actions is impossible, as is producing an explicit probability density for each of them. Usually, the agent's policy is implemented as a Gaussian distribution, parameterized by a mean μ and standard deviation σ, these two parameters being output by a neural network given a state s_t. At each timestep, a *single* action a_t is sampled from the Gaussian policy with mean μ and standard deviation σ. Continuous actions are challenging for Reinforcement Learning, and not every algorithm is compatible with them. For instance, every on-line RL algorithm for continuous actions need an explicit actor: there is no critic-only RL algorithm for continuous actions.

In this paper, we use the Soft Actor-Critic (SAC) [15] [2], the current state-of-the-art RL algorithm designed for environments with continuous actions that implements an explicit actor, which, in each state s_t, predicts a mean μ and a standard deviation σ. We extend this algorithm to allow the actor to be advised by an advisor (that produces continuous actions too), and will use this contribution in the Self-Transfer setting that we propose in this paper.

2.3 Policy Shaping by Altering the Exploration Strategy

Policy Shaping lets an external advisory policy π_A alter or determine the agent's behavior at acting time, either sporadically or continuously throughout learning. This general approach, shared with the Safe RL and the learning from human interactions RL subfields [1, 12, 14], can easily be used for Transfer Learning purposes. In this section, we review two methods to shape an RL agent's policy: Probabilistic Policy Reuse [11, 13, PPR], and Policy Intersection [6, 14, originally called "Policy Shaping"]. The first work we know of that introduced the term "Policy Shaping" is [14]. However, the method presented (which we detail below) is very specific, while the term Policy Shaping could apply to a wide variety of methods [17, 21]. Since Griffith et al. [14] introduces an element-wise multiplication between discrete probability distributions, and that this operation represents taking the intersection between the distributions, we rebaptized Griffith's formula Policy Intersection in this paper for clarity. Similarly, a Policy Union algorithm can be achieved by summing the probability distributions, although the qualification and evaluation of this approach is outside the scope of this paper.

[2]The specific SAC implementation we use can be found at: https://github.com/Rafael1s/Deep-Reinforcement-Learning-Algorithms/tree/master/Ant-PyBulletEnv-Soft-Actor-Critic

2.3.1 *Probabilistic Policy Reuse.*

First introduced by [11], the exploration strategy π-reuse (policy reuse) works similarly to ε-Greedy: at each timestep, an action is sampled from an external advisory policy $\pi_A(s_t)$ with probability ψ, or sampled from the currently learned policy $\pi_L(s_t)$ with probability $1-\psi$:

$$a_t \sim \begin{cases} \pi_A(s_t) & \text{with probability } \psi \\ \pi_L(s_t) & \text{with probability } 1-\psi \end{cases} \tag{1}$$

where $\pi_L(s_t)$ is the state-dependent policy learned by the agent, $\pi_A(s_t)$ is the state-dependent advice, and ψ is the probability of sampling an action a_t from the advisor π_A rather than from π_L. Even though PPR only *sometimes* executes an action from π_A, when it does so, π_A fully determines the action executed by the agent. Moreover, PPR has no provision for the advisor to choose, state for state, when to advise the agent and when to let it choose an action itself.

2.3.2 *Policy Intersection.*

This second approach at Policy Shaping multiplies the π_L and π_A vectors together then normalizes the resulting policy vector at each timestep:

$$a_t \sim \pi_L \times \pi_A = \frac{\overbrace{\pi_L(s_t)\,\pi_A(s_t)}^{\text{element-wise product}}}{\underbrace{\pi_L(s_t)\cdot\pi_A(s_t)}_{\sum_{a\in A}\pi_L(a|s_t)\pi_A(a|s_t)}} \tag{2}$$

where $\pi_L(s_t)\cdot\pi_A(s_t)$ is the dot product of the two policies. At acting time, the agent samples an action a_t from the mixture $\pi_L\times\pi_A$ of the agent's current learned policy $\pi_L(s_t)$ and the external advisory policy $\pi_A(s_t)$, instead of sampling only from $\pi_L(s_t)$. This product of probability distribution vectors amounts to taking the intersection between what the current agent's policy wants to do, and what the transferred policy would do in that state. As a result, although this formula was first introduced by [14] as "Policy Shaping", we re-name it Policy Intersection in this paper for clarity. In contrast to PPR, Policy Intersection allows π_L and π_A to more cooperatively select actions, with π_A able to increase or decrease the probability of actions, but without fully determining the action. However, Equation 2 can only be applied to tasks for which there is a finite set of actions, excluding environments with a continuous action space.

The Actor-Advisor [26] is the first attempt made at using the Policy Intersection formula in Equation 2 to achieve Transfer Learning. Their main contribution is to directly influence a Policy Gradient agent's policy with some off-policy external advice π_A without convergence issue. However, that approach is also limited to discrete actions.

2.4 Dual Policy Distillation

One of the closest existing technique to our contribution, that is using more than one actor, is Dual Policy Distillation [19]. DPD launches two agents (both using the same RL algorithm, either DPG or PPO agents in the original paper) at the same time in *two instances of the same environment*, but with different initializations of their neural network weights. As the agents learn to solve the same task in parallel, they optimize a distillation objective that incites the policies of the two agents to remain close to each other. A theoretical justification of why the two policies complement each other knowledge-wise is provided in [19]: as with our Self-Transfer setting, having more than one policy intervening in the action selection procedure (or learning procedure) limits convergence to poor local optima. However, methods leveraging more than one agent/actor have two potential downsides: i) their deployment to real-life settings is likely to be more difficult if they require the two actors to execute actions *concurrently*, in two instances of the environment, and ii) such methods assume that all the policies are RL agents, hence one cannot be easily replaced by a fixed transferred policy. Algorithms such as PPR and PI do not make such assumptions about the advisor policy, allowing it to be from any source.

We now detail how Policy Intersection can be extended to RL algorithms designed for environments with continuous actions.

3 POLICY INTERSECTION FOR CONTINUOUS ACTION SPACES

We now present our first contribution, a Policy Intersection algorithm compatible with continuous actions.

Equation 2 expresses the product of two state-dependent probability distributions: the actor's $\pi_L(s_t)$, and the advisor's $\pi_A(s_t)$. In the discrete actions case, $\pi_L(s_t)$ and $\pi_A(s_t)$ are both vectors of size $|A|$, where A is a finite set of actions available to the agent. Multiplying those two vectors amounts to taking the intersection between those two probability distributions. In other words, Policy Intersection samples actions that both the actor and the advisor "agree on", as these actions have a non-zero probability in both $\pi_L(s_t)$ and $\pi_A(s_t)$ [3]. Unfortunately, when the action space is continuous, finding actions in the intersection between the actor's and the advisor's policies is less straightforward.

In Reinforcement Learning algorithms that can be applied to environments with continuous actions, such as Proximal Policy Optimization (PPO) [29] and Soft Actor-Critic (SAC) [15], individual actions can directly be sampled from the actor, but is it difficult to access a probability distribution for all possible actions, as there is an infinite number of actions. The actor can also provide, given a particular action, the probability density of that action. Assuming that the advisor can similarly be sampled, our approach to sampling the intersection between which actions are allowed by the actor and those allowed by the advisor is the following:

(1) At action selection time, we sample a large amount of actions A^A from the advisor (where the superscript 'A' stands for advisor).
(2) We then "submit" A^A to the actor π_L, which returns probability densities $\pi_L(s_t)$. For each action $a_i^A \in A^A$, we now have a corresponding density $\pi_L(a_i^A|s_t)$.
(3) We transform each probability *density* into a *possibility* [34], according to Castineira et al. [5, Lemma 3.1 on page 305], by computing $p_i = \frac{\pi_L(a_i^A|s_t)}{max_{a \in A^A} \pi_L(a|s_t)}$.
(4) We execute in the environment the first action a_i^A such that $p_i > p \sim U(0,1)$, with p a uniformly-sampled random number in $[0,1]$.

This method implements an "AND" between the actor and the advisor's policies. We basically consider that Policy Intersection computes the product of independent probabilities in the discrete-action case, note that it is equivalent to computing the probability of the "AND" of two events, and implement that "AND" event in the continuous-action case, where the product is impossible to efficiently implement.

While our sampling technique allows to compute the intersection between an actor and an advisor, with no assumption about their distributions (we do not assume that they are Gaussian, for instance), we acknowledge that alternative approaches could have been devised, such as creating a new stochastic policy resulting of the point-wise multiplication of the advisor's probability density function with the actor's probability density function, sampled for a large number of actions. This method, however, requires an advisor that can produce density values for arbitrary actions, while ours merely requires an advisor that can sample actions.

Finally, thanks to Lemma 3.1 in [5, p. 305], we could interpret the distributions π_L and π_A as possibility distributions rather than probability distributions. Intuitively, the possibility distribution expresses to what extend a certain action fits the label "acceptable action" for the given state. A common practice in possibility theory, which finds its roots in fuzzy set theory [34], is to use the *minimum* operator (instead of element-wise multiplication or probabilistic "AND") to express the intersection between two possibility distributions. We leave the evaluation of such a *minimum*-based intersection to future work.

[3]Note that if no agreement can be made, i.e., the intersection between which action the actor and the advisor want to choose is empty, it can be arbitrarily decided that either the advisor or the actor has the last word, depending on the problem. In this paper, we choose to always give the last word to the actor.

Algorithm 1 Policy Intersection for Continuous Actions

```
Require: $\pi_L$ is the currently learning actor, and $\pi_A$ is the frozen actor used as advisor
  $A^A$ = set of actions sampled from $\pi_A(s_t)$
  for every action $a_i^A \in A^A$ do
    Get probability $\pi_L(a_i^A|s_t)$
    Compute possibility $p_i = \frac{\pi_L(a_i^A|s_t)}{max_{a \in A^A} \pi_L(a|s_t)}$ (see text)
    if $p_i > p \sim U(0,1)$ then
      Execute action $a_i^A$
      return
    end if
  end for
  Execute action $a \sim \pi_L(s_t)$
```

4 SELF-TRANSFER

Our second contribution is the Self-Transfer framework, in which two policies sequentially learned in the same environment are used in a Transfer Learning setting, to increase sample-efficiency and final policy quality. This contribution builds on the Continuous Policy Intersection algorithm presented in Section 3.

Our Self-Transfer procedure works as follows: an RL agent (using Soft Actor-Critic, in this work) is launched in an environment as usual. After a given amount of episodes I (at the designer's

Algorithm 2 Self-Transfer

```
Require: $I$ is the amount of episodes reserved to training the advisor $\pi_A$
  Initialize SAC critics and actor $\pi_L$
  for every episode $e = 1..\infty$ do
      if $e = I$ then
          $\pi_A \leftarrow$ a copy of $\pi_L$
          Reset $\pi_L$ and critic networks
      end if
      for every time-step $t$ until the end of episode do
          if $\pi_A$ exists then
              Sample $a_t$ with Policy Intersection($\pi_L, \pi_A, s_t$)
          else
              Sample $a_t$ from $\pi_L(s_t)$
          end if
          Periodically learn using the standard SAC equations
      end for
  end for
```

discretion), the agent's learning is interrupted, and its actor π_L is deep-copied into π_A. Then, π_L and, for actor-critic algorithms such as SAC, the critics, are re-initialized to fresh random weights. Learning then resumes, except that the fresh agent π_L is now advised by π_A, using the Policy Intersection algorithm described in Section 3. Pseudocode for the Self-Transfer setting is shown in Algorithm 2.

5 RELATED WORK

Before providing empirical evidence that our Self-Transfer framework, based on our Continuous Policy Intersection algorithm, leads to higher sample-efficiency and final policy quality than vanilla Soft-Actor Critic, we review related work.

An intuitive way to help an RL agent learn to perform tasks in a more sample-efficient manner is Transfer Learning. Transfer Learning aims at making an RL agent learn new tasks faster by allowing it to reuse previously learned knowledge efficiently. A Transfer Learning setting often involves a *source* task and a *target* task; first, the agent learns the source task, then the knowledge acquired in the source task is intelligently leveraged by the agent while tackling the target task, using a Transfer Learning algorithm [32, 36]. Transferred knowledge can be the agent's learned policy or Q-values [3, 11, 33]; learned skills or options [2, 18, 28], some general enough skill (like walking) to fit a large set of target tasks [31]; parts of a modular neural network policy in which each module deals with a different aspect of the task [10, 22]. Methods to effectively transfer knowledge include reward shaping [4], policy reuse methods, which shape the agent's exploration strategy [11, 14], initializing a policy [33] using information from the source task, and initializing parts of the target network with the source network [7, 10, 22], to only cite a few. When it comes to the specific algorithms we use, i.e., Probabilistic Policy Reuse and Policy Intersection, they can both be labelled as Transfer Learning via guiding the agent's exploration [12]. PPR [11] has specifically been though out for Transfer Learning purposes; PI [14], on the other hand, was first introduced as a way to incorporate human interventions in the RL agent's learning process to improve its performance.

However, due to the relatively odd nature of the particular setting in which we use PPR and PI, i.e., the transfer of knowledge acquired in a given task to the same task, it is a little difficult to find directly comparable existing work. Existing Transfer Learning work usually focuses on transferring knowledge across tasks with different goals or differing environmental dynamics [11, 27, 33, 35, only a few examples], while we transfer from a task in a given environment to the same task, in the same environment; only the initialization of the agent's networks changes. Methods using more than one actor to improve exploration of a given environment are somewhat related to Self-Transfer, such as A3C [23], Multi-Agent RL settings in which agents actively share knowledge with each other [9, 16, 25], and Dual Policy Distillation (DPD) [19] (see Section 2.4).

In contrast to DPD, but similarly to Self-Transfer, Self-Imitation Learning [24] does not require a second agent to improve exploration; the agent learns to reproduce its own past good decisions to deepen exploration. In [24], a preference for previously chosen actions that lead to a greater return than the current value estimate for a given state is integrated in an actor-critic loss. Although this concept is very close to ours, Self-Transfer leverages knowledge from past experiences produced by an agent with a different initialization than the agent currently learning, while Self-Imitation never resets the agent. We suspect that agents with different initialization can lead to distinct, complementary experiences of the same environment, and thus achieve a more thorough exploration when joining knowledge.

6 EXPERIMENTS

We present a preliminary evaluation of our contribution by applying Self-Transfer on three Pybullet [8] continuous control environments: Ant, a four-legged insect-like creature; Half-Cheetah, a two-legged creature; and Hopper, a single disembodied leg hopping away. In these three environments with continuous actions, the goal is generally to learn to move forward as fast as possible, without falling. We evaluate our Self-Transfer setting based on two Policy Shaping algorithms: our Continuous Policy Intersection, and Probabilistic Policy Reuse (see Section 2.3). For PPR, we tested two values for ψ, the probability in each timestep to sample the advisor's policy: 0.1 and 0.2. Setting ψ to 0.1 leads to the best results, therefore we only show the results generated by that configuration in Figures 1 and 2. For Continuous Policy Intersection, the size of the vector A^A of actions sampled from the advisor, then submitted to the actor, is set to 4096.

Results are reported in Figure 1. Each line is produced by averaging the results of 8 random seeds, with the 95% confidence interval shown as shaded regions. Both when the advisor is trained for 100 episodes and 200 episodes, Policy Intersection outperforms Probabilistic Policy Reuse. In the Ant environment, PPR with a 200 episodes trained advisor shows a good jumpstart until approximately 1500 episodes; our Continuous PI with a 100 episodes trained advisor exceeds PPR's performance after that. Moreover, Self-Transfer with Policy Intersection brings a non-negligible performance gain compared to not using Self-Transfer, even while helped by an advisor trained during only 100 episodes. This performance gain is present even when the advisor does not have enough training time to learn how to produce rewards above zero, as shown in the bottom plot (for the Hopper environment), and an advisor trained for 100 episodes seems more beneficial than one trained for double that time in that particular environment.

We also report results obtained by Dual Policy Distillation in Figure 1, although these are not directly comparable to PI and PPR since they were produced by the implementation provided with the original paper [4].

When it comes to the positive difference in performance brought by each individual method to their baseline, our Continuous Policy Intersection outperforms both Self-Transfer using PPR, and DPD. Figure 2 shows for each algorithm the difference between its performance and the performance of its baseline. For instance, the "DPD DDPG" curve is obtained by, for each timestep, subtracting the reward obtained by the original Deep Deterministic Policy Gradient (DDPG) [20] algorithm from that obtained when DPD is enabled. Similarly, the curves for PPR and PI are computed by subtracting the reward obtained by SAC alone from that when Self-Transfer is used. We kept the configurations for PPR and PI that led to their best results, namely 200 advisor training episodes for PPR and 100 advisor training episodes for PI. In contrast to learning curves as in Figure 1, computing this gain achieved by a given extension of an RL algorithm allows extensions implemented on different baselines to be fairly compared. However, we do not think that this completely excludes the need to compare extensions once they are all implemented on top of the same baseline.

The gain brought by Self-Transfer in general, be it using PPR or our Continuous PI is especially visible on the Ant environment, while DPD brings a much smaller gain slightly above zero. However, it seems much harder for all three algorithms to increase the rewards of their baseline on Half-Cheetah and Hopper. In particular, DPD starts off with a positive difference during the first half of the experiment on Half-Cheetah, then worsen DDPG's performance during the other half. PPR very negatively impacts performance in the Hopper environment, while PI and DPD manage to generally remain close to zero if not slightly above. Our Continuous PI is the only algorithm leading to a non-negligible improvement of its baseline in both Ant and Half-Cheetah, and manages to keep its impact either null or positive in Hopper.

7 DISCUSSION AND FUTURE WORK

In this paper, we present Self-Transfer, a scheme leveraging conventional Transfer Learning algorithms to allow an RL agent to improve its performance at solving a given task. Self-Transfer does not require a second instance of the agent learning in parallel, nor a second, separate instance of the environment. The only cost of performing Self-Transfer is an extra training time; we empirically showed in our preliminary experiments that dedicating less than 5 % of the total training time needed to learn the task is enough to gain a significant performance improvement.

As suggested by [19], two agents with a different initialization launched in the same environment can gather a distinct experience of that environment, and can improve each other's exploration by exchanging knowledge. Hence, it is likely that the performance gain achieved by our contribution is a result of an improved exploration, as we randomly reset the agent's networks weights after

[4]This implementation uses Deep Deterministic Policy Gradient (DDPG) [20] instead of Soft Actor-Critic; we did not re-implementing DPD on top of SAC as this consists in a non-trivial task considering the complexity of the DPD algorithm.

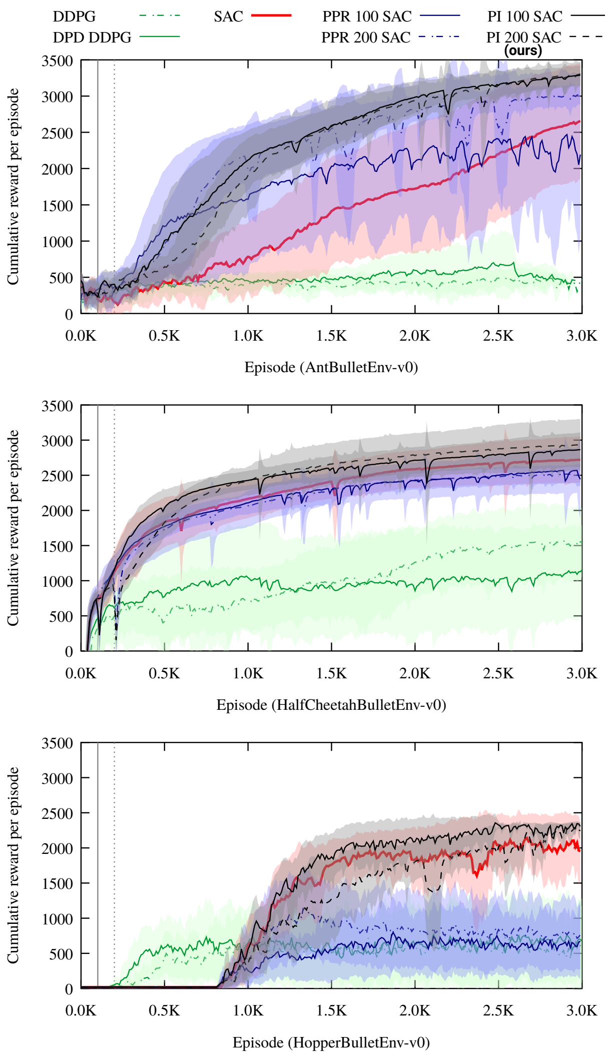

Figure 1: Comparison between Soft Actor-Critic with no Self-Transfer, and using Self-Transfer with either Probabilistic Policy Reuse (PPR) or Policy Intersection (PI). These curves are averaged over 8 runs per algorithm. The two vertical lines indicate when the current actor is saved as a frozen advisor, and that the current agent is replaced with a fresh agent. In all three PyBullet environments, Self-Transfer with our Continuous Policy Intersection algorithm significantly outperforms its baseline (SAC) and Self-Transfer with Probabilistic Policy Reuse. Results obtained by Dual Policy Distillation on the three PyBullet environments are also shown. However, DPD's results cannot directly be compared to that of PPR and PI since they are implemented on top of SAC, while DPD is using Deep Deterministic Policy Gradient (DDPG), which seems to learn these tasks less well than SAC.

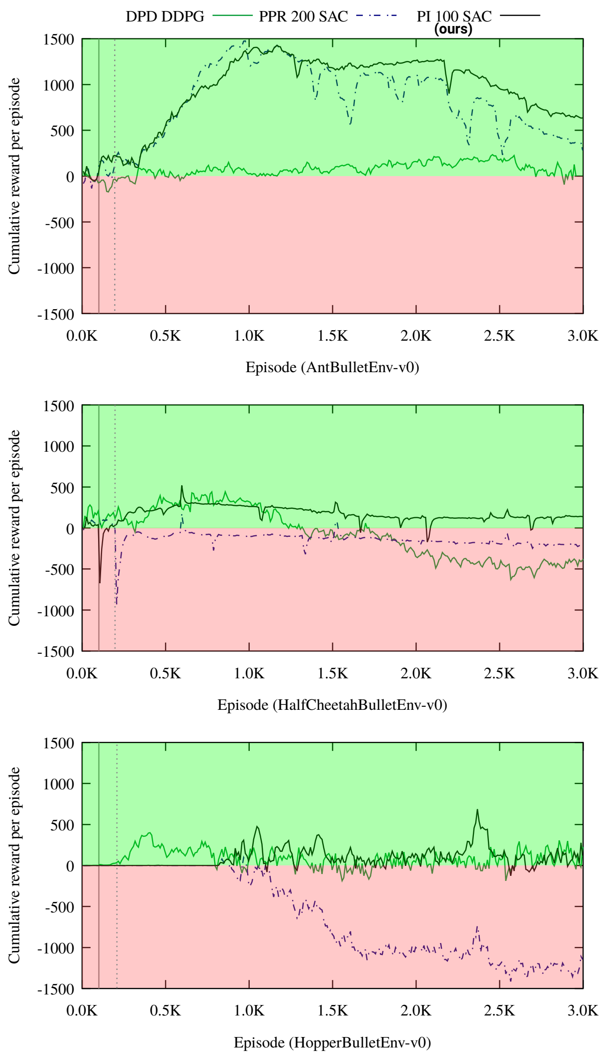

Figure 2: Comparison between the performance gain brought by DPD to its baseline DDPG, the gain brought by PPR to SAC and by our Continuous PI to SAC in each PyBullet environment. These curves are averaged over 8 runs per algorithm. The performance gain is computed by subtracting the reward obtained by the baseline (in this case, either SAC or DDPG) from the reward obtained by the baseline augmented with the transfer or distillation algorithm for each timestep of the experiment. Hence, we can see the impact of each algorithm over time and how much they concretely improve their baseline. Moreover, this allows for a fair comparison between algorithms implemented on top of different baselines. Out of all three algorithms, our Continuous PI is the only one that either consistently significantly improves its baseline, or that does not decrease its baseline performance.

the advisor has been trained. A potentially interesting experiment to verify that hunch could be to have the advisor and the advised agent have the exact same initialization, and compare that to distinct individual initialization. In addition, as future work, we will compare Self-Transfer to Dual Policy Distillation and Self-Imitation on the same reinforcement learning algorithm, and investigate the potential of learning to imitate an agent with a different initialization. Finally, we will compare our current implementation of Policy Intersection for continuous action spaces to the use of the *minimum* operator from fuzzy set theory on the actor's and advisor's possibility distributions.

ACKNOWLEDGMENTS

If you wish to include any acknowledgments in your paper (e.g., to people or funding agencies), please do so using the '`acks`' environment. Note that the text of your acknowledgments will be omitted if you compile your document with the '`anonymous`' option.

REFERENCES

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[2] David Andre and Stuart J Russell. 2002. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*. 119–125.
[3] Tim Brys. 2016. *Reinforcement Learning with Heuristic Information.* Ph.D. Dissertation. PhD thesis, Vrije Universitet Brussel.
[4] Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. 2015. Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 181–188.
[5] Elena Castineira, Susana Cubillo, and Enric Trillas. 2007. On the coherence between probability and possibility measures. (2007).
[6] Thomas Cederborg, Ishaan Grover, Charles L Isbell Jr, and Andrea Lockerd Thomaz. 2015. Policy Shaping with Human Teachers.. In *IJCAI*. 3366–3372.
[7] Devendra Singh Chaplot, Guillaume Lample, Kanthashree Mysore Sathyendra, and Ruslan Salakhutdinov. 2016. Transfer deep reinforcement learning in 3d environments: An empirical study. In *NIPS Deep Reinforcemente Leaning Workshop*.
[8] Erwin Coumans and Yunfei Bai. 2016–2019. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.
[9] Felipe Leno da Silva, Ruben Glatt, and Anna Helena Reali Costa. 2017. Simultaneously Learning and Advising in Multiagent Reinforcement Learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (São Paulo, Brazil) *(AAMAS '17)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1100–1108.
[10] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. 2017. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2169–2176.
[11] Fernando Fernández and Manuela M. Veloso. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *International Conference on Autonomous Agents and Multiagent Systems*.
[12] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.
[13] Javier García and Fernando Fernández. 2019. Probabilistic policy reuse for safe reinforcement learning. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 13, 3 (2019), 1–24.
[14] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. 2013. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. In *Neural Information Processing Systems*.
[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).
[16] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. 2016. Cooperative Inverse Reinforcement Learning. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc., 3909–3917. https://proceedings.neurips.cc/paper/2016/file/c3395dd46c34fa7fd8d729d8cf88b7a8-Paper.pdf
[17] Anna Harutyunyan, Tim Brys, Peter Vrancx, and Ann Nowé. 2014. Off-policy shaping ensembles in reinforcement learning. *arXiv preprint arXiv:1405.5358* (2014).
[18] George Konidaris and Andrew G Barto. 2007. Building Portable Options: Skill Transfer in Reinforcement Learning.. In *IJCAI*, Vol. 7. 895–900.

[19] Kwei-Herng Lai, Daochen Zha, Yuening Li, and Xia Hu. 2020. Dual Policy Distillation. *arXiv preprint arXiv:2006.04061* (2020).

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[21] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, Guan Wang, David L Roberts, Matthew E Taylor, and Michael L Littman. 2017. Interactive learning from policy-dependent human feedback. In *International Conference on Machine Learning*. PMLR, 2285–2294.

[22] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. 2018. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*. 2419–2430.

[23] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[24] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. 2018. Self-imitation learning. *arXiv preprint arXiv:1806.05635* (2018).

[25] Shayegan Omidshafiei, Dong-Ki Kim, Miao Liu, Gerald Tesauro, Matthew Riemer, Christopher Amato, Murray Campbell, and Jonathan P How. 2019. Learning to teach in cooperative multiagent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 6128–6136.

[26] Hélène Plisnier, Denis Steckelmacher, Diederik M Roijers, and Ann Nowé. 2019. The Actor-Advisor: Policy Gradient With Off-Policy Advice. *arXiv preprint arXiv:1902.02556* (2019).

[27] Hélene Plisnier, Denis Steckelmacher, Diederik M Roijers, and Ann Nowé. 2019. Transfer Reinforcement Learning across Environment Dynamics with Multiple Advisors.. In *BNAIC/BENELEARN*.

[28] Balaraman Ravindran and Andrew G Barto. 2003. Relativized options: Choosing the right transformation. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 608–615.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[30] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. 1999. Policy gradient methods for reinforcement learning with function approximation.. In *Neural Information Processing Systems (NeurIPS)*, Vol. 99. 1057–1063.

[31] Haoran Tang and Tuomas Haarnoja. [n.d.]. Learning Diverse Skills via Maximum Entropy Deep Reinforcement Learning. *URL http://bair. berkeley. edu/blog/2017/10/06/soft-q-learning* ([n. d.]).

[32] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* (2009).

[33] Matthew E Taylor, Peter Stone, and Yaxin Liu. 2007. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research* 8, Sep (2007), 2125–2167.

[34] Lotfi Asker Zadeh. 1978. Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and systems* 1, 1 (1978), 3–28.

[35] Amy Zhang, Harsh Satija, and Joelle Pineau. 2018. Decoupling dynamics and reward for transfer learning. *arXiv preprint arXiv:1804.10689* (2018).

[36] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. 2020. Transfer Learning in Deep Reinforcement Learning: A Survey. arXiv:2009.07888 [cs.LG]