Université de Lille
École Doctorale MADIS

A dissertation submitted in partial satisfaction of the
requirements for the degree of
DOCTOR OF PHILOSOPHY
in **Computer Science**
submitted by
Léonard Hussenot

APPRENTICESHIP LEARNING: TRANSFERRING HUMAN MOTIVATIONS TO ARTIFICIAL AGENTS

---

under the supervision of **Olivier Pietquin & Philippe Preux**.

Publicly defended on **December 14th, 2022** at **Villeneuve d'Ascq**, to the doctoral committee

Damien **Ernst**, Professor	Université de Liège	Thesis Jury President
Pierre-Yves **Oudeyer**, Research Director	Inria	Thesis Referee
Emmanuel **Rachelson**, Associate Professor	ISAE-SUPAERO	Thesis Referee
Martha **White**, Associate Professor	Alberta University	Thesis Jury
Olivier **Pietquin**, Professor	Google Research	Thesis Supervisor
Philippe **Preux**, Professor	Inria, Université de Lille	Thesis Supervisor
Matthieu **Geist**, Professor	Google Research	Invited Member
Olivier **Bachem**, Doctor	Google Research	Invited Member

Research Center in Computer Science, Signal and Automatic Control of Lille (CRIStAL),
UMR 9189 Scool team, 59650, Villeneuve d'Ascq, France

Université de Lille
École Doctorale MADIS

Thèse de Doctorat

Spécialité **Informatique**

présentée par
Léonard Hussenot

Apprentissage par Démonstrations: transfert des motivations humaines aux algorithmes

---

sous la direction de **Olivier Pietquin** & **Philippe Preux**.

Soutenue publiquement le **14 décembre 2022** à **Villeneuve d'Ascq** devant le jury composé de

Damien **Ernst**, Professor	Université de Liège	Président du Jury
Pierre-Yves **Oudeyer**, Research Director	Inria	Rapporteur
Emmanuel **Rachelson**, Associate Professor	ISAE-SUPAERO	Rapporteur
Martha **White**, Associate Professor	Alberta University	Examinatrice
Olivier **Pietquin**, Professor	Google Research	Directeur de thèse
Philippe **Preux**, Professor	Inria, Université de Lille	Directeur de thèse
Matthieu **Geist**, Professor	Google Research	Invité
Olivier **Bachem**, Doctor	Google Research	Invité

Centre de Recherche en Informatique, Signal et Automatique de Lille (CRIStAL),
UMR 9189 Équipe Scool, 59650, Villeneuve d'Ascq, France

Abstract

Reinforcement learning is a generic mathematical and algorithmic framework which aims at building artificial agents that experience their environment and improve over time to maximize their overall outcome. It permits to tackle various sequential decision making problems like robotics, board and video games or self-driving vehicles. Yet reinforcement learning agents require a large amount of interactions with their environment to learn, and it is often hard to design a reward function that is precisely aligned with what we expect the agent to do. *Apprenticeship learning* tackles this problem by not only relying on the reward function but also on additional sources of knowledge. Particularly, *learning from demonstrations* makes use of examples of how to solve the task at hand. How to make the most of such demonstrations, in particular when they are produced by a human being, is an open question. This thesis presents practical contributions to apprenticeship learning. We first focus on imitation learning, where the goal is to mimic the demonstrations that are considered as optimal. We investigate how to properly design an adversarial imitation learning algorithm. We give recommendations on the various choices to make when designing such an agent and we highlight differences between mimicking synthetic and human demonstrations. To bypass the brittleness of these methods (a consequence of their mathematical formulation), we design a new imitation learning algorithm. By deriving an upper-bound of an optimal-transport distance, we avoid the saddle-point optimization and obtain a simple algorithm with very little hyperparameters. We show its strong performance on simulated robotic tasks in the very low data regime. We then study how to select hyperparameters in the context of imitation learning, without access to the reward function. We use, as test-bed, the two aforementioned algorithms as well as the standard supervised learning approach, behavioral cloning. In a second part, in order to make the most of both demonstrations and a reward function, we design an algorithm that enables the transfer of intrinsic motivation from the demonstrations to the agent, improving its exploration of the environment. We show that the learned motivation, extracted from the data, carries out information from the demonstrations. Finally, we show that one can transfer human incentives in a different manner, by learning a state-dependent discretization of the action space of the agent. We show the efficiency of the resulting algorithms on a variety of robotic tasks, using human demonstrations as well as human "play-data", with or without the reward function.

Résumé

L'apprentissage par renforcement est un cadre mathématique et algorithmique générique qui vise à developper des algorithmes qui interagissent avec leur environnement et s'améliorent au fil du temps pour maximiser leur recompense sur le long terme. Il permet d'aborder divers problèmes de prise de décision séquentielle comme la robotique, les jeux de plateaux ou jeux vidéo ou encore les véhicules autonomes. Cependant, ces agents d'apprentissage par renforcement nécessitent une grande quantité d'interactions avec leur environnement pour apprendre, et il est souvent difficile de concevoir une fonction de récompense qui soit précisément alignée sur ce que nous attendons de lui. L'*apprenticeship learning* s'attaque à ce problème en s'appuyant non seulement sur la fonction de récompense, mais aussi sur des données supplémentaires. Typiquement, ces données se présentent sous la forme de démonstrations. Comment tirer le meilleur parti de telles démonstrations, en particulier lorsqu'elles sont produites par un être humain, est une question ouverte. Cette thèse présente des contributions pratiques à l'apprenticeship learning. Nous nous intéressons d'abord à l'apprentissage par imitation, où le but est de mimer les démonstrations considérées comme optimales. Nous étudions comment concevoir un algorithme d'apprentissage par imitation adversariale. Nous donnons des recommandations sur les différents choix à faire lors de la conception d'un tel agent et nous soulignons les différences entre imiter des démonstrations synthétiques et humaines. Pour contourner la fragilité de ces méthodes (conséquence de leur formulation mathématique), nous concevons un nouvel algorithme d'apprentissage par imitation. En dérivant une borne supérieure d'une distance de transport optimale, nous évitons l'optimisation du point de selle et obtenons un algorithme simple avec peu d'hyperparamètres. Nous démontrons ses performances sur des tâches robotiques simulées dans un régime de données très faible. Nous étudions ensuite comment sélectionner des hyperparamètres dans le cadre de l'apprentissage par imitation, sans accès à la fonction de récompense. Nous utilisons pour cela les deux algorithmes susmentionnés ainsi que l'approche standard d'imitation par apprentissage supervisé. Dans une seconde partie, afin de tirer le meilleur parti à la fois des démonstrations et de la récompense, nous concevons un algorithme qui permet le transfert de la motivation intrinsèque des démonstrations à l'agent, améliorant ainsi son exploration de l'environnement. Nous montrons que la motivation apprise, extraite des données, transmet de l'information venant des démonstrations. Enfin, nous montrons que l'on peut transférer les incitations humaines d'une manière différente, en apprenant une discrétisation état-dépendante de l'espace d'action de l'agent. Nous montrons l'efficacité des algorithmes résultants sur une variété de tâches robotiques, en utilisant des démonstrations humaines ainsi que des données de jeux (sans objectif défini) humaines, avec ou sans fonction de récompense.

Contents

Chapter 1

Introduction

> *"By three methods we may learn wisdom: first, by reflection, which is noblest; second, by imitation, which is easiest; and third by experience, which is the bitterest."*
>
> — Confucius, *Analects 16:19*

How smart a decision is can often only be judged in the long run. In a game, in one's work, or in one's life, every decision can have consequences far beyond the moment it was taken. Behaving in a way that takes the future into consideration is a key part of *intelligence*. If recent advances in machine learning have shown amazing capabilities for single step predictions, like understanding the content of a picture, transcribing speech to text, or predicting the shape of a protein, designing algorithms able to adapt their behavior to consider future outcomes is arguably one of the greatest challenges of modern artificial intelligence research. We usually name this problem *sequential decision making*.

Dopamine, a molecule synthesized in human and animal brains and kidneys, has been shown to be one of the regulators of reward-motivated behaviors [Berridge, 2007]. In particular, the secretion of dopamine in anticipation of a rewarding event or in response to a reward higher than expected [Montague et al., 1996] suggests that humans and animals are able to continuously learn behaviors that take into account the influence of their actions on a future outcome.

While Richard E. Bellman had laid the foundation of *reinforcement learning* in the seminal *Dynamic Programming* of 1966, the understanding of these biological phenomena shed a new light on his work. Reinforcement learning, as formalized later by Sutton et al., is both a mathematical and an algorithmic paradigm that allows to train agents to experience their environment and learn a behavior that maximizes the overall reward they obtain over time. In doing so, they learn to avoid actions that will be detrimental in the future and to act in a way that improves their final outcome. Most algorithms referenced in this thesis heavily rely on Bellman's work, but their development often took deep inspiration from psychology and biology.

Reinforcement learning is generic and allows to tackle any sequential problem for which a reward function can be computed. As the examples and references provided along this thesis will show, the resulting algorithms are capable of learning robotic tasks, of playing board and video games, of driving vehicles, or of controlling plasma in a tokamak.

Yet reinforcement learning still suffers many issues. Notably, algorithms require a massive amount of interactions with their environment to learn a correct behavior. In comparison, humans are much more efficient learners. One hypothesis to explain this discrepancy is that humans do not solely rely on reward

to learn. There are other mechanisms that provide them with learning signal. Interestingly, Byrne et al. showed that apes are able to learn various skills by *imitation*, without giving them any external reward (think food, hugs etc...). Imitative learning is also widely leveraged in human education and provides a complementary approach to reward seeking. It is particularly interesting for complex tasks in which the reward is very sparse. For example, one would not think about teaching children how to play chess by just telling them "good" when they win a game or "bad" when they lose it. Providing demonstrations on how to play is a central phase of the learning process.

Leveraging demonstrations is a key challenge towards making reinforcement learning agents more efficient, but it also has a wider impact. It notably allows to tackle problems for which a reward function cannot be written explicitly, in a closed-form.

Indeed, in many applications, the goal is hard to specify through a reward function. For example, designing a reward function for "driving well" would be extremely hard. It would have to take into account safety, smoothness, duration, interactions with other road users, so that maximizing it leads to the perfect driving behavior. Designing such a reward function could be as hard as directly designing the behavior itself. Another example could be dialog systems, for which defining a reward function that tells how "human-like" an interaction is, may be very hard. In such cases, setting the goal purely in terms of imitation may be extremely useful.

In some other cases, a sparse reward can easily be implemented, for example describing if one won $(+1)$ or lost (-1) in a game. But learning from such a sparse signal is extremely challenging as it implies a complex *credit assignment*. Understanding which past actions led to victory or defeat is, in these cases, very hard as the relation between an action at a given time – *e.g.* moving a pawn at a the beginning of the game– and the final outcome –checkmating the opponent's king– is complex.

This thesis stems from the idea that human demonstrations, whether optimal or not, implicitly contain a lot of information. Not only on the task to solve but also on how to solve it and on the constraints to follow while solving it. Imagine training a robot to play a given sport. While the notion of victory is easy to implement in a reward function, giving the robot a sense of *fair-play* is not straightforward. It is a complex behavior, resulting from ethos like sportsmanship and honor for which designing reward functions is impossible and usually even have negative consequences with respect to the actual goal of the game. Nonetheless, such behaviors are absolutely necessary if we ever want human-robot interactions in such context. Rather than designing a reward function that encourages fair-play, demonstrating such behavior is arguably the only way to induce such complex behavior in artificial agents.

1.1 Overview

Part I, chapters 3, 4, 5 of this thesis focus on imitation learning, which consists in learning to imitate the behavior of another (potentially human) agent, given a finite set of examples. The paradigm can easily be cast in a supervised learning problem, by learning to predict actions from states in order to mimic the demonstrations. But such approaches do not take into account the dynamics of the environment. They ignore the fact that a mistake in a given state can lead to a state that may not be covered by the demonstrations. Neglecting this dynamical aspect leads to accumulating errors and diverging behaviors, as making a mistake at a given moment increases the chance of making a mistake down the road. Several remedies have been proposed by the literature to incorporate this dynamic aspect. A potential one is the use of adversarial imitation learning. This framework lets the agent interact with the environment and rewards it for "fooling" a classifier that is simultaneously trained to discriminate between the agent's

behavior and the demonstrations. Numerous methods were developed to enhance this technique, yet very little studies properly compared these various options in a controlled setup. We thus study how adversarial imitation learning methods compare in a systematic way. We highlight several recommendations on how to design different parts of the algorithm, like the choice of discriminator regularization, that has a key role in preventing the discriminator from overfitting the provided data, the choice of reward function derived from the discriminator or the choice of direct reinforcement learning agent. We also underline important differences between imitating a human agent and an artificial one, showing flaws in current evaluations of imitation learning algorithms.

One notorious shortcoming of these adversarial imitation learning methods is that they suffer from a brittleness arising from their mathematical formulation. They indeed have to solve a saddle-point optimization problem, which is notoriously unstable. We thus propose a new imitation learning algorithm that alleviates the need for such optimization through a reformulation of the optimal transport problem. We derive an upper-bound on the classical Wasserstein-1 distance between the distribution of agent's state-action pairs and demonstrations' one, then train a reinforcement learning agent to minimize it. We show how competitive this imitation learning algorithm is in the very-low data regime, training in simulation a humanoid to walk with no more than a single demonstration. This new algorithm has the large advantage of having very little hyperparameters, *i.e.* parameters that are not learnt and have to be tweaked by practitioners for every new problem. It makes it very practical and easy to use.

Most imitation learning algorithms have a large number of these *hyperparameters* that users have to change every time they train the algorithm on a new problem, with very little prior on what are the correct values. This hinders the real-world usability of these algorithms. The empirical studies of each of these algorithms generally rely on unprincipled hyperparameter selection principles. They use information –the reward signal– that is, by assumption, not available in the imitation learning setup. We thus study surrogate ways of selecting hyperparameters that respect the imitation learning constraints and improve imitation learning algorithms usability. We first propose to select hyperparameters using proxy metrics and find results suggesting that the usage of optimal transport distances is the best solution. We then study how well hyperparameters transfer between environments and highlight the difficulties of this setup.

Part II, chapters 6, 7 of the thesis tackle reinforcement learning from demonstrations, a setup in which, in addition to a reward function, the experience from another agent (like a human) can be leveraged to accelerate the learning process. The goal here is not to purely imitate the demonstrations but to make the most of them to maximize, as efficiently as possible, the –possibly sparse– reward function.

We first show how one can transfer motivations from demonstrations to the agent. In particular, we focus on extracting the intrinsic motivation of the demonstrator to provide priors on how to explore to the agent. Intrinsic motivation is the part of the behavior driven by intrinsic rewards rather than extrinsic rewards. Humans usually generate their own intrinsic reward, like curiosity, fear, or the spirit of competition. They drive our way to interact with our environment and define our own goals. It has become quite common in reinforcement learning to explicitly implement intrinsic motivations to steer the agent's exploration of its environment, in particular when the reward is sparse.Instead of mathematically and explicitly modelling what we think are human intrinsic motivation –like curiosity–, this thesis proposes that we learn those motivations from actual demonstrations, and to demonstrates how they convey knowledge from the demonstrator.

Ultimately, we introduce a new algorithm for continuous-control that enables to transfer the human's grasp of the environment to the reinforcement learning agent. Which allows us to learn a state-dependent discretization of the action space from human demonstrations. By learning candidate actions on the demonstrations, we restrict the exploration of the agents to meaningful actions and allow the use of

discrete reinforcement learning agents on continuous control tasks. We finally show that the resulting agents are extremely efficient at solving challenging robotics tasks that were not solvable before. We demonstrate that, although simple in its conception, the algorithm has very strong performances across a variety of tasks and setups, with or without reward functions, with expert or "play" demonstrations.

1.2 Publications & Preprints

The elements and results presented in this thesis have been published or open-sourced in the following works:

- Manu Orsini*, Anton Raichuk*, **Léonard Hussenot***, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. **What Matters for Adversarial Imitation Learning?**, *Advances in Neural Information Processing Systems, 34, 2021.* [Orsini et al., 2021]. Covered in Chapter 3.
- Robert Dadashi, **Léonard Hussenot**, Matthieu Geist, and Olivier Pietquin. **Primal Wasserstein Imitation Learning**, *International Conference on Learning Representations, 2021.* [Dadashi et al., 2021a]. Covered in Chapter 4.
- **Léonard Hussenot***, Marcin Andrychowicz*, Damien Vincent*, Robert Dadashi, Anton Raichuk, Sabela Ramos, Nikola Momchev, Sertan Girgin, Raphaël Marinier, Lukasz Stafiniak, Sertan Girgin, Raphaël Marinier, Nikola Momchev, Sabela Ramos, Manu Orsini, Olivier Bachem, Matthieu Geist, Olivier Pietquin. **Hyperparameter Selection for Imitation Learning**, *In International Conference on Machine Learning, 2021.* [Hussenot et al., 2021a]. Covered in Chapter 5.
- **Léonard Hussenot**, Robert Dadashi, Matthieu Geist, and Olivier Pietquin. **Show me the Way: Intrinsic motivation from Demonstrations**, *International Conference on Autonomous Agents and Multiagent Systems, 2020.* [Hussenot et al., 2021c]. Covered in Chapter 6.
- Robert Dadashi*, **Léonard Hussenot***, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. **Continuous Control with Action Quantization from Demonstrations**, *International Conference on Learning Representations, 2022.* [Dadashi et al., 2022]. Covered in Chapter 7.
- Sabela Ramos, Sertan Girgin, **Léonard Hussenot**, Damien Vincent, Hanna Yakubovich, Daniel Toyama, Anita Gergely, Piotr Stanczyk, Raphaël Marinier, Jeremiah Harmsen, Olivier Pietquin, Nikola Momchev. **RLDS: An Ecosystem to Generate, Share and Use Datasets in Reinforcement Learning**, *arXiv preprint arXiv:2111.02767, 2021.* Ramos et al. [2021]. Partially covered in the background 2.6.
- Matthew W. Hoffman*, Bobak Shahriari*, John Aslanides, Gabriel Barth-Maron, Nikola Momchev*, Danila Sinopalnikov*, Piotr Stańczyk*, Sabela Ramos, Anton Raichuk, Damien Vincent, **Léonard Hussenot***, Robert Dadashi*, Gabriel Dulac-Arnold, Manu Orsini, Alexis Jacq, Johan Ferret, Nino Vieillard, Seyed Kamyar Seyed Ghasemipour, Sertan Girgin, Olivier Pietquin, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Abe Friesen, Ruba Haroun, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Andrew Cowie, Ziyu Wang, Bilal Piot, Nando de Freitas. **Acme: A Research Framework for Distributed Reinforcement Learning**, *arXiv preprint arXiv:2006.00979, 2022.* Hoffman et al. [2020]. Partially covered in the background 2.6.

*Equal Contribution / Core Contributor

These other works are not included in the thesis. They tackle related issues in reinforcement learning like the brittleness of agents against adversarial attacks, how to learn a policy without interacting with the environment, a problem also known as offline reinforcement learning, as well as technical analyses on how to train deep reinforcement learning agents.

- **Léonard Hussenot**, Matthieu Geist, and Olivier Pietquin. **CopyCAT: Taking control of Neural Policies with Constant Attacks**, *International Conference on Autonomous Agents and Multiagent Systems, 2020.* [Hussenot et al., 2020]
- Marcin Andrychowicz, Anton Raichuk, Piotr Stanczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, **Léonard Hussenot**, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, Olivier Bachem. **What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study.** *In International conference on Learning Representations, 2020.* [Andrychowicz et al., 2020a]
- Robert Dadashi, Shideh Rezaeifar, Nino Vieillard, **Léonard Hussenot**, Olivier Pietquin, and Matthieu Geist. **Offline Reinforcement Learning with Pseudometric Learning**, *In International Conference on Machine Learning, pages 2307–2318. PMLR, 2021.* [Dadashi et al., 2021b]
- Shideh Rezaeifar, Robert Dadashi, Nino Vieillard, **Léonard Hussenot**, Olivier Bachem, Olivier Pietquin, and Matthieu Geist. **Offline reinforcement Learning as Anti-Exploration,** *In AAAI Conference on Artificial Intelligence, 2022.* [Rezaeifar et al., 2022]
- Mathieu Blondel, Felipe Llinares-López, **Robert Dadashi**, Léonard Hussenot, Matthieu Geist. **Learning Energy Networks with Generalized Fenchel-Young Losses**, *In Advances in Neural Information Processing Systems, 35, 2022.* [Blondel et al., 2022]

Chapter 2

Background

We provide here the mathematical background and the references that will be needed along the thesis.

2.1 Markov Decision Processes

The reinforcement learning (RL) problem is usually framed as a Markov Decision Process (MDP) [Puterman, 2014, Sutton et al., 1998]. An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \rho_0)$:

- We call $\mathcal{S}$ the state space, which represents the set of possible states of the environment. It is generally numeric values, encoded in a tensor, representing either an image (like the screen of a video game) or the physical state of the system (for example the positions and velocities of the joints of a robot).
- $\mathcal{A}$ is the action space, which is the set of allowed actions at each state. It can be either discrete like $\{up, down, left, right\}$ in a game with these four actions are available, or continuous like $[-1, 1]^8$ for controlling a robot with eight degrees of freedom.
- $\mathcal{P}$ is the transition kernel, where $\mathcal{P}(\cdot|s, a)$ is either a distribution over $\mathcal{S}$ if $\mathcal{S}$ is discrete, or a probability density function if it is continuous, that gives the probability of arriving in a given state if the agent takes action a in state s.
- r is the reward function: $r(s, a)$ gives the average reward received for taking action a in state s.
- $\gamma \in [0, 1[$ is a discount factor appearing in the objective of the MDP.
- ρ_0 is a distribution over $\mathcal{S}$ representing where the agents starts: the initial state distribution.

All along this thesis, the time is considered discrete. A policy π is a mapping from states to distributions over actions. A deterministic policy maps a state to a single action, while a stochastic policy could assign positive weights to various actions in a given state. The space of all policies is denoted Π. We will denote $\pi(a|s)$ the probability π assigns to action a in state s.

One possible formulation of the goal is to learn a policy π^* that maximizes the expected cumulative discounted reward it gets, *i.e.*,

$$\pi^* \in \underset{\pi \in \Pi}{\operatorname{argmax}}\, \mathbb{E}\Big[\sum_{t \geq 0} \gamma^t r(s_t, a_t) \Big| s_0 \sim \rho_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)\Big]. \tag{2.1}$$

We generally call the discounted sum of reward $\sum_{t \geq 0} \gamma^t r(s_t, a_t)$ the *return*. For the rest of the chapter,

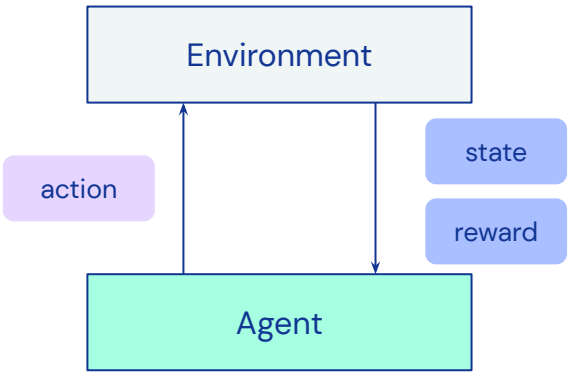


Figure 2.1: The reinforcement learning framework: an *agent* (at the bottom) interacts with an *environment* (at the top). At each time step, the agent takes an *action*. The *environment* provides a *reward* and a new *state*. The goal of the agent is to learn a behavior, that we call a *policy*, that maximizes the overall *cumulative reward* it will get. This means the agent might have to sacrifice immediate reward to maximize its longer-term outcome.

we will consider that $\mathcal{S}$ and $\mathcal{A}$ are discrete for the ease of notation, but all objects can also be defined for to continuous spaces.

2.2 Value Functions & (Approximate) Dynamic Programming

A mathematical object often useful in the context of MDPs is the value function of a policy $V^\pi : \mathcal{S} \to \mathbb{R}$. It represents the expected return the agent gets starting from a given state, a following policy π:

$$V^\pi(s) = \mathbb{E}\big[\sum_{t\geq 0} \gamma^t r_t \big| s_0 = s, a_t \sim \pi(\cdot|s_t), r_t = r(s_t, a_t)\big].$$

Throughout the thesis, we will use $\mathbb{E}_\pi$ to denote the expectation under policy π and the considered MDP. We can then rewrite the previous equation as:

$$V^\pi(s) = \mathbb{E}_\pi\big[\sum_{t\geq 0} \gamma^t r_t \big| s_0 = s\big],$$

Similarly, the Q-function, or action-value function of a policy $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, gives the expected return if the agents starts in state s, takes action a and then follows policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi\big[\sum_{t\geq 0} \gamma^t r(s_t, a_t) \big| s_0 = s, a_0 = a\big].$$

One particular property of value functions is that they follow a recursive equation known as the

Bellman equation [Bellman, 1957]:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\big[\sum_{t\geq 0}\gamma^t r_t \big| s_0 = s\big] \\ &= \mathbb{E}_\pi\big[r(s,a) + \gamma\sum_{t\geq 1}\gamma^t r_t \big| s_0 = s, a\sim\pi(s)\big] \\ &= \sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\mathcal{P}(s'|s,a)\big(r(s,a) + \gamma\mathbb{E}_\pi\big[\sum_{t\geq 0}\gamma^t r_t\big|s_0=s'\big]\big) \\ &= \sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\mathcal{P}(s'|s,a)\big(r(s,a) + \gamma V^\pi(s')\big) \end{aligned}$$

This equation links the value of a state to the value of its successors. It is central to dynamic programming and reinforcement learning as it will allow to *propagate* the value from one state to the others. The same applies to the Q-function:

$$Q^\pi(s,a) = r(s,a) + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}(s'|s,a)\sum_{a'\in\mathcal{A}}\pi(a'|s')Q^\pi(s',a')$$

We define the greedy policy with respect to an arbitrary Q-function as $\pi_{\text{greedy}} = \text{argmax}_{a\in A}\, Q(s,a)$.

A policy π^* is optimal if and only if $\forall\pi\in\Pi, \forall s\in\mathcal{S}, V^{\pi^*}(s)\geq V^\pi(s)$. For the objective considered (2.1), there is always at least one optimal policy. All optimal policies have the same value-function, which is the *optimal value-function*: $V^* = \max_{\pi\in\Pi} V^\pi$. Similarly, their action-value-function is *the optimal action-value-function* and is denoted: $Q^* = \max_{\pi\in\Pi} Q^\pi$. It can be shown that only optimal policies are greedy with respect to their own Q-function.

Dynamic programming [Bellman, 1957] is a class of algorithms that allows to solve equation (2.1) and thus find an optimal policy. Yet several constraints are required: 1) the MDP $\mathcal{M}$ needs to be finite, *i.e.* to have a finite number of states and actions; 2) the number of states needs to be reasonably small because, as you will see, these methods require to store $V^\pi(s)$ for all $s\in\mathcal{S}$; 3) we need perfect knowledge of the environment, *i.e.*, the dynamics $\mathcal{P}$ as well as the reward function r.

One classical dynamic programming algorithm is policy iteration. It consists in repeating the two following steps:

1. evaluate the current policy, *i.e.* compute its value function $V^\pi(s)$ for all $s\in\mathcal{S}$. It can be found using the fact that V^π is the fixed point of a contraction. Using the fixed-point theorem, one can show that iteratively applying this contraction converges to the desired function function.
2. improve current policy by taking the greedy policy with respect to the computed value function.

We refer to Bertsekas [2012] for conditions and proof of convergence of this algorithm.

In various setups, the dynamics of the environment $\mathcal{P}$ and the reward function r are unknown, and can only be discovered step-by-step by interacting with the environment. In this case, classical dynamic programming cannot be readily applied, and some forms of approximation are needed. Q-learning [Watkins and Dayan, 1992] is a classical algorithm to tackle such unknown dynamics. It consists in initializing a Q-function, interacting with the environment with the greedy policy with respect to Q, and, when arriving in state s_{t+1} after taking action a_t in state s_t and receiving r_t, updating Q with the following rule:

$$Q(s_t,a_t)\leftarrow Q(s_t,a_t) + \alpha\big[r_t + \gamma\max_{a\in\mathcal{A}}Q(s_{t+1},a) - Q(s_t,a_t)\big]$$

As one can see, we use the current value of $Q(s_{t+1}, .)$ to define the target update of $Q(s_t, a_t)$. This technique is called *bootstrapping* and is widely used in RL.

This algorithm belongs to the class of *approximate dynamic programming* methods, that allow to solve unknown dynamics problem with a finite, small number of states, that is known as the *tabular* setup. These methods inspired most of RL algorithms. Some real world applications were enabled thanks to RL, like the optimization of data center cooling systems [Lazic et al., 2018] that allowed Google's data centers to reduce their energy consumption by 40% [Evans and Gao, 2021].

2.3 Deep Reinforcement Learning

When facing large and potentially infinite state and action spaces, one cannot store $Q(s, a)$ for all states and actions anymore. Using deep learning [LeCun et al., 2015, Goodfellow et al., 2016] became, over the past years, the canonical way of doing function approximation in RL. First examples of using neural network in combination with RL were applied three decades ago to play Backgammon [Tesauro, 1995, Williams, 1992]. Approximating the Q-function with a neural network, along with techniques to stabilize training, were the main contribution of the seminal work of Mnih et al., demonstrating that we can train agents to play Atari games [Bellemare et al., 2013] with deep RL. Their algorithm, DQN, showed super-human performances on a variety of games, while taking the image as input.

The algorithm is inspired from Q-learning yet requires additional components to be able to learn directly from pixels. In particular, transitions (s_t, a_t, r_t, s_{t+1}) are collected in a *replay buffer* $\mathcal{U}$, which can be seen as the memory of the agent. The Q-network, a neural network with parameters θ, approximates the Q-function. It is trained by updating the parameters θ to minimize:

$$\mathbb{E}_{(s,a,r,s')\sim\mathcal{U}}\left[\left(r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') - Q_\theta(s, a)\right)^2\right]. \tag{2.2}$$

The parameters $\bar{\theta}$ are copied from θ periodically. Having them periodically frozen during the training, rather than constantly equal to θ, helps stabilizing the algorithm. If we do not do so, as the Q-network uses its own prediction to define the target (it uses *bootstrapped* targets as shown in equation (2.2)), the training becomes unstable.

A large number of improvements and tricks were proposed over DQN: changes of the neural network architecture [Van Hasselt et al., 2016], prediction of the distribution of return instead of its expected value [Dabney et al., 2018, Bellemare et al., 2017], or regularization techniques [Vieillard et al., 2020]. Combining these improvements [Hessel et al., 2018] and scaling up the number of parallel actors [Kapturowski et al., 2018] have been key components of recent deep RL successes.

Deep RL agents were trained to play competitive video games like Dota 2 [Berner et al., 2019] or Startcraft II [Vinyals et al., 2019] at super-human levels, beating world champions in competition. They were trained to play Go, a famously hard two-player board game, at which they defeated the world champion Lee Seedol [Silver et al., 2018]. The same algorithm was successfully applied to chess and shogi [Silver et al., 2017], beating state-of-the-art automatic systems. They were used, in robotics, for locomotion [Lillicrap et al., 2016, Mnih et al., 2016, Schulman et al., 2015a, 2017, Haarnoja et al., 2018a,c], manipulation [Levine et al., 2016], grasping [Pinto and Gupta, 2016, Levine et al., 2018] or door opening [Gu et al., 2017]. Deep RL techniques that were developed for robotics [Abdolmaleki et al., 2018] were successfully applied to control tokamak plasma [Degrave et al., 2022], a key step towards developing stable nuclear fusion.

2.4 Reinforcement Learning from Demonstrations

The training of an RL agent starts *tabula rasa*, with no prior knowledge. We say that it is "learning from scratch". In the tabular setting described in section 2.2, Q-values are generally initialized at 0. In the deep RL setting described in section 2.3, initial Q-values are given by the output of a randomly initialized neural network. In both cases, the agent only learns from its own experience, starting from nothing.

Leveraging experience that was previously collected by another agent –be it a human or another artificial agent– has been a central question in several domains like games and robotics. We call this previously collected data: *demonstrations*. Many different assumptions can be made on this data, whether it contains the reward information or not, whether it is optimal or not, and so on. We call this paradigm Reinforcement Learning from Demonstrations (RLfD).

When the demonstrations are deemed optimal (a setup generally called Reinforcement Learning with Expert Demonstrations or RLED), one prototypical idea is to leverage demonstrations by initializing the RL policy with a behavior that is learnt on this data [Schaal, 1997, Peters and Schaal, 2008], through supervised learning. It allows RL training not to start from scratch, but rather to mimic expert data at the start, before maximising the reward function.

Piot et al. proposed to add a margin loss to encourage the Q-values of the expert actions to be higher than the Q-values of the other actions. The idea was successfully applied on both Atari games [Hester et al., 2018] and on robotics tasks [Vecerik et al., 2017]. It notably allowed to solve tasks with very sparse rewards and thus requiring complex exploration behaviors [Paine et al., 2019, Salimans and Chen, 2018, Aytar et al., 2018, Pohlen et al., 2018]. These methods, when used in the deep RL setting, usually rely on additional tricks like filling the replay buffer $\mathcal{U}$ with the expert demonstrations. One other avenue is to use reward shaping [Ng et al., 1999] using the demonstrations [Brys et al., 2015]. In parallel, some works have focused on making the most of imperfect demonstrations [Gao et al., 2018].

2.5 Imitation Learning

Yet, a reward function is not always available. Either because it is expensive to compute or because it is not even possible to design a reward function that properly defines the task at hand [Pan et al., 2022]. Defining a reward that corresponds to "driving well" is, for example, notoriously hard [Abbeel, 2008]. When defining a reward function becomes as hard as hand-writing a behavior, RL is not a solution anymore. In these cases, it is sometimes much easier to simply demonstrate examples of the correct behavior. Whenever it is possible, a solution is thus to define the objective as simply *imitating* the demonstrations, rather than maximizing a reward function. We call this paradigm *Imitation Learning* (IL).

Imitation learning [Schaal, 1999] is typically looked at through two different lenses. On the one hand, behavioral cloning (BC) [Pomerleau, 1991, Bagnell et al., 2007] tries to directly match the demonstrator's behavior, using supervised learning. From a dataset of expert demonstrations $\mathcal{D} = \{(s_t, a_t)\}$, it learns a mapping $f : \mathcal{S} \to \mathcal{A}$ from states to actions. This paradigm, although simple and efficient, presents several limitations. By casting the imitation learning problem in a supervised learning one, the procedure is exposed to *compounding errors*. Indeed, when a mistake is made by the agent at step t, then it is going to end-up in a state s_{t+1} further away from the expert demonstrations. In this new state, the agent will thus be more likely to make a mistake, and thus end up even further from the expert demonstrations. The error accumulates. We call this phenomenon the *distributional shift*. Ross and Bagnell proposed to tackle the problem by recollecting demonstrations along the training. Although efficient, this is not practical as

it requires several human interventions during training. These first approaches neglect the *dynamics* of the environment. They treat separately each state. On the other hand, Inverse Reinforcement Learning [Russell, 1998, Ng et al., 2000] first tries to recover a reward for which the demonstrator's behavior is optimal, before optimizing the reward. This allows to imitate the demonstrator, taking into account the dynamics of the environment. Some of these methods output an explicit reward [Klein et al., 2013, Abbeel and Ng, 2004, Ng et al., 2000, Ziebart et al., 2008] while others, like adversarial imitation learning can be seen as IRL with implicit reward recovery [Ho and Ermon, 2016, Fu et al., 2017, Finn et al., 2016]. This latter class of algorithms are of particular interest for this thesis. Drawing inspiration from IRL [Ng et al., 2000, Ziebart, 2010] and Generative Adversarial Networks [Goodfellow et al., 2014], adversarial imitation learning [Ho and Ermon, 2016] aims at learning a behavior similar to that of the expert given a set of expert demonstrations $\mathcal{D}_{\text{expert}}$ and the ability to interact with the environment.

To do so, the agent with policy π is initialized randomly and interacts with the environment. A discriminator network D is trained to distinguish between samples coming from the agent $(s_t, a_t, s_{t+1}) \sim \mathcal{D}_\pi$ and samples coming from the expert dataset $(s_t, a_t, s_{t+1}) \sim \mathcal{D}_{expert}$ typically with a cross-entropy loss. A reward function for the policy is then defined based on the discriminator prediction, e.g. $r(s, a) = -\ln(1 - D(s, a))$, where $D(s, a)$ denotes the probability of classifying the state-action pair as expert by the discriminator. The agent is then trained with an RL algorithm to maximize this reward and thus fool the discriminator. As in GANs, the training of the discriminator and that of the agent (here playing the role of the *generator*) are interleaved. Therefore, at the high level, the algorithm repeats the following steps in a loop: interact with the environment using the current policy and store the experience in a replay buffer; update the discriminator; perform an RL update accordingly to the RL algorithm used.

Ghasemipour et al. showed that AIL methods aim at matching the state-action distribution of the agent with the distribution of the expert, using different measures of similarity. For instance, GAIL [Ho and Ermon, 2016] considers the Shannon-Jensen divergence while AIRL [Fu et al., 2017] considers the Kullback-Leibler divergence. In the context of GANs, Arjovsky et al. [2017] show that replacing the f-divergence by the Wasserstein distance through the Kantorovich-Rubinstein duality [Villani, 2009] leads to better training stability, which a number of methods in IL have leveraged [Li et al., 2017, Lacotte et al., 2019, Kostrikov et al., 2019, Xiao et al., 2019, Liu et al., 2020, Pacchiano et al., 2020]. However, its implementation comes with a number of practical approximations (*e.g.*, weight clipping to ensure Lipchitz continuity). Although these methods improve on the classical BC baseline, they rely on a min-max optimization problem, that is unstable and hard to optimize. DAC [Kostrikov et al., 2019] demonstrates that, while AIL approaches are based on GAIL, they can still be quite sample efficient. However, they rely on a carefully tuned discriminator (e.g. network architecture, regularization strategy, scheduled learning rates) which requires to interact with the environment.

Numerous successes were accomplished thanks to behavioral cloning, from playing games [Vinyals et al., 2019] to locomotion [Nakanishi et al., 2004, Kalakrishnan et al., 2009] and autonomous driving [Bojarski et al., 2016, Pomerleau, 1988]. Inverse reinforcement learning also unlocked various capabilities like navigation [Ziebart et al., 2008], autonomous helicopter flight [Abbeel and Ng, 2004], or manipulation [Finn et al., 2016].

Overall these methods all assume that the near-optimality of the demonstrations. Some works try to relax this assumption and learn from sub-optimal demonstrations [Jacq et al., 2019, Brown et al., 2019].

2.6 Reproducibility

The recent success of deep learning for computer vision is largely due to the increased reproducibility of experiments. In particular, the usage of common datasets, like ImageNet [Deng et al., 2009] has allowed researchers to compare results to each other in a controlled fashion. Reinforcement learning, while suffering reproducibility issues [Henderson et al., 2018], has taken a similar path by using common environments like Atari games [Bellemare et al., 2013] or Gym Mujoco environments [Brockman et al., 2016b]. Yet very little effort has been made to standardize the full experiment pipeline in RL, IL or RLfD.

The work presented in this dissertation, and especially the large-scale studies that allowed comparing several algorithms on common datasets, would not have been possible without the concurrent development of several tools, that were all open-sourced all along the thesis work.

2.6.1 Acme

We developed Acme [Hoffman et al., 2020], a Reinforcement Learning framework for distributed agents. At its core Acme is a framework designed to enable readable and efficient implementations of reinforcement learning algorithms that target the development of novel RL agents and their applications. One of the hallmarks of modern reinforcement learning is the scale at which such algorithms are run. As a result, algorithms implemented with Acme usually culminate in a *distributed agent* (see Figure 2.2) which involves many separate (parallel) acting, learning, as well as diagnostic and helper processes. However, one of the key design decisions of Acme is to re-use the same components between simple, single-process implementations and large, distributed agents. This gives researchers and algorithm developers the ability to implement RL algorithms once, regardless of how they will later be executed.

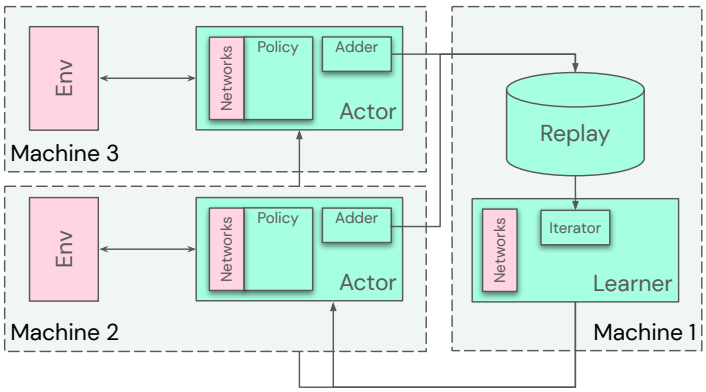


Figure 2.2: This diagram illustrates a distributed training loop. Here two environment loops are running on different machines, feeding data to a single replay, which is co-located with the Learner process on a third machine.

The experiments presented all along this thesis have all been implemented using Acme. Conversely, the work presented in this thesis has also shaped the Acme framework, through design and active agent implementations, for it to support numerous setups including the Imitation Learning and the Learning from Demonstrations ones.

2.6.2 RLDS

In order to increase reproducibility of experiments who requires additional demonstrations, we also developed RLDS [Ramos et al., 2021], a tool that greatly improved dataset management for RL.

A reinforcement learning dataset is made out of the interactions between an *agent* and an *environment*. Agents can be, for example, RL policies, rule-based controllers, formal planners, humans, animals or a combination of these. More precisely, an RL dataset logically represents a –potentially ordered– set of **episodes**, where each episode is a variable-length ordered sequence of interactions (**steps**) of the agent with the environment. We propose a simple data format which was designed to have the following properties:

1. **lossless**: It preserves all information obtained from interacting with an environment without making any assumption on how the data is going to be used or transformed. In particular, it maintains the temporal and episodic information. This ensures that each dataset can be used in conjunction with the widest possible variety of algorithms.
2. **uniform**: It provides a standard way to access the most common fields (e.g. observation, action and reward) returned by reinforcement learning environments and used in sequential decision making algorithms. It also ensures that those have consistent semantics across all datasets. This provides the ability to implement reusable data pipelines to manipulate and transform those fields.
3. **flexible**: It allows users to store custom information at the dataset, episode, or step level (for example, weights of the policy used to generate a given episode, or a visual rendering of the environment). Besides, it does not impose any limitations on the shape and type of the data fields (e.g. observations can include ground-truth features, images or nested dictionaries containing both representations). This ensures that the format can be used with any environment and algorithm, including non-standard ones.

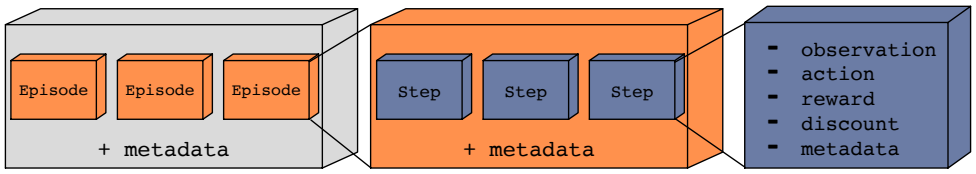


Figure 2.3: RLDS takes advantage of the inherently standard structure of RL datasets and represents them as a dataset of episodes where each of the episodes contains a nested dataset of steps.

RLDS stores sequences of steps where each step contains the fields described below. To select this set of fields, we looked at classical sources [Sutton et al., 1998], common environment suites [Brockman et al., 2016a, Muldal et al., 2019, Zhu et al., 2020], existing datasets [Fu et al., 2020b, Gulcehre et al., 2020] and common RL frameworks [Hoffman et al., 2020, Guadarrama et al., 2018].

- `observation`: the current observation,
- `action`: action applied to the current observation,
- `reward`: reward obtained as a result of applying `action`.
- `discount`: discount obtained together with `reward`. This is generally set to 0 on terminal states.

Each step and episode may contain custom metadata in the form of additional fields. These fields can be used to store environment-related or model-related metadata, *e.g.* hyperparameters. Besides, each step includes a set of flags standing for step properties:

- `is_first`: indicates if this step is the first of the episode.

- `is_last`: indicates that if step is the last of the episode.
- `is_terminal`: indicates if the environment considered this step terminal. A time limit may end a trajectory before reaching a terminal state.

RLDS supports any environments and has allowed reproducible research across a wide variety of them, as depicted in figure 2.4.

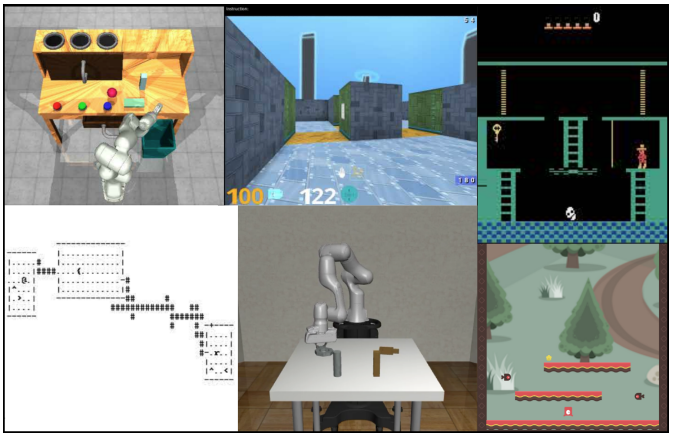


Figure 2.4: Environments for which RLDS allowed datasets to be built and shared: Atari games, DMLab (3D learning environment based on id Software's Quake III Arena), NetHack (single-player text-only roguelike game), Procgen (procedurally-generated 2D games), Robodesk and Robosuite (robot arm). They all have different observation spaces as well as action spaces.

Overall RLDS has allowed us to implement various IL and LfD algorithms working with common datasets in a common format, ensuring high reproducibility of our experiments.

Part I

Imitation Learning: an Empirical Study

Chapter 3

What Matters for Adversarial Imitation Learning?

Contents

Adversarial imitation learning (AIL) has become a popular framework for imitation in continuous control. Over the years, several variations of its components were proposed to enhance the performance of the learned policies as well as the sample complexity of the algorithm. In practice, these choices are rarely tested all together in rigorous empirical studies. It is therefore difficult to discuss and understand what choices, among the high-level algorithmic options as well as low-level implementation details, matter. To tackle this issue, we implement more than 50 of these choices in a generic adversarial imitation learning framework and investigate their impacts in a large-scale study (>500k trained agents) with both synthetic and human-generated demonstrations. While many of our findings confirm common practices, some of them are surprising or even contradict prior work. In particular, our results suggest that artificial demonstrations are not a good proxy for human data and that the very common practice of evaluating imitation algorithms only with synthetic demonstrations may lead to algorithms which perform poorly in the more realistic scenarios with human demonstrations.

3.1 Introduction

Reinforcement learning (RL) has shown its ability to perform complex tasks in contexts where clear reward functions can be set-up (e.g. +1 for winning a chess game) [Silver et al., 2016, Berner et al., 2019,

Andrychowicz et al., 2020b, Vinyals et al., 2019] but for many real-world applications, designing a correct reward function is either tedious or impossible [Popov et al., 2017], while demonstrating a correct behavior is often easy and cheap. Therefore, imitation learning (IL, [Schaal, 1999, Argall et al., 2009]) might be the key to unlock the resolution of more complex tasks, such as autonomous driving, for which reward functions are much harder to design [Feldt, 1998, Sims, 1994, Ecoffet et al., 2021].

The simplest approach to IL is Behavioral Cloning (BC, [Pomerleau, 1991]) which uses supervised learning to predict the expert's action for any given state. However, BC is often unreliable as prediction errors compound in the course of an episode. Adversarial Imitation Learning (AIL, [Ho and Ermon, 2016]) aims to remedy this using inspiration from Generative Adversarial Networks (GANs, [Goodfellow et al., 2014]) and Inverse RL [Russell, 1998, Ng et al., 2000, Ziebart et al., 2008]: the policy is trained to generate trajectories that are indistinguishable from the expert's ones. As in GANs, this is formalized as a two-player game where a discriminator is co-trained to distinguish between the policy and expert trajectories (or states). See Appendix 2.5 for an introduction to AIL.

A myriad of improvements over the original AIL algorithm were proposed over the years [Fu et al., 2017, Kostrikov et al., 2019, Ghasemipour et al., 2020, Blondé et al., 2020, Xu and Denil, 2019], from changing the discriminator's loss function [Fu et al., 2017] to switching from on-policy to off-policy agents [Kostrikov et al., 2019]. However, their relative performance is rarely studied in a controlled setting, and never these changes have never been compared simultaneously. The performance of these high-level choices may also depend on low-level implementation details which might be silenced in the original publications [Islam et al., 2017, Henderson et al., 2018, Tucker et al., 2018, Andrychowicz et al., 2020a], as well as the hyperparameters (HPs) used. Thus, assessing whether the proposed changes are the reason for the presented improvements becomes extremely difficult. This lack of proper comparisons slows down the overall research in imitation learning and the industrial applicability of these methods.

Our contributions. We investigate such high- and low-level choices in depth and study their impact on the algorithm performance. Hence, as our key contributions, we **(1)** implement a highly-configurable generic AIL algorithm, with various axes of variation (over 50 HPs), including 4 different RL algorithms and 7 regularization schemes for the discriminator, **(2)** conduct a large-scale ($>500k$k trained agents) experimental study on 10 continuous-control tasks[1] and **(3)** analyze the experimental results to provide practical insights and recommendations for designing novel and using existing AIL algorithms. We release this generic AIL agent, implemented in JAX [Bradbury et al., 2018] as part of the Acme [Hoffman et al., 2020] framework: `https://github.com/deepmind/acme/blob/master/acme/agents/jax/ail` .

Most surprising finding #1: regularizers. While many of our findings confirm common practices in AIL research, some of them are surprising or even contradict prior work. In particular, we find that standard regularizers from Supervised Learning — dropout [Srivastava et al., 2014] and weight decay [Hanson and Pratt, 1988] often perform similarly to the regularizers designed specifically for adversarial learning like gradient penalty [Gulrajani et al., 2017]. Moreover, for easier environments (which were often the only ones used in prior work), we find that it is possible to achieve excellent results without using any explicit discriminator regularization.

Most surprising finding #2: human demonstrations. Not only does the performance of AIL heavily depend on whether the demonstrations were collected from a human operator or generated by an

[1] A *task* is defined by an environment and the demonstrator type (either human or RL agent).

RL algorithm, but the relative performance of algorithmic choices also depends on the demonstration source. Our results suggest that artificial demonstrations are not a good proxy for human data and that the very common practice of evaluating IL algorithms only with synthetic demonstrations may lead to algorithms which perform poorly in the more realistic scenarios with human demonstrations.

Chapter outline. In Section 3.2, we describe our experimental setting and the performance metrics used. We then present and analyze the results related to the agent (Section 3.3) and discriminator (Section 3.4) training. Afterwards, we compare RL-generated and human-collected demonstrations (Section 3.5) and analyze the choices influencing the computational cost of running the algorithm (Section 3.6). The appendices contain the details of the different algorithmic choices in AIL (Appendix A.1) as well as the raw results of the experiments (Appendix A.4–A.6).

3.2 Experimental design

Environments. We focus on continuous-control tasks as robotics appears as one of the main potential applications of IL and a vast majority of the IL literature thus focuses on it. In particular, we run experiments with five widely used environments from OpenAI Gym [Brockman et al., 2016b]: `HalfCheetah-v2`, `Hopper-v2`, `Walker2d-v2`, `Ant-v2`, and `Humanoid-v2` and three manipulation environments from Adroit [Rajeswaran et al., 2017]: `pen-v0`, `door-v0`, and `hammer-v0`. All the environments are shown in Figure 3.1. The Adroit tasks consist in aligning a pen with a target orientation, opening a door and hammering a nail with a 5-fingered hand.

These two benchmarks bring orthogonal contributions. The former focuses on locomotion but has 5 environments with different state/action dimensionality. The latter, more varied in term of tasks, has an almost constant state-action space.

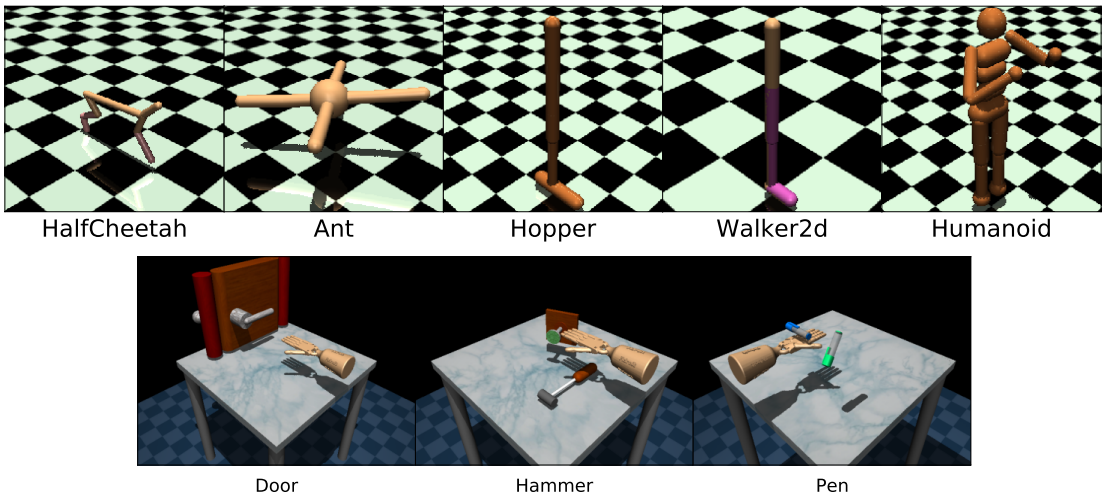


Figure 3.1: Environments: OpenAI Gym (top) and Adroit (bottom).

Demonstrations. For the Gym tasks, we generate demonstrations with a SAC [Haarnoja et al., 2018a] agent trained on the environment reward. For the Adroit environments, we use the "expert" and "human"

datasets from D4RL [Fu et al., 2020a], [2] which are, respectively, generated by an RL agent and collected from a human operator. As far as we know, our work is the first to solve these tasks with human datasets in the imitation setup (most of the prior work concentrated on Offline RL). For all environments, we use 11 demonstration trajectories. Following prior work [Ho and Ermon, 2016, Kostrikov et al., 2019, Ghasemipour et al., 2020], we subsample expert demonstrations by only using every 20^{th} state-action pair to make the tasks harder.

Adversarial Imitation Learning algorithms. We researched prior work on AIL algorithms and made a list of commonly used design decisions like policy objectives or discriminator regularization techniques. We also included a number of natural options which we have not encountered in literature (e.g. dropout [Srivastava et al., 2014] in the discriminator or clipping rewards bigger than a threshold). All choices are listed and explained in Appendix A.1. Then, we implemented a single highly-configurable AIL agent which exposes all these choices as configuration options in the Acme framework [Hoffman et al., 2020] using JAX [Bradbury et al., 2018] for automatic differentiation and Flax [Heek et al., 2020] for neural networks computation. The configuration space is so wide that it covers the whole family of AIL algorithms, in particular, it mostly covers the setups from AIRL[3] [Fu et al., 2017] and DAC [Kostrikov et al., 2019].

Experimental design. We created a large HP sweep (57 HPs swept, >120k agents trained) in which each HP is sampled uniformly at random from a discrete set and independently from the other HPs. We manually ensured that the sampling ranges of all HPs are appropriate and cover the optimal values. Then, we analyzed the results of this initial experiment (called *wide*, detailed description and results in Appendix A.4), removed clearly suboptimal options and ran another experiment with the pruned sampling ranges (called *main*, 43 HPs swept, >250k agents trained, detailed description and results in Appendix A.5). The latter experiment serves as the basis for most of the conclusions drawn in this chapter but we also run a few additional experiments to investigate some additional questions (Appendix A.6 and Appendix A.7).

This pruning of the HP space guarantees that we draw conclusions based on training configurations which are highly competitive (training curves can be found in Figure A.19) while using a large HP sweep (including, for example, multiple different RL algorithms) ensures that our conclusions are robust and valid not only for a single RL algorithm and specific values of HPs, but are more generally applicable. Moreover, many choices may have strong interactions with other related choices, for example we find a surprisingly strong interaction between the discriminator regularization scheme and the discriminator learning rate (Section 3.4). This means that such choices need to be tuned together (as it is the case in our study) and experiments where only a single choice is varied but the interacting choices are kept fixed may lead to misleading conclusions.

Performance measure. For each HP configuration and each of the 10 environment-dataset pairs we train a policy and evaluate it 10 times through the training by running it for 50 episodes and computing the average undiscounted return using the environment reward. We then average these scores to obtain a single performance score which approximates the area under the learning curve. This ensures we assign higher scores to HP configurations that learn quickly.

[2] For `pen`, we only use the "expert" dataset, the "human" one consisting of a single (yet very long) trajectory.

[3] Seminal AIRL uses TRPO [Schulman et al., 2015a] to train the policy, not supported in our implementation (PPO [Schulman et al., 2017] used here).

Analysis. We consider two different analyses for each choice[4]:

Conditional 95th percentile: For each potential value of that choice (e.g., `RL Algorithm` = `PPO`), we look at the performance distribution of sampled configurations with that value. We report the 95th percentile of the performance as well as error bars based on bootstrapping.[5] This corresponds to an estimate of the performance one can expect if all other choices were tuned with random search and a limited budget of roughly 13 HP configurations[6]. All scores are normalized so that 0 corresponds to a random policy and 1 to the expert performance (expert scores can be found in Appendix A.3).

Distribution of choice within top 5% configurations. We further consider for each choice the distribution of values among the top 5% HP configurations. In particular, we measure the ratio of the frequency of the given value in the top 5% of HP configurations with the best performance to the frequency of this value among all HP configurations. If certain values are over-represented in the top models (ratio higher than 1), this indicates that the specific choice is important for good performance.

Robustness of results. While we take 3 random seeds to compute the performance measure for a single choice configuration, it is important to note that all the experimental results reported in this paper are based on more than 3 random seeds: The reported *conditional 95th percentile* and *distribution of choice within top 5% configurations* are computed based upon the performance of hundreds of choice configurations. Hence, the numbers reported in the paper are the results of not only three but hundreds of training runs. Furthermore, we also report confidence intervals for the *conditional 95th percentile*.

We release the raw results of our experiments[7] along with a Notebook allowing to load and study it[8].

3.3 What matters for the agent training?

Summary of key findings. The AIRL reward function perform best for synthetic demonstrations while $-\ln(1-D)$ is better for human demonstrations. Using explicit absorbing state is crucial in environments with variable length episodes. Observation normalization strongly affects the performance. Using an off-policy RL algorithm is necessary for good sample complexity while replaying expert data and pretraining with BC improves the performance only slightly.

Implicit reward function. In this section, we investigate choices related to agent training with AIL, the most salient of which is probably the choice of the implicit reward function. Let $D(s,a)$ be the probability of classifying the given state-action pair as *expert* by the discriminator[9]. In particular, we run experiments with the following reward functions: $r(s,a) = -\log(1-D(s,a))$ (used in the original GAIL paper [Ho and Ermon, 2016]), $r(s,a) = \log D(s,a) - \log(1-D(s,a))$ (called the AIRL reward [Fu et al., 2017]), $r(s,a) = \log D(s,a)$ (a natural choice we have not encountered in literature), and the FAIRL [Ghasemipour et al., 2020] reward function $r(s,a) = -h(s,a)\cdot e^{h(s,a)}$, where $h(s,a)$ is the discriminator logit[10]. It can be shown that, under the assumption that all episodes have the same length, maximizing these reward

[4] This analysis is based on a similar type of study focused on on-policy RL algorithms [Andrychowicz et al., 2020a].

[5] We compute each metric 20 times based on a randomly selected half of all training runs, and then report the mean of these 20 measurements while the error bars show mean-std and mean+std.

[6] The probability that all 13 configurations score worse than the 95th percentile is equal $0.95^{13} \approx 50\%$.

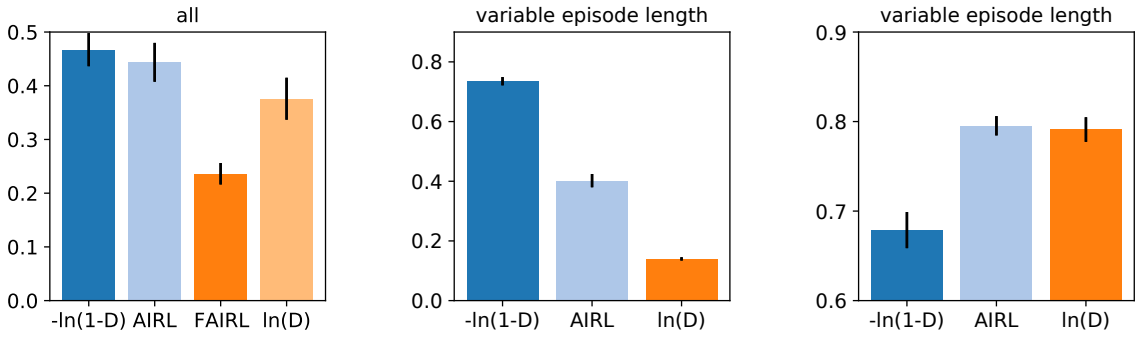
[7] https://storage.googleapis.com/what-matters-in-imitation-learning/data.json

[8] https://storage.googleapis.com/what-matters-in-imitation-learning/analysis_colab.ipynb

[9] Some prior works, including GAIL [Ho and Ermon, 2016], use the opposite notation, with $D(s,a)$ the *non-expert* probability.

[10] It can also be expressed as $h(s,a) = \log D(s,a) - \log(1-D(s,a))$.

functions corresponds to the minimization of different divergences between the marginal state-action distribution of the expert and the policy. See [Ghasemipour et al., 2020] for an in-depth discussion on this topic. We also consider clipping the rewards with absolute values bigger than a threshold which is a HP.



(a) wide HP search (b) no absorbing state (c) absorbing state

Figure 3.2: Comparison of different reward functions. The bars show the 95th percentile across HPs sampling of the *average* policy performance during training. Plot (a) shows the results averaged across all 10 tasks. Plots (b) and (c) show the performance on the subset of environments with variable length episodes when the absorbing state is disabled (b) or enabled (c). See Figure A.7 and Figure A.67 for the individual results in all environments.

The FAIRL reward performed much worse than all others in the initial wide experiment (Figure 3.2a) and therefore was not included in our main experiment. This is mostly caused by its inferior performance with off-policy RL algorithms (Figure A.17). Moreover, reward clipping significantly helps the FAIRL reward (Figure A.18) while it does not help the other reward functions apart from some small gains for $-\ln(1-D)$ (Figure A.77). Therefore, we suspect that the poor performance of the FAIRL reward function may be caused by its exponential term which may have very high magnitudes. Moreover, the FAIRL paper [Ghasemipour et al., 2020] mentions that the FAIRL reward is more sensitive to HPs than other reward functions which could also explain its poor performance in our experiments.

Figure A.20 shows that the $\ln(D)$ reward functions performs a bit worse than the other two reward functions in the main experiment. Five out of the ten tasks used in our experiments have variable length episodes with longer episodes correlated with better behaviour[11] (`Hopper`, `Walker2d`, `Ant`, `Humanoid`, `pen`) — on these tasks we can notice that $r(s,a) = -\ln(1-D(s,a))$ often performs best and $r(s,a) = \ln D(s,a)$ worst. This can be explained by the fact that $-\ln(1-D(s,a)) > 0$ and $\ln D(s,a) < 0$ which means that the former reward encourages longer episodes and the latter one shorter ones [Kostrikov et al., 2019]. Absorbing state (described in Appendix A.1.2) is a technique introduced in the DAC paper [Kostrikov et al., 2019] to mitigate the mentioned bias and encourage the policy to generate episodes of similar length to demonstrations. In Figure 3.2b-c we show how the performance of different reward functions compares in the environments with variable length episodes depending on whether the absorbing state is used. We can notice that without the absorbing state $r(s,a) = -\ln(1-D(s,a)) > 0$ performs much better in the environments with variable episode length which suggests that the learning is driven to a large extent by the reward bias and not actual imitation of the expert behaviour [Kostrikov et al., 2019]. This effect disappears when the absorbing state is enabled (Figure 3.2c).

[11] The episodes are terminated earlier if the simulated robot falls over or if the pen is dropped.

Figure A.67 shows the performance of different reward functions in all environments conditioned on whether the absorbing state is used. If the absorbing state is used, the AIRL reward function performs best in all the environments with RL-generated demonstrations, and $\ln(D)$ performs only marginally worse. The $-\ln(1-D)$ reward function underperforms on the `Humanoid` and `pen` tasks while performing best with human datasets. We provide some hypothesis for this behaviour in Section 3.5, where we discuss human demonstrations in more details.

Observation normalization. We consider observation normalization which is applied to the inputs of all neural networks involved in AIL (policy, critic and discriminator). The normalization aims to transform the observations so that that each observation coordinate has mean 0 and standard deviation 1. In particular, we consider computing the normalization statistics either using only the expert demonstrations so that the normalization is *fixed* throughout the training, or using data from the policy being trained (called *online*). See Appendix A.1.6 for more details. Figure 3.3 shows that input normalization significantly influences the performance with the effects on performance being often much larger than those of algorithmic choices like the reward function or RL algorithm used. Surprisingly, normalizing observations can either significantly improve or diminish performance and whether the fixed or online normalization performs better is also environment dependent.

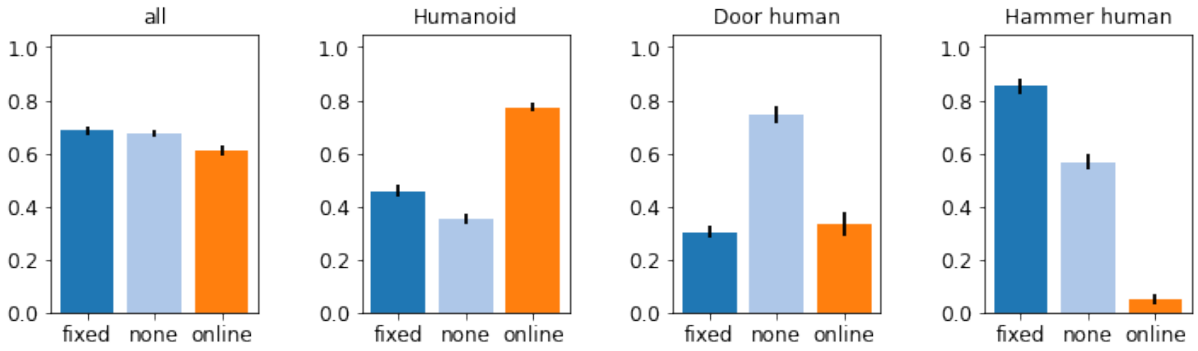


Figure 3.3: Comparison of observation normalization schemes. The bars show the 95th percentile of performance. The leftmost plot shows the results averaged across all 10 tasks. See Figure A.21 for the results on all environments.

Replaying expert data. When demonstrations as well as external rewards are available, it is common for RL algorithms to sample batches for off-policy updates from the demonstrations in addition to the replay buffer [Hester et al., 2018, Paine et al., 2019]. We varied the ratio of the policy to expert data being replayed but found only very minor gains (Figure A.78). Moreover, in the cases when we see some benefits, it is usually best to replay 16–64 times more policy than expert data. On some tasks (`Humanoid`) replaying even a single expert transitions every 256 agent ones significantly hurts performance. We suspect that, in contrast to RL with demonstrations, we see little benefit from replaying expert data in the setup with learned rewards because (1) replaying expert data mostly helps when the reward signal is sparse (not the case for discriminator-based rewards), and (2) discriminator may overfit to the expert demonstrations which could result in incorrectly high rewards being assigned to expert transitions.

Pretraining with BC. We also experiment with pretraining a policy with Behavioral Cloning (BC, [Pomerleau, 1991]) at the beginning of training. Despite starting from a much better policy than a random one, we usually observe that the policy quality deteriorates quickly at the beginning of training (see the `pen` task in Figure 3.6) due to being updated using randomly initialized critic and discriminator networks, and the overall gain from pretraining is very small in most environments (Figure A.22).

RL algorithms. We run experiments with four different RL algorithms, three of which are off-policy algorithms (SAC [Haarnoja et al., 2018a], TD3 [Fujimoto et al., 2018] and D4PG [Barth-Maron et al., 2018]), as well as PPO [Schulman et al., 2017] which is nearly on-policy. Figure 3.4 shows that the sample complexity of PPO is significantly worse than that of the off-policy algorithms while all off-policy algorithms perform overall similarly.

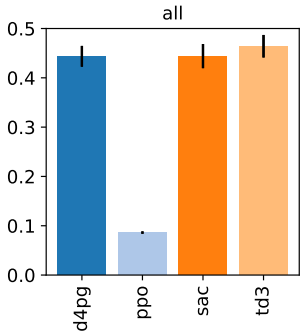


Figure 3.4: Comparison of RL algorithms (wide HP search). See Figure A.2 for the results on individual environments.

RL algorithms HPs. Figure A.3 shows that the discount factor is one of the most important HPs with the values of $0.97 - 0.99$ performing well on all tasks. Figure A.24 shows that in most environments it is better not to erase any data from the RL replay buffer and always sample from all the experience encountered so far. It is common in RL to use a noise-free version of the policy during evaluation and we observe that it indeed improves the performance (Figure A.25). The policy MLP size does not matter much (Figs. A.26-A.27) while bigger critic networks perform significantly better[12] (Figs. A.4-A.5). Regarding activation functions[13], relu performs on par or better than tanh in all environments apart from `door` in which tanh is significantly better (Figure A.28). Our implementation of TD3 optionally applies gradient clipping[14] but it does not affect the performance much (Figure A.29). D4PG can use n-step returns, this improves the performance on the Adroit tasks but hurts on the Gym suite (Figure A.30).

3.4 What matters for the discriminator training?

Summary of key findings. MLP discriminators perform on par or better than AIL-specific architectures. Explicit discriminator regularization is only important in more complicated environments

[12] We thus only include critics with at least two hidden layers with the size at least 128 in the main experiment.
[13] We use the same activation function in the policy and critic networks.
[14] The reason for that is that the DAC paper [Kostrikov et al., 2019] uses TD3 with gradient clipping.

(`Humanoid` and harder ones). Spectral norm is overall the best regularizer but standard regularizers from supervised learning often perform on par. Optimal learning rate for the discriminator may be 2–2.5 orders of magnitude lower than the one for the RL agent.

Discriminator input. In this section we look at the choices related to the discriminator training. Figure A.44 shows how the performance depends on the discriminator input. We can observe that while it is beneficial to feed actions as well as states to the discriminator, the state-only demonstrations perform almost as well. Interestingly, on the `door` task with human data, it is better to ignore the expert actions. We explore the results with human demonstrations in more depth in Section 3.5.

Discriminator architecture. Regarding the discriminator network, our basic architecture is an MLP but we also consider two modifications introduced in AIRL [Fu et al., 2017]: a reward shaping term and a $\log \pi(a|s)$ logit shift which introduces a dependence on the current policy (only applicable to RL algorithms with stochastic policies, which in our case are PPO and SAC). See Appendix A.1.3 for a detailed description of these techniques. Figure A.8 shows that the logit shift significantly hurts the performance. This is mainly due to the fact that it does not work well with SAC which is off-policy (Figure A.16). Figure A.45 shows that the shaping term does not affect the performance much. While the modifications from AIRL does not improve the sample complexity in our experiments, it is worth mentioning that they were introduced for another purpose, namely the recovery of transferable reward functions.

Regarding the size of the discriminator MLP(s), the best results on all tasks are obtained with a single hidden layer (Figure A.46), while the size of the hidden layer is of secondary importance (if it is not very small) with the exception of the tasks with human data where fewer hidden units perform significantly better (Figure A.47). All tested discriminator activation functions perform overall similarly while sigmoid performs best with human demonstrations (Figure A.48).

Discriminator training. Figure A.49 shows that it is best to use as large as possible replay buffers for sampling negative examples (i.e. agent transitions). Prior work has claimed the initialization of the last *policy* layer can significantly influence the performance in RL [Andrychowicz et al., 2020a], thus we tried initializing the last *discriminator* layer with smaller weights but it does not make much difference (Fig A.50).

Discriminator regularization. An overfit or too accurate discriminator can make agent's training challenging, and therefore it is common to use additional regularization techniques when training the AIL discriminator (or GANs in general). We run experiments with a number of regularizers commonly used with AIL, namely Gradient Penalty [Gulrajani et al., 2017] (GP, used e.g. in [Kostrikov et al., 2019]), spectral norm [Miyato et al., 2018] (e.g. in [Blondé et al., 2020]), Mixup [Zhang et al., 2017] (e.g. in [Chen et al., 2021]), as well as using the PUGAIL loss [Xu and Denil, 2019] instead of the standard cross entropy loss to train the discriminator. Apart from the above regularizers, we also run experiments with regularizers commonly used in Supervised Learning, namely dropout [Srivastava et al., 2014], the weight decay [Hanson and Pratt, 1988] variant from AdamW [Loshchilov and Hutter, 2018] as well as the entropy bonus of the discriminator output treated as a Bernoulli distribution. The detailed description of all these regularization techniques can be found in Appendix A.1.5.

Figure 3.5 shows how the performance depends on the regularizer. Spectral normalization performs overall best, while GP, dropout and weight decay all perform on par with each other and only a bit worse

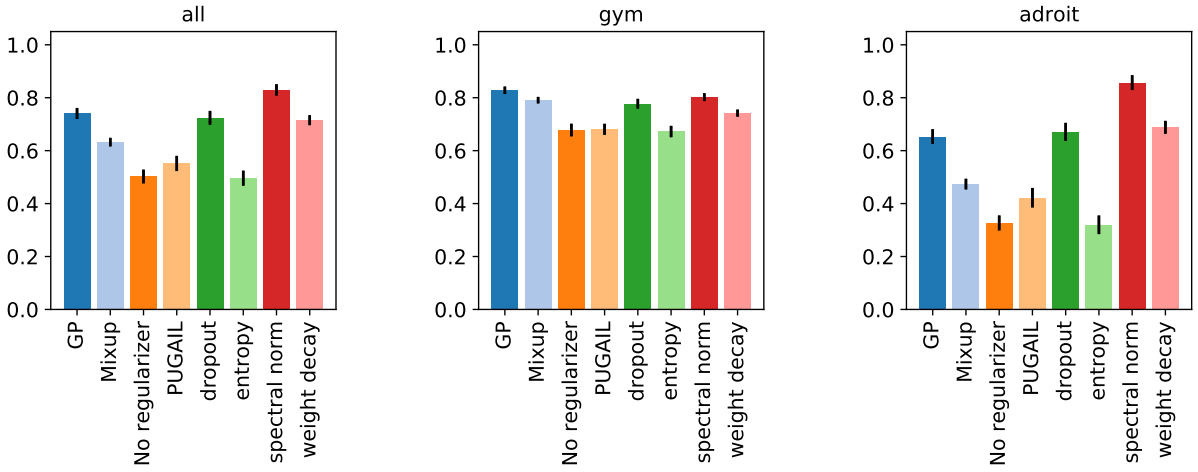


Figure 3.5: The 95th percentile of performance for different discriminator regularizers. The central plot shows the average performance across 5 tasks from OpenAI Gym and the right one the average performance across 5 tasks from the Adroit suite. See Figure A.51 for the plots for individual environments.

than spectral normalization. We find this conclusion to be quite surprising given that we have not seen dropout or weight decay being used with AIL in literature. We also notice that the regularization is generally more important on harder tasks like `Humanoid` or the tasks in the Adroit suite (Figure A.51).

Most of the regularizers investigated in this section have their own HPs and therefore the comparison of different regularizers depends on how these HPs are sampled. As we randomly sample the regularizer-specific HPs in this analysis, our approach favours regularizers that are not too sensitive to their HPs. At the same time, there might be regularizers that are sensitive to their HPs but for which good settings may be easily found. Figure A.62 shows that even if we condition on choosing the optimal HPs for each regularizer, the relative ranking of regularizers does not change.

Moreover, there might be correlations between the regularizer and other HPs, therefore their relative performance may depend on the distribution of all other HPs. In fact, we have found two such surprising correlations. Figure A.68 shows the performance conditioned on the regularizer used *as well* as the discriminator learning rate. We notice that for PUGAIL, entropy and no regularization, the performance significantly increases for lower discriminator learning rates and the best performing discriminator learning rate (10^{-6}) is in fact 2-2.5 orders of magnitude lower than the best learning rate for the RL algorithm (0.0001–0.0003, Figs. A.9, A.33, A.34, A.36, A.42).[15] On the other hand, the remaining regularizers are not too sensitive to the discriminator learning rate. This means that the performance gap between PUGAIL, entropy and no regularization and the other regularizers is to some degree caused by the fact that the former ones are more sensitive to the learning rate and may be smaller than suggested by Figure 3.5 if we adjust for the appropriate choice of the discriminator learning rate. We can notice that PUGAIL and entropy are the only regularizers which only change the discriminator loss but do not affect the internals of the discriminator neural network. Given that they are the only two regularizers benefiting from very low discriminator learning rate, we suspect that it means that a very low learning rate can play a regularizing role in the absence of an explicit regularization inside the network.

[15]The optimal learning rate for those regularizers was the smallest one included in the main experiment. We also run an additional sweep with smaller rates but found that even lower ones do not perform better (Figure A.76).

Another surprising correlation is that in some environments, the regularizer interacts strongly with observation normalization (described Appendix A.1.6) employed on discriminator inputs (see Figure A.69 for an example on `Ant`). These two correlations highlight the difficulty of comparing regularizers, and algorithmic choices more broadly, as their performance significantly depends on the distribution of other HPs.

We also supplement our analysis by comparing the performance of different regularizers for the *best* found HPs. More precisely, we choose the best value for each HP in the main experiment (listed in Appendix A.2) and run them with different regularizers. To account for the mentioned correlations with the discriminator learning rate and observation normalization, we also include these two choices in the HP sweep and choose the best performing variant (as measure by the area under the learning curve) for each regularizer and each environment.

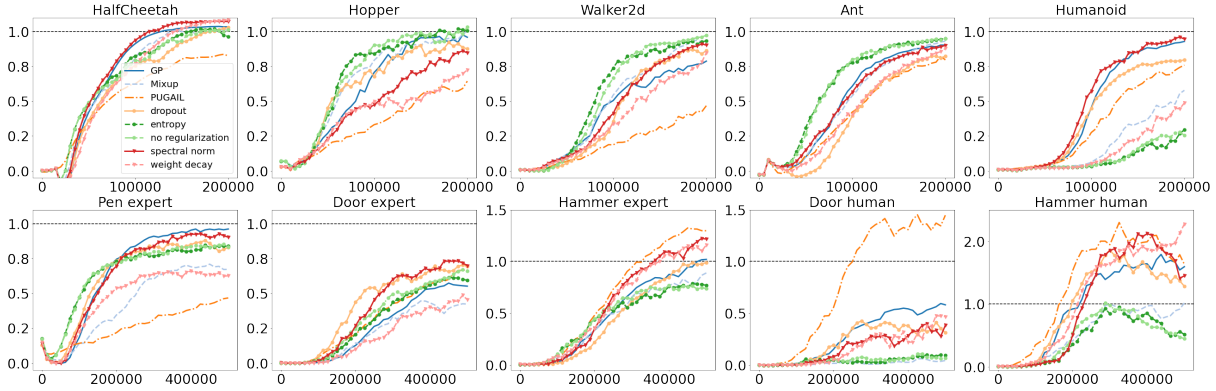


Figure 3.6: Learning curves for different discriminator regularizers when the other HPs are set to the best performing value across all tasks. The y-axis shows the average policy return normalized so that 0 corresponds to a random policy and 1 to the expert. See Appendix A.2 for the HPs used. The plots shows the averages across 30 random seeds. Best seen in color.

While it is not guaranteed that the performance is going to be good at all because we greedily choose the best performing value for each HP and there might be some unaccounted HP correlations, we find that the performance is very competitive (Figure 3.6). Notice that we use the same HPs in *all* environments[16] and the performance can be probably improved by varying some HPs between the environments, or at least between the two environment suites.

We notice that on the four easiest tasks (`HalfCheetah`, `Hopper`, `Walker2d`, `Ant`), investigated discriminator regularizers provide no, or only minor performance improvements and excellent results can be achieved without them. On the tasks where regularization is beneficial, we usually see that there are multiple regularizers performing similarly well, with spectral normalization being one of the best regularizers in all tasks apart from the two tasks with human data where PUGAIL performs better.

Regularizers-specific HPs. For GP, the target gradient norm of 1 is slightly better in most environments but the value of 0 is significantly better in `hammer-human` (Figure A.52), while the penalty strength of 1 performs best overall (Figure A.53). For dropout, it is important to apply it not only to hidden layers but also to inputs (Figure A.54) and the best results are obtained for 50% input dropout and 75%

[16] Apart from the discriminator learning rate and observation normalization used.

hidden activations dropout (Figs. A.54, A.55 and A.62). For weight decay, the optimal decay coefficient in the AIL setup is much larger than the values typically used for Supervised Learning, the value $\lambda = 10$ performs best in our experiments (Figure A.56). For Mixup, $\alpha = 1$ outperforms the other values on almost all tested environments (Figure A.57). For PUGAIL, the unbounded version performs much better on the Adroit suite, while the bounded version is better on the Gym tasks (Figure A.58), and positive class prior of $\eta = 0.7$ performs well on most tasks (Figure A.59). For the discriminator entropy bonus, the values around 0.03 performed best overall (Figure A.60). All experiments with spectral normalization enforce the Lipschitz constant of 1 for each weight matrix.[17]

3.5 Are synthetic demonstrations a good proxy for human demonstrations?

Summary of key findings Human demonstrations significantly differ from synthetic ones. Learning from human demonstrations benefits more from discriminator regularization and may work better with different discriminator inputs and reward functions than RL-generated demonstrations.

Using a dataset of human demonstrations comes with a number of additional challenges. Compared to synthetic demonstrations, the human policy can be multi-modal in that for a given state different decisions might be chosen. A typical example occurs when the human demonstrator remains idle for some time (for example to think about the next action) before taking the actual relevant action: we have two modes in that state, the relevant action has a low probability while the idle action has a very high probability. The human policy might not be exactly markovian either. Those differences are significant enough that the conclusions on synthetic datasets might not hold anymore.

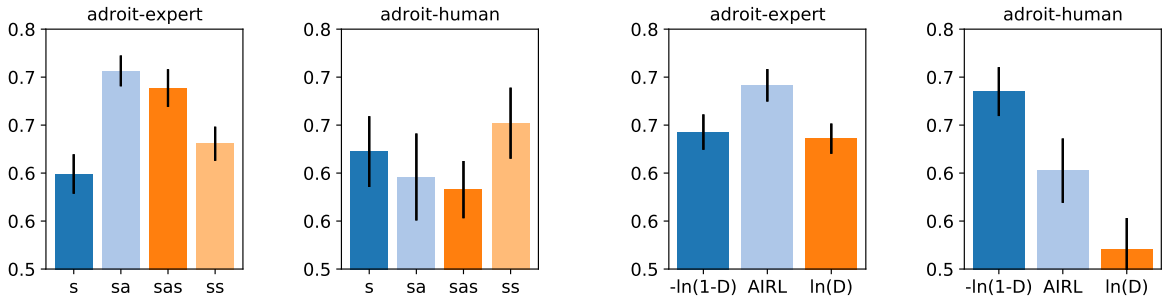
In this section, we focus on the Adroit `door` and `hammer` environments for which we run experiments with human as well as synthetic demonstrations. [18] Note that on top of the aforementioned challenges, the setup with the Adroit environments using human demonstrations exhibits a few additional specifics. The demonstrations were collected letting the human decide when the task is completed: said in a different way, the demonstrator is offered an additional action to jump directly to a terminal state and this action is not available to the agent imitating the expert. The end result is a dataset of demonstrations of variable length while the agent can only generate episodes consisting of exactly 200 transitions. Note that there was no time limit imposed on the demonstrator and some of the demonstrations have a length greater than 200 transitions. Getting to the exact same state distribution as the human expert may be impossible, and imitation learning algorithms may have to make some trade-offs. The additional specificity of that setup is that the reward of the environment is not exactly what the human demonstrator optimized. In the `door` environment, the reward provided by the environment is the highest when the door is *fully* opened while the human might abort the task slightly before getting the highest reward. However, overall, we consider the reward provided by the environment as a reasonable metric to assess the quality of the trained policies. Moreover, in the `hammer` environment, some demonstrations have a low return and we suspect those are not successful demonstrations.[19]

[17] There may be different Lipschitz constants for different networks depending on those of related activations.

[18] For `pen`, we only use the "expert" dataset, the "human" one consists of a single (yet very long) trajectory.

[19] D4RL datasets [Fu et al., 2020a] contain only the policy observations and not the simulator states and therefore it is not straightforward to visualize the demonstrations.

Discriminator regularization. When comparing the results for RL-generated (`adroit-expert`[20]) and human demonstrations (`adroit-human`) we can notice differences on a number of HPs related to the discriminator training. Human demonstrations benefit more from using discriminator regularizers (Figure A.51) and they also work better with smaller discriminator networks (Figure A.47) trained with lower learning rates (Figure A.61). The increased need for regularization suggest that it is easier to overfit to the idiosyncrasies of human demonstrations than to those of RL policies.



(a) discriminator input

(b) reward functions

Figure 3.7: Comparison of discriminator inputs (a) and reward functions (b) for environments with human demonstrations. See Figure A.44 and Figure A.20 for the individual results in all environments.

Discriminator input. Figure 3.7a shows the performance given the discriminator input depending on the demonstration source. For most tasks with RL-generated demonstrations, feeding actions as well as states improves the performance (Figure A.44). Yet, the opposite holds when human demonstrations are used. We suspect that it might be caused by the mentioned issue with demonstrations lengths which forces the policy to repeat a similar movement but with a different speed than the demonstrator.

Reward functions. Finally, we look at how the relative performance of different reward functions depends on the demonstration source. Figure 3.7b shows that for RL-generated demonstrations the best reward function is AIRL while $-\ln(1-D)$ performs better with human demonstrations. Under the assumption that the discriminator is optimal, these two reward functions correspond to the minimization of different divergences between the state (or state-action depending on the discriminator input) occupancy measures of the policy and the expert — See Table 3.1 for the details.

The reward function performing best with human demonstrations ($-\ln(1-D)$) corresponds to the minimization of the Jensen-Shannon divergence (proof in [Ho and Ermon, 2016]).[21] Interestingly, this divergence is symmetric ($D_{\text{JS}}(\pi||E) = D_{\text{JS}}(E||\pi)$) and bounded ($0 \leq D_{\text{JS}}(\pi||E) \leq \ln(2)$). For AIL, the symmetry means that it penalizes the policy for doing things the expert never does with exactly the same weight as for not doing some of the things the expert does while the boundedness means that the penalty for not visiting a single state is always finite. We suspect that this boundedness is beneficial for learning with human demonstrations because it may not be possible to exactly match the human distribution for the reasons explained earlier.

[20] We do not include `pen` in the `adroit-expert` plots so that both `adroit-expert` and `adroit-human` show the results averages across the `door` and `hammer` tasks and differ only in the demonstrations used.

[21] $D_{\text{JS}}(P||Q) = KL(P||M) + KL(Q||M)$, where $M = \frac{P+Q}{2}$.

Table 3.1: Reward functions and corresponding divergences. π and E denote the state occupancy measures of, respectively, the policy and the expert. The proofs can be found in [Ho and Ermon, 2016] and [Ghasemipour et al., 2020]. The *bounded* and *symmetric* column show whether the given *divergence* is bounded or symmetric. D denotes the probability of being classified as *expert*.

Paper	Reward	Divergence	Bounded	Symmetric
GAIL [Ho and Ermon, 2016]	$-\ln(1-D)$	$D_{\mathrm{JS}}(\pi\|\|E)$	✓	✓
AIRL [Fu et al., 2017]	$\ln(D)-\ln(1-D)$	$KL(\pi\|\|E)$	×	×

In contrast to Jensen-Shannon, the $KL(\pi||E)$ divergence which is optimized by the AIRL reward (proof in [Ghasemipour et al., 2020]) is neither symmetric, nor bounded — it penalizes the policy much more heavily for doing the things the expert never does that for not doing all the things the expert does and the penalty for visiting a single state the expert never visits is infinite (assuming a perfect discriminator).

While it is hard to draw any general conclusions only from the two investigated environments for which we had access to human demonstrations, our analysis shows that the differences between synthetic and human-generated demonstrations can influence the relative performance of different algorithmic choices. This suggests that RL-generated data are not a good proxy for human demonstrations and that the very common practice of evaluating IL only with synthetic demonstrations may lead to algorithms which perform poorly in the more realistic scenarios with human demonstrations.

3.6 How to train efficiently?

So far we have analysed how HPs affect the performance of AIL algorithms measured after fixed numbers of environment steps. Here we look at the HPs which influence sample complexity as well as the computational cost of running an algorithm. Raw experiment report can be found in Appendix A.6.

Batch size and replay ratio. One of the main factors influencing the throughput of a particular imitation algorithm is the number of times each transition is replayed on average and the batch size used.[22] See Appendix A.1.1 for the detailed description of the HPs involved. Figure A.71 shows that smaller batches perform overall better (given a fixed replay ratio) and increasing the replay ratio improves the performance, at least up to some threshold depending on the environment (Figure A.72). There is a very strong correlation between the two HPs — Figure A.75 shows that for most batch sizes, the optimal replay ratio is equal to the batch size, which corresponds to replaying exactly one batch of data per environment step. If we compare different batch sizes under the ratio of batches to environment steps fixed to one, the performance is mostly independent of the batch size (Figure A.75).

While in most of our experiment the discriminator and the RL agent are trained with exactly the same number of batches, we also tried doubling the number of discriminator batches. Figure A.73 shows that it improves the performance slightly on the Adroit suite.

[22] We use the same batch size for the policy and actor networks while the discriminator batch size is effectively two times larger because its batches contain always `batch size (C7)` demonstration transitions and `batch size (C7)` policy transitions. The replay ratio is the same for all networks with the exception of the discriminator which can have its replay ratio doubled depending on the value of `discriminator to RL updates ratio (C44)`. See Appendix A.1.4 for details.

Combining multiple batches. We also consider processing multiple batches at once for improved accelerator (GPU or TPU) utilization. In particular, we sample an N-times larger batch from a replay buffer, split it back into N smaller/proper batches on an accelerator, and process them sequentially. In order to keep the replay ratio unaffected, we decrease the frequency of updates accordingly, e.g. instead of performing one gradient update for every environment step, we perform N gradients updates every N environment steps. We apply this technique to the discriminator as well as the RL agent training. The effect on the sample complexity of the algorithm can be seen in Figure A.74. There is a small negative effect for values larger or equal to 16. The effect of this parameter on the throughput of our system could be observed in Figure 3.8. The value of 8 provides a good compromise: almost no noticeable sample complexity regression while decreasing the training time by 2–3 times.

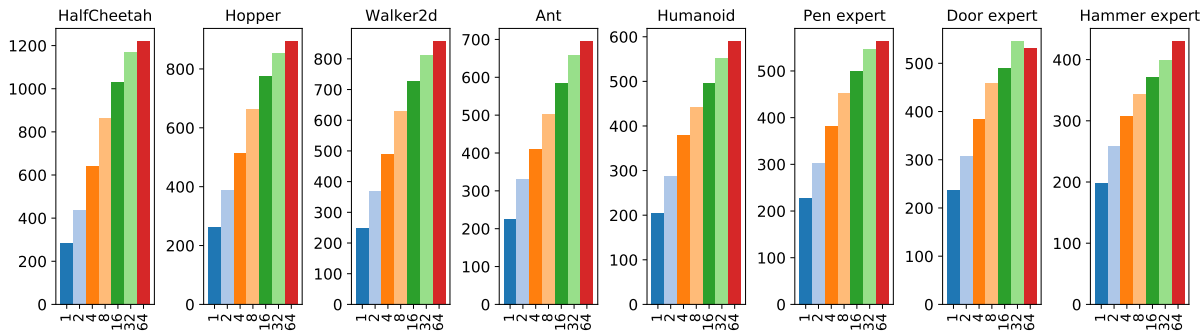


Figure 3.8: Training speed (in terms of environment steps per second) for combining multiple batches. The x-axis denotes the number of batches combined. Other HPs are set to the best performing value across all tasks (listed in Appendix A.2). The plots shows the averages across 10 random seeds.

3.7 Related work

The most similar work to ours is probably [Blondé et al., 2020] which compares the performance of different discriminator regularizers and concludes that gradient penalty is necessary for achieving good performance with off-policy AIL algorithms. In contrast to [Blondé et al., 2020], which uses a single HP configuration, we run large-scale experiments with very wide HP sweeps which allows us to reach more robust conclusions. In particular, we are able to achieve excellent sample complexity on all the environments used in [Blondé et al., 2020]footnote [Blondé et al., 2020] evaluated gradient penalty in the off-policy setup in the following environments: `HalfCheetah`, `Hopper`, `Walker2d` and `Ant`, as well as `InvertedPendulum` which we did not use due to its simplicity. without using any explicit discriminator regularizer (Figure 3.6).

Another empirical study of IL algorithms is [Hussenot et al., 2021a], which investigates the problem of HP selection in IL under the assumption that the reward function is not available for the HP selection.

The methodology of our study is mostly based on [Andrychowicz et al., 2020a] which analyzed the importance of different choices for on-policy actor-critic methods. Our work is also similar to other large-scale studies done in other fields of Deep Learning, e.g. model-based RL [Langlois et al., 2019], Generative Adversarial Networks [Lucic et al., 2018], Natural Language Processing [Kaplan et al., 2020], disentangled representations [Locatello et al., 2018] and convolution network architectures [Radosavovic et al., 2020].

3.8 Discussion

In this chapter, we investigate in depth many aspects of the AIL framework, including discriminator architecture, training and regularization as well as many choices related to the agent training. Our key findings can be divided into three categories: (1) Corroborating prior work, e.g. for the underlying RL problem, off-policy algorithms are more sample efficient than on-policy ones; (2) Adding nuances to previous studies, e.g. while the regularization schemes encouraging Lipschitzness improve the performance, more classical regularizers like dropout or weight decay often perform on par; (3) Raising concerns: we observe a high discrepancy between the results for RL-generated and human demonstrations. We hope this study will be helpful to anyone using or designing AIL algorithms.

Additionally we released the unified AIL agent[23] we implemented in JAX within the Acme framework as well as the raw data[24] of our experiment, along with a Notebook[25] that allows to load and study them.

Studying the various components of the adversarial imitation learning framework highlighted the very large number of hyperparameters this algorithm has. It also underlined the necessity to tweak them as we went from artificially generated demonstrations to human demonstrations. This brittleness calls for new algorithms that would not require solving this min-max optimization problem. In the following chapter, we will see how, by taking the primal formulation of the Wasserstein distance instead of the dual, one can derive a simple and powerful imitation learning algorithm, improving over the state-of-the-art adversarial imitation learning one.

[23] https://github.com/deepmind/acme/tree/master/acme/agents/jax/ail
[24] https://storage.googleapis.com/what-matters-in-imitation-learning/data.json
[25] https://storage.googleapis.com/what-matters-in-imitation-learning/analysis_colab.ipynb

Chapter 4

Beyond Adversarial Imitation Learning

Contents

Imitation Learning methods seek to match the behavior of an agent with that of an expert. In this chapter, we propose a new IL method based on a conceptually simple algorithm: Primal Wasserstein Imitation Learning (PWIL), which ties to the primal form of the Wasserstein distance between the expert and the agent state-action distributions. We present a reward function which has a closed-form, as opposed to recent adversarial IL algorithms that learn a reward function through interactions with the environment. The proposed method is thus efficient and requires little fine-tuning. We show that we can recover expert behavior on a variety of continuous control tasks of the MuJoCo domain in a sample efficient manner in terms of agent interactions and of expert interactions with the environment. Finally, we show that the behavior of the agent we train matches the behavior of the expert with the Wasserstein distance, rather than the commonly used proxy of performance.

4.1 Introduction

Reinforcement Learning (RL) has solved a number of difficult tasks whether in games [Tesauro, 1995, Mnih et al., 2013, Silver et al., 2016] or robotics [Abbeel and Ng, 2004, Andrychowicz et al., 2020b]. However, RL relies on the existence of a reward function, that can be either hard to specify or too sparse to be used in practice. Imitation Learning (IL) is a paradigm that applies to these environments with *hard to specify rewards*: we seek to solve a task by learning a policy from a fixed number of demonstrations generated by an expert.

IL methods can typically be folded into two paradigms: Behavioral Cloning (BC) [Pomerleau, 1991, Bagnell et al., 2007, Ross and Bagnell, 2010] and Inverse Reinforcement Learning (IRL) [Russell, 1998, Ng et al., 2000]. In BC, we seek to recover the expert's behavior by directly learning a policy that *matches* the expert behavior in some sense. In IRL, we assume that the demonstrations come from an agent that acts optimally with respect to an unknown reward function that we seek to recover, to subsequently train an agent on it. Although IRL methods introduce an intermediary problem (i.e. recovering the environment's reward) they are less sensitive to distributional shift [Pomerleau, 1991], they generalize to environments with different dynamics [Piot et al., 2013], and they can recover a near-optimal agent from suboptimal demonstrations [Brown et al., 2019, Jacq et al., 2019].

However, IRL methods are usually based on an iterative process alternating between reward estimation and RL, which might result in poor sample-efficiency. Earlier IRL methods [Ng et al., 2000, Abbeel and Ng, 2004, Ziebart et al., 2008] require multiple calls to a Markov decision process solver [Puterman, 2014], whereas recent adversarial IL approaches [Finn et al., 2016, Ho and Ermon, 2016, Fu et al., 2017] interleave the learning of the reward function with the learning process of the agent. Adversarial IL methods are based on an adversarial training paradigm similar to generative adversarial networks (GANs) [Goodfellow et al., 2014], where the learned reward function can be thought of as the confusion of a discriminator that learns to differentiate expert transitions from non expert ones. These methods are well suited to the IL problem since they implicitly minimize an f-divergence between the state-action distribution of an expert and the state-action distribution of the learning agent [Ghasemipour et al., 2020, Ke et al., 2019]. However the interaction between a generator (the policy) and the discriminator (the reward function) makes it a minmax optimization problem, and therefore comes with practical challenges that might include training instability, sensitivity to hyperparameters and poor sample efficiency.

In this work, we use the Wasserstein distance as a measure between the state-action distributions of the expert and of the agent. Contrary to f-divergences, the Wasserstein distance is a true distance, it is smooth and it is based on the geometry of the metric space it operates on. The Wasserstein distance has gained popularity in GAN approaches [Arjovsky et al., 2017] through its dual formulation which comes with challenges (see Section 4.5). Our approach is novel in the fact that we consider the problem of minimizing the Wasserstein distance through its primal formulation. Crucially, the primal formulation prevents the minmax optimization problem, and requires little fine tuning.

We introduce a reward function computed offline based on an upper bound of the primal form of the Wasserstein distance. As the Wasserstein distance requires a distance between state-action pairs, we show that it can be hand-defined for locomotion tasks, and that it can be learned from pixels for a hand manipulation task. The inferred reward function is non-stationary, like adversarial IL methods, but it is not re-evaluated as the agent interacts with the environment, therefore the reward function we define is computed *offline*. We present a true distance to compare the behavior of the expert and the behavior of the agent, rather than using the common proxy of performance with respect to the true return of the task we consider (as it is unknown in general). Our method recovers expert behaviour comparably to existing

state-of-the-art methods while being based on significantly fewer hyperparameters; it operates even in the extreme low data regime of demonstrations, and is the first method that makes *Humanoid* run with a single (subsampled) demonstration.

4.2 Background and Notations

Markov decision processes. In this chapter, we describe environments as episodic Markov Decision Processes (MDP) with finite time horizon [Sutton and Barto, 2018] $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \rho_0, T)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition kernel, r is the reward function, γ is the discount factor, ρ_0 is the initial state distribution and T is the time horizon. We will denote the dimensionality of $\mathcal{S}$ and $\mathcal{A}$ as $|\mathcal{S}|$ and $|\mathcal{A}|$ respectively. A policy π is a mapping from states to distributions over actions; we denote the space of all policies by Π. In RL, the goal is to learn a policy π^* that maximizes the expected sum of discounted rewards it encounters, that is, the expected return. Depending on the context, we might use the concept of a cost c rather than a reward r [Puterman, 2014], which essentially moves the goal of the policy from maximizing its return to minimizing its cumulative cost.

State action distributions. Suppose a policy π visits the successive states and actions $s_1, a_1, \ldots, s_T, a_T$ during an episode, we define the empirical state-action distribution $\hat{\rho}_\pi$ as:

$$\hat{\rho}_\pi = \frac{1}{T} \sum_{t=1}^{T} \delta_{s_t, a_t},$$

where δ_{s_t, a_t} is a Dirac distribution centered on (s_t, a_t). Similarly, suppose we have a set of expert demonstrations $\mathcal{D} = \{s^e, a^e\}$ of size D, then the associated empirical expert state-action distribution $\hat{\rho}_e$ is defined as:

$$\hat{\rho}_e = \frac{1}{D} \sum_{(s,a) \in \mathcal{D}} \delta_{s,a}.$$

Wasserstein distance. Suppose we have the metric space (M, d) where M is a set and d is a metric on M. Suppose we have μ and ν two distributions on M with finite moments, the p-th order Wasserstein distance [Villani, 2009] is defined as

$$\mathcal{W}_p^p(\mu, \nu) = \inf_{\theta \in \Theta(\mu,\nu)} \int_{M \times M} d(x, y)^p d\theta(x, y)$$

, where $\Theta(\mu, \nu)$ is the set of all couplings between μ and ν. In the remainder, we only consider distributions with finite support. A coupling between two distributions of support cardinal T and D is a doubly stochastic matrix of size $T \times D$. We note Θ the set of all doubly stochastic matrices of size $T \times D$:

$$\Theta = \Big\{ \theta \in \mathbb{R}_+^{T \times D} \mid \forall j \in [1:D], \sum_{i'=1}^{T} \theta[i', j] = \frac{1}{D}, \forall i \in [1:T], \sum_{j'=1}^{D} \theta[i, j'] = \frac{1}{T} \Big\}.$$

The Wasserstein distance between distributions of state-action pairs requires the definition of a metric d in the space $(\mathcal{S}, \mathcal{A})$. Defining a metric in an MDP is non trivial [Ferns et al., 2004, Mahadevan and

Maggioni, 2007]; we show an example where the metric is learned from demonstrations in Section 4.4.4. For now, we assume the existence of a metric $d : (\mathcal{S}, \mathcal{A}) \times (\mathcal{S}, \mathcal{A}) \mapsto \mathbb{R}^+$.

4.3 Primal Wasserstein Imitation Learning

We present the theoretical motivation of our approach: the minimization of the Wasserstein distance between the state-action distributions of the agent and the expert. We introduce a reward based on an upper-bound of the primal form of the Wasserstein distance inferred from a relaxation of the optimal coupling condition, and present the resulting algorithm: Primal Wasserstein Imitation Learning (PWIL).

4.3.1 Wasserstein distance minimization

Central to our approach is the minimization of the Wasserstein distance between the state-action distribution of the policy we seek to train $\hat{\rho}_\pi$ and the state-action distribution of the expert $\hat{\rho}_e$. In other words, we aim at optimizing the following problem:

$$\inf_{\pi \in \Pi} \mathcal{W}_p^p(\hat{\rho}_\pi, \hat{\rho}_e) = \inf_{\pi \in \Pi} \inf_{\theta \in \Theta} \sum_{i=1}^{T} \sum_{j=1}^{D} d((s_i^\pi, a_i^\pi), (s_j^e, a_j^e))^p \theta[i, j]. \tag{4.1}$$

In the rest of the chapter, we only consider the 1-Wasserstein ($p = 1$ in Equation 4.1) and leave the extensive study of the influence of the order p for future work. We can interpret the Wasserstein distance using the earth's movers analogy [Villani, 2009]. Consider that the state-action pairs of the expert are D *holes* of mass D^{-1} and that the state-action pairs of the policy are *piles of dirt* of mass T^{-1}. A coupling θ is a transport strategy between the piles of dirt and the holes, where $\theta[i, j]$ stands for how much of the pile of dirt i should be moved towards the hole j. The optimal coupling is the one that minimizes the distance that the earth mover travels to put all piles of dirt to holes. Note that to compute the optimal coupling, we need knowledge of the locations of all piles of dirt. In the context of RL, this means having access to the full trajectory generated by π. From now on, we write θ_π^* as the optimal coupling for the policy π, that we inject in Equation (4.1):

$$\begin{aligned} \theta_\pi^* &= \operatorname*{argmin}_{\theta \in \Theta} \sum_{i=1}^{T} \sum_{j=1}^{D} d((s_i^\pi, a_i^\pi), (s_j^e, a_j^e)) \theta[i, j] \\ \inf_{\pi \in \Pi} \mathcal{W}_1(\hat{\rho}_\pi, \hat{\rho}_e) &= \inf_{\pi \in \Pi} \sum_{i=1}^{T} c_{i,\pi}^* \text{ with } c_{i,\pi}^* = \sum_{j=1}^{D} d((s_i^\pi, a_i^\pi), (s_j^e, a_j^e)) \theta_\pi^*[i, j]. \end{aligned} \tag{4.2}$$

In Equation (4.2), we have introduced $c_{i,\pi}^*$, which we interpret as a cost to minimize using RL. As $c_{i,\pi}^*$ depends on the optimal coupling θ_π^*, we can only define $c_{i,\pi}^*$ at the very end of an episode. This can be problematic if an agent learns in an online manner or in large time-horizon tasks. Thus, we introduce an upper bound to the Wasserstein distance that yields a cost we can compute online, based on a suboptimal coupling strategy.

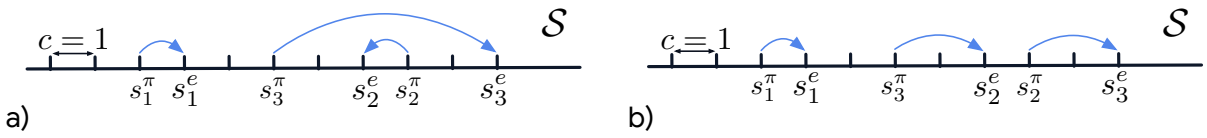


Figure 4.1: Illustration of the difference between the a) greedy coupling and the b) optimal coupling. We present an MDP where we drop the dependency on the action. The state space is $\mathbb{R}$ and the metric associated is the Euclidean distance. We note the states visited by the policy: $s_1^\pi, s_2^\pi, s_3^\pi$ and the states visited by the expert: s_1^e, s_2^e, s_3^e. When the policy encounters the state s_2^π, and because we do not know s_3^π yet, the greedy coupling strategy consists in coupling it with s_2^e although the optimal coupling strategy would be to couple it with s_3^e. Note that the total cost (omitting the constant coupling multiplication factor) with the greedy coupling is 7 whereas the total cost with the optimal coupling is 5 . This highlights that the optimal coupling needs knowledge about the whole policy's trajectory to be derived.

4.3.2 Greedy coupling

In this section we introduce the greedy coupling $\theta_\pi^g \in \Theta$, defined recursively for $1 \leq i \leq T$ as:

$$\theta_\pi^g[i,:] = \underset{\theta[i,:]\in\Theta_i}{\operatorname{argmin}} \sum_{j=1}^{D} d((s_i^\pi, a_i^\pi), (s_j^e, a_j^e))\theta[i,j] \tag{4.3}$$

with:

$$\Theta_i = \Big\{\theta[i,:] \in \mathbb{R}_+^D \;\Big|\; \underbrace{\sum_{j'=1}^{D} \theta[i,j'] = \frac{1}{T}}_{\text{constraint}(a)}, \underbrace{\forall k \in [1:D], \sum_{i'=1}^{i-1} \theta_g[i',k] + \theta[i,k] \leq \frac{1}{D}}_{\text{constraint}(b)}\Big\}.$$

Similarly to Equation (4.1), Equation (4.3) can be interpreted using the earth mover's analogy. Contrary to Equation (4.1) where we assume knowledge of the positions of the T piles of dirts, we now consider that they appear sequentially, and that the earth's mover needs to transport the new pile of dirt to holes *right when it appears*. To do so, we derive the distances to all holes and move the new pile of dirt to the closest remaining available holes, hence the *greedy* nature of it. In Equation (4.3), the constraint (a) means that all the dirt that appears at the i-th timestep needs to be moved, and the constraint (b) means that we cannot fill the holes more than their capacity $\frac{1}{D}$. We show the difference between the greedy coupling and the optimal coupling in Figure 4.1, and provide the pseudo-code to derive it in Algorithm 1.

We can now define an upper bound of the Wasserstein distance using the greedy coupling (since by definition it is suboptimal):

$$\begin{aligned}\inf_{\pi\in\Pi} \mathcal{W}_1(\hat{\rho}_\pi, \hat{\rho}_e) &= \inf_{\pi\in\Pi} \sum_{i=1}^{T}\sum_{j=1}^{D} d((s_i^\pi, a_i^\pi), (s_j^e, a_j^e))\theta_\pi^*[i,j] \\ &\leq \inf_{\pi\in\Pi} \sum_{i=1}^{T}\sum_{j=1}^{D} d((s_i^\pi, a_i^\pi), (s_j^e, a_j^e))\theta_\pi^g[i,j]. \end{aligned} \tag{4.4}$$

In Section 4.4, we empirically validate this bound. We infer a cost from Equation (4.1) at each timestep i:

$$c^g_{i,\pi} = \sum_{j=1}^{D} d((s^\pi_i, a^\pi_i), (s^e_j, a^e_j))\theta^g_\pi[i, j]. \tag{4.5}$$

Note that the greedy coupling $\theta^g_\pi[t, .]$ defined in Equation (4.3) depends on all the previous state-actions visited by the policy π, which makes the cost $c^g_{i,\pi}$ non-stationary, similarly to adversarial IL methods. We can infer a reward from the cost,

$$r_{i,\pi} = f(c^g_{i,\pi})$$

where f is a monotonically decreasing function. This reward function is an episodic history dependent reward function $r(s_0, a_0, \ldots, s_t, a_t)$ where r is uniquely defined from expert demonstrations (hence is said to be derived *offline*). Crucially, we have defined a reward function that we seek to maximize, without introducing an inner minimization problem (which is the case with adversarial IL approaches).

We can now derive an algorithm building on this reward function (Algorithm 1). The algorithm presented is written using pseudo-code for a generic agent A which implements a policy π_A, it observes tuples (s, a, r, s') and possibly updates its components. The runtime of the algorithm for a single reward step computation has complexity $\mathcal{O}((|\mathcal{S}| + |\mathcal{A}|)D + \frac{D^2}{T})$ (see Appendix B.5.2 for details), and therefore has complexity $\mathcal{O}((|\mathcal{S}| + |\mathcal{A}|)DT + D^2)$ for computing rewards across the entire episode. Note that in the case where $T = D$, computing the greedy coupling is simply a lookup of the expert state-action pair that minimizes the distance with the agent state-action pair, followed by a pop-out of this minimizing expert state-action pair from the set of expert demonstrations.

Algorithm 1 introduces an agent with the capability to *observe* and *update* itself, which is directly inspired from Acme's formalism of agents [Hoffman et al., 2020], and is general enough to characterize a wide range of agents. We give an example rundown of the algorithm in Section B.5.1.

Algorithm 1 PWIL: Primal Wasserstein Imitation Learning

Input: Expert demonstrations $\mathcal{D} = \{s^e_j, a^e_j\}_{j\in[1:D]}$, Agent A, number of episodes N
for $k = 1$ **to** N **do**
  Copy expert demonstrations with weight: $\mathcal{D}' := \{s^e_j, a^e_j, w^e_j\}_{j\in[1:D]}$, with $w^e_j = \frac{1}{D}$
  Reset environment, initial state s
  for $i = 1$ **to** T **do**
    Take action $a := \pi_A(s)$, observe next state s'
    Initialize weight $w^\pi := \frac{1}{T}$, cost $c := 0$
    while $w^\pi > 0$ **do**
      Compute $s^e, a^e, w^e := \text{argmin}_{(s^e,a^e,w^e)\in\mathcal{D}'} d((s, a), (s^e, a^e))$
      if $w^\pi \geq w^e$ **then**
        $c := c + w^e d((s, a), (s^e, a^e))$
        $w^\pi := w^\pi - w^e$
        $\mathcal{D}'$.pop(s^e, a^e, w^e)
      else
        $c := c + w^\pi d((s, a), (s^e, a^e))$
        $w^e := w^e - w^\pi$
        $w^\pi := 0$
    $r := f(c)$
    A observes (s, a, r, s'), updates itself

4.4 Experiments

In this section, we present the implementation of PWIL and perform an empirical evaluation, based on the ACME framework [Hoffman et al., 2020]. We test PWIL on MuJoCo locomotion tasks and compare it to the state-of-the-art GAIL-like algorithm DAC [Kostrikov et al., 2019] and against the common BC baseline. As DAC is based on TD3 [Fujimoto et al., 2018] which is a variant of DDPG [Lillicrap et al., 2016], we use a DDPG-based agent for fair comparison: D4PG [Barth-Maron et al., 2018]. We also provide a proof-of-concept experiment on a door opening task where the demonstrations do not include actions and are pixel-based and therefore the metric of the MDP has to be learned. In this setup, we use SAC as the direct RL method. We answer the following questions:

1. Does PWIL recover expert behavior?
2. How sample efficient is PWIL?
3. Does PWIL actually minimize the Wasserstein distance between the distributions of the agent and the expert?
4. Does PWIL extend to visual based observations where the definition of an MDP metric is not straightforward?

We also show the dependence of PWIL on multiple of its components through an ablation study. The complete description of the experiments is given in Appendix B.1. We provide experimental code and videos of the trained agents here: `https://sites.google.com/view/wasserstein-imitation`.

4.4.1 Implementation

We test PWIL following the same evaluation protocol as GAIL and DAC, on six environments from the OpenAI Gym MuJoCo suite [Todorov et al., 2012, Brockman et al., 2016b]: Reacher-v2, Ant-v2, Humanoid-v2, Walker2d-v2, HalfCheetah-v2 and Hopper-v2. For each environment, multiple demonstrations are gathered using a D4PG agent trained on the actual reward of these environments (which was chosen as it is a base agent in the Acme framework). We subsample demonstrations by a factor of 20: we select one out of every 20 transitions with a random offset at the beginning of the episode (expect for Reacher for which we do not subsample since the episode length is 50). The point of subsampling is to simulate a low data regime, so that a pure behavioral cloning approach suffers the behavioral drift and hence does not perform to the level of the expert. We run experiments with multiple number of demonstrations: $\{1, 4, 11\}$ (consistently with the evaluation protocol of DAC).

Central to our method is a metric between state-action pairs. We use the *standardized Euclidean distance* which is the L2 distance on the concatenation of the observation and the action, weighted along each dimension by the inverse standard deviation of the expert demonstrations. From the cost c_i defined in Equation (4.5), we define the following reward:

$$r_i = \alpha \exp(-\frac{\beta T}{\sqrt{|\mathcal{S}| + |\mathcal{A}|}} c_i). \tag{4.6}$$

For all environments, we use $\alpha = 5$ and $\beta = 5$. The scaling factor $T/\sqrt{|\mathcal{S}| + |\mathcal{A}|}$ acts as a normalizer on the dimensionality of the state and action spaces and on the time horizon of the task. We pick $f : x \mapsto \exp(-x)$ as the monotonically decreasing function.

We test our method against a state-of-the-art imitation learning algorithm: DAC and BC. DAC is based on GAIL: it discriminates between expert and non-expert states and it uses an off-policy algorithm

TD3 which is a variant of DDPG instead of the on-policy algorithm TRPO [Schulman et al., 2015a]. We used the open-source code provided by the authors. The Humanoid environment is not studied in the original DAC paper. The hyperparameters the authors provided having poor performance, we led an exhaustive search detailed in Section B.1.2, and present results for the best hyperparameters found.

For our method, we use a D4PG agent (see details in Appendix B.1) and provide additional experiments with different direct RL agents in Appendix B.3. We initialize the replay buffer by filling it with expert state-action pairs with maximum reward α (however, note that we used subsampled trajectories which makes it an approximation to replaying the actual trajectories). We train PWIL and DAC in the limit of 1M environment interactions (2.5M for Humanoid) on 10 seeds. Every 10k environment steps, we perform 10-episode rollouts per seed of the policy without exploration noise and report performance with respect to the environment's original reward in Figure 4.2.

4.4.2 Results

In Figure 4.2, PWIL shows improvement on final performance for Hopper, Ant, HalfCheetah and Humanoid over DAC, similar performance for Walker2d, and is worse on Reacher. On Humanoid, although we ran an exhaustive hyperparameter search (see Section B.1.2) DAC has poor performance, thus exemplifying the brittleness of GAIL-like approaches. On the contrary, PWIL has near-optimal performance, even with a single demonstration.

In terms of sample efficiency, DAC outperforms PWIL on HalfCheetah, Ant and Reacher and has similar performance for Hopper and Walker2d. Note that ablated versions of PWIL are more sample-efficient than DAC on HalfCheetah and Ant (Section B.2). This seems rather contradictory with the claim that GAIL-like methods have poor sample efficiency. However the reward function defined in DAC requires careful tuning (e.g. architecture, optimizer, regularizers, learning rate schedule) which demands experimental cycles and therefore comes with a large number of environment interactions.

Remarkably, PWIL is the first method able to consistently learn policies on Humanoid with an average score over 7000 even with a single demonstration, which means that the agent can actually *run*, (other works consider Humanoid is "solved" for a score of 5000 wich corresponds to *standing*).

Wasserstein distance. In Figure 4.3, we show the Wasserstein distance to expert demonstrations throughout the learning procedure for PWIL and DAC. The Wasserstein distance is evaluated at the end of an episode using a transportation simplex [Bonneel et al., 2011]; we used the implementation from Flamary and Courty [2017]. For both methods, we notice that the distance decreases while learning. PWIL leads to a smaller Wasserstein distance than DAC on all environments but HalfCheetah where it is similar and Reacher where it is larger. This should not come as a surprise since PWIL's objective is to minimize the Wasserstein distance. PWIL defines a reward from an upper bound to the Wasserstein distance between the state-action distributions of the expert and the agent, that we include as well in Figure 4.3. Notice that our upper bound is "empirically tight", which validates the choice of the greedy coupling as an approximation to the optimal coupling. We emphasize that this distance is useful in real settings of IL, regardless of the method used, where we cannot have access to the actual reward of the task.

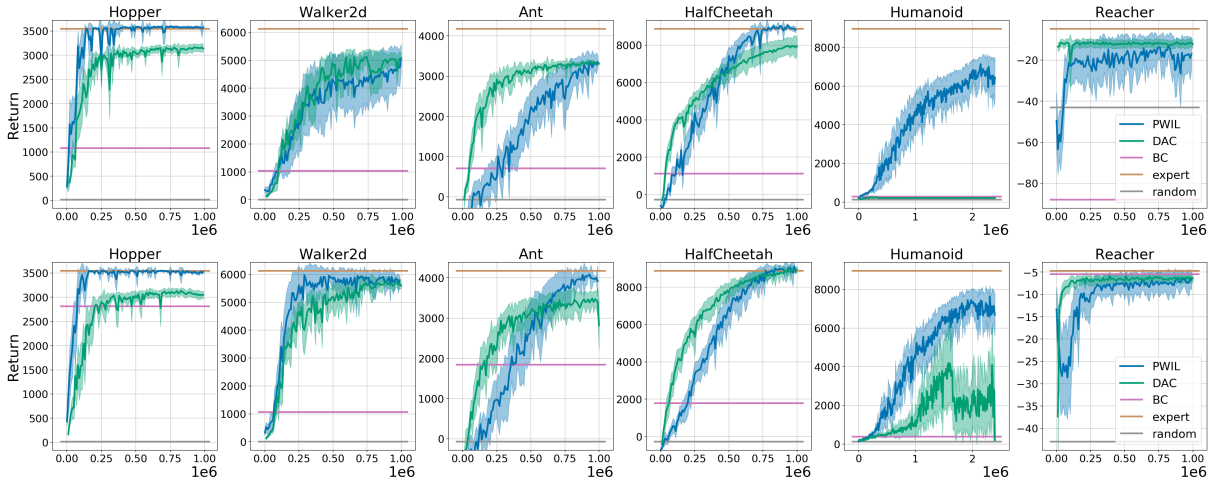


Figure 4.2: Mean and standard deviation return of the evaluation policy over 10 rollouts and 10 seeds, reported every 10k environment steps. The return is in term of the environment's original reward. Top row: 1 demonstration, bottom row: 11 demonstrations.

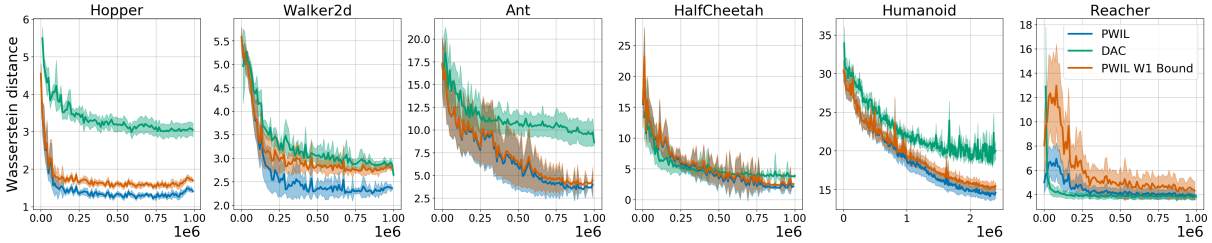


Figure 4.3: Mean of the Wasserstein distance between the state-action distribution of the evaluation policy and the state-action distribution of the expert over 10 rollouts and 10 seeds. Agents were trained with 11 demonstrations. For PWIL, we include the upper bound on the Wasserstein distance based on the greedy coupling defined in Equation (4.4).

4.4.3 Ablation Study

In this section, we present the evaluation performance of PWIL in the presence of ablations and report final performances in Figure 4.4. The learning curves for all ablations can be found in Appendix B.2. We keep the hyperparameters of the original PWIL agent fixed.

PWIL-state. In this version of PWIL, we do not assume that we have access to the actions taken by the expert. Therefore, we try to match the state distribution of the agent with the state distribution of the expert (instead of the state-action distribution). The setup is referred to as Learning from Observation (LfO) [Torabi et al., 2018, Edwards et al., 2019]. The reward is defined similarly to PWIL, using a state distance rather than a state-action distance. Note that in this version, we cannot pre-fill the replay buffer with expert state-action pairs since we do not have access to actions. Remarkably, PWIL-state recovers non-trivial behaviors on all environments.

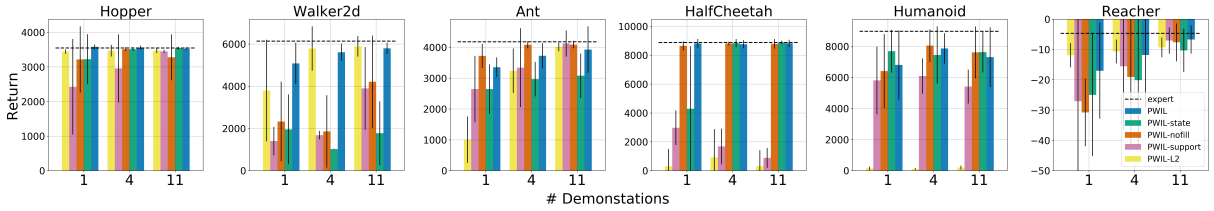


Figure 4.4: Mean and standard deviation of the evaluation performance of PWIL and variants at the 1M environment interactions mark (2.5M for Humanoid). Results are computed over 10 seeds and 10 episodes for each seed.

PWIL-L2. In this version of PWIL, we use the Euclidean distance between state-action pairs, rather than the standardized Euclidean distance. In other words, we do not weight the state-action distance by the inverse standard deviation of the expert demonstrations along each dimension. There is a significant drop in performance for all environments but Hopper-v2. This shows that the performance of PWIL is sensitive to the quality of the MDP metric.

PWIL-nofill. In this version, we do not prefill the replay buffer with expert transitions. This leads to a drop in performance which is significant for Walker and Ant. This should not come as a surprise since a number of IL methods leverage the idea of expert transitions in the replay buffer [Reddy et al., 2020, Hester et al., 2018].

PWIL-support. In this version of PWIL, we define the reward as a function of the following cost:

$$\forall i \in [1:T],\ c_{i,\pi} = \inf_{\substack{\theta_1,\dots,\theta_D \in \mathbb{R} \\ \sum_{j=1}^{D} \theta_j \leq \frac{1}{T}}} \sum_{j=1}^{D} d((s_i^\pi, a_i^\pi), (s_j^e, a_j^e))\theta_j.$$

We can interpret this cost in the context of Section 4.3 as the problem of moving piles of dirt of mass $\frac{1}{T}$ into holes of infinite capacity. It is thus reminiscent of methods that consider IL as a support estimation problem [Wang et al., 2019, Brantley et al., 2020]. This leads to a significant drop in performance for Ant, Walker and HalfCheetah.

4.4.4 Door opening task: Reward from pixel-based observations

In this section, we show that PWIL extends to visual-based setups, where the MDP metric has to be learned. We use the door opening task from Rajeswaran et al. [2018], with *human* demonstrations generated with Haptix [Kumar and Todorov, 2015]. The goal of the task is for the controlled hand to open the door (Figure 4.5) whose location changes at each episode. We add an early termination constraint when the door is opened (hence suppressing the incentive for survival). We assume that we can only access the high-dimensional visual rendering of the demonstrations rather than the internal state (and hence have no access to actions, that is LfO setting).

The PWIL reward is therefore defined using a distance between the visual rendering of the current state and the visual rendering of the demonstrations. This distance is learned offline through self-supervised learning: we construct an embedding of the frames using Temporal Cycle-Consistency Learning (TCC)

[Dwibedi et al., 2019]; TCC builds an embedding by aligning frames of videos of a task coming from multiple examples (in this case the demonstrations). The distance learned is thus the L2 distance between embeddings. The distance is learned on the 25 human demonstrations provided by Rajeswaran et al. [2018] which are then re-used to define the PWIL reward. From the cost c_i defined in Equation (4.5), we define the following reward: $r_i = \alpha \exp(-\beta T c_i)$, with $\alpha = 5, \beta = 1$. We used SAC [Haarnoja et al., 2018a] as the forward RL algorithm, which was more robust than D4PG on the task. We train the agent on 1.5M environment steps and show that we indeed recover expert behaviour and consistently solve the task in Figure 4.6.

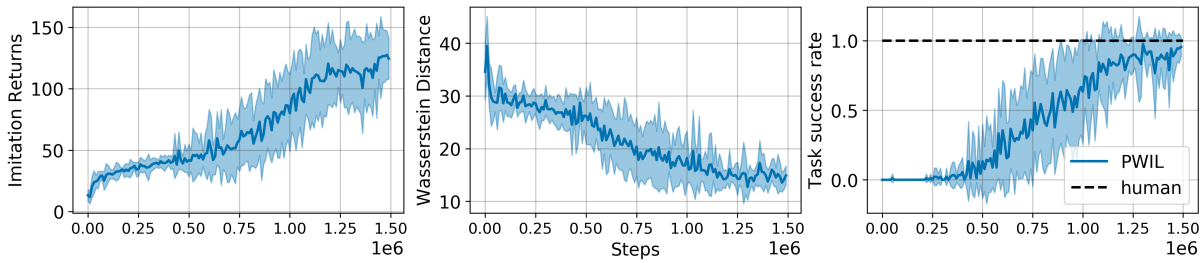


Figure 4.6: We present the average and standard deviation of evaluation performance of PWIL on the door environment. Metrics are computed every 10k environment steps and aggregated over 10 seeds and 10 rollouts. Left: the imitation returns of the agent. Center: Wasserstein distance in the embedding space to expert demonstrations. Right: Success rate of the agent, *i.e.* ratio of episodes where the agent manages to open the door.

4.5 Related Work

We refer to Section 2.5 for a background on imitation learning.

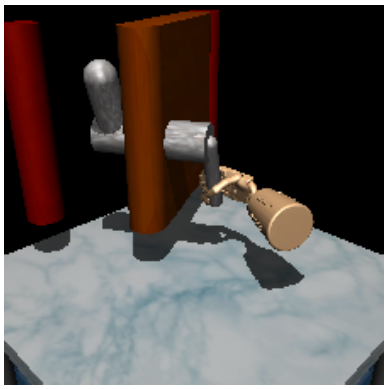


Figure 4.5: Visual rendering of the door opening task.

Expert support estimation. Another line of research in IL consists in estimating the state-action support of the expert and define a reward that encourages the agent to stay on the support of the expert [Piot et al., 2014, Schroecker and Isbell, 2017, Wang et al., 2019, Brantley et al., 2020, Reddy et al., 2020]. Note that the agent might stay on the support of the expert without recovering its state-action distribution. Soft Q Imitation Learning [Reddy et al., 2020] assigns a reward of 1 to all expert transitions and 0 to all transitions the expert encounters; the method learns to recover expert behavior by balancing out the expert transitions with the agent transitions in the replay buffer of a value-based agent. Random Expert Distillation [Wang et al., 2019] estimates the support by using a neural network trained on expert transitions whose target is a fixed random neural network. Disagreement Regularized Imitation Learning [Brantley et al., 2020] estimates the expert support through the variance of an ensemble of BC agents, and use a reward based on the distance to the expert support and the KL divergence with the BC policies. PWIL differs from these methods since it is based on the distance to the

support of the expert, with a support that *shrinks* through "pop-outs" to enforce the agent to visit the whole support. We showed in the ablation study that "pop-outs" are *sine qua non* for recovering expert behaviour (see *PWIL-support*).

Trajectory-based imitation learning. Similarly to the door opening experiment, Aytar et al. [2018] learn an embedding offline and reinforces a reward based on the distance in the embedding space to a single demonstration. However, it is well suited for deterministic environments since they use a single trajectory that the agent aims at reproducing while the door experiment shows that PWIL is able to generalize as the door and handle locations are picked at random. Peng et al. [2018] reinforces a temporal reward function (the reward at time t is based on the distance of the state of the agent to the state of the expert at time t). Contrary to PWIL, the imitation reward function is added to a task-dependent reward function; the distance to expert demonstrations is defined along handpicked dimensions of the observation space; the temporal constraint makes it hard to generalize to environment with stochastic initial states.

Offline reward estimation. Existing approaches derive a reward function without interaction with the environment [Boularias et al., 2011, Klein et al., 2013, 2012, Piot et al., 2016]. They typically require strong assumptions on the structure of the reward (*e.g.* linearity in some predefined features). PWIL also derives a reward offline, without structural assumptions, however it is non-stationary since it depends on the state-action visited in the episode.

MMD-based imitation learning. Generative Moment Matching Imitation Learning [Kim and Park, 2018] assumes a metric on the MDP and aims at minimizing the Maximum Mean Discrepancy (MMD) between the agent and the expert. However, the MMD (like the Wasserstein distance) requires complete rollouts of the agent, and does not present a natural relaxation like the greedy coupling. GMMIL is based on an on-policy agent which makes it significantly less sample-efficient than PWIL. We were not able to implement a version of MMD based on an off-policy agent.

4.6 Discussion

In this chapter, we present Imitation Learning as a distribution matching problem and introduce a reward function which is based on an upper bound of the Wasserstein distance between the state-action distributions of the agent and the expert. A number of imitation learning methods are developed on synthetic tasks, where the evaluation of the imitation learning method can be done with the actual return of the task. We emphasize that in our work, we present a direct measure of similarity between the expert and the agent, that we can thus use in real imitation learning settings, that is in settings where the reward of the task cannot be specified.

The reward function we introduce has a closed-form, and does not need to be learned through interactions with the environment. It requires little tuning, as it only has two hyperparameters, and it can recover near expert performance with as little as a single demonstration on all considered environments, even the challenging Humanoid. Finally our method extends to the harder visual-based setting, based on a metric learned offline from expert demonstrations using self-supervised learning.

All along this chapter, we insisted on measuring the performance of the agent in terms of Wasserstein distance and not only in terms of reward. In imitation learning, the reward information is supposed unknown (and possibly undefined). Nevertheless, most imitation learning methods rely on this unknown

information for a critical part of their algorithms: *hyperparameter selection*. This procedure, necessary in every machine learning algorithm, is key towards performance. In supervised learning, we tackle this procedure by splitting datasets in three subparts: (1) train, the data the algorithm is actually trained on, (2) validation, the data that is used to measure performance and select the best hyperparameters and (3) test, the data that is used to report final performance.

In imitation learning, there is no such procedure. The final goal is the performance on the environment. This performance, depending on the task at hand, can either be to maximise an underlying (and unknown) reward function or to mimic as accurately as possible the demonstrations.

Using the unknown reward function to select hyperparameters breaks the hypotheses of imitation and hinders the practical applicability of these algorithms. In the next chapter, we study the effect of using surrogate metrics, like the Wasserstein distance, to select hyperparameters in imitation learning.

Chapter 5

Hyperparameter Selection for Imitation Learning

Contents

We now address the issue of selecting hyperparameters (HPs) for imitation learning algorithms in the context of continuous-control, when the underlying reward function of the demonstrating expert cannot be observed at any time. The vast literature in imitation learning mostly considers this reward function to be available for HP selection, but this is not a realistic setting. Indeed, would this reward function be available, it could then directly be used for policy training and imitation would not be necessary. To tackle this mostly ignored problem, we propose a number of possible proxies to the external reward. We evaluate them in an extensive empirical study (more than 10'000 agents across 9 environments) and make practical recommendations for selecting HPs. Our results show that while imitation learning algorithms are sensitive to HP choices, it is often possible to select good enough HPs through a proxy to the reward function.

5.1 Introduction

Recent advances in reinforcement learning now allow optimizing control policies with respect to a given reward function even for high-dimensional observation and action spaces [Berner et al., 2019, Vinyals et al., 2019]. However, in many cases it is impossible or impractical to design a reward function which captures the desired outcomes [Popov et al., 2017]. One of the approaches to overcome this issue is imitation learning, which relies on a set of demonstrations presenting the desired behaviour instead of a reward signal [Schaal, 1999, Argall et al., 2009]. Imitation learning can be achieved through pure supervised learning [Pomerleau, 1991] but many imitation learning approaches leverage the assumption that the expert implements an optimal policy according to an *unknown* reward function. This approach, also known as Inverse Reinforcement Learning, tries to recover this unknown reward function and use an RL algorithm to train a policy to maximize it [Russell, 1998, Ng et al., 2000, Ziebart et al., 2008]. Both approaches have their advantages and drawbacks [Piot et al., 2013].

While all machine learning approaches require some degree of hyperparameter tuning, the issue is especially pronounced in RL. Indeed, RL algorithms are known to be very sensitive to the values of their numerous HPs [Henderson et al., 2018, Andrychowicz et al., 2020a]. RL algorithms' HPs are usually chosen by letting different agents interact with the environment and selecting the one which performed best as measured with the environment reward.

Surprisingly, this is also a common practice in the imitation learning domain where one does not have access to an environment reward function which accurately describes the task. If such a reward were known, it could be directly used to train a controller via RL. Although expert trajectories can also be useful in this case, this is not the setting of IL, but of RL with demonstrations where *both* the expert demonstrations and reward signals are used [Kim et al., 2013, Piot et al., 2014, Hester et al., 2018]. This constitutes a gap between the imitation learning framework and the experimental design of imitation learning agents that hinders the practical utility of imitation learning. It is indeed unclear how to tune the imitation agent without access to the reward function.

In some cases, although a per-step reward function is not available, a success signal can be computed or given by a human rater. We focus on the tasks for which (1) such signal is not available or (2) the cost of such signal is prohibitive or (3) such signal could bias the resulting policy. Indeed, just as the reward-engineering problem leads to policies maximizing a reward in an unexpected way [Sims, 1994, Feldt, 1998, Ecoffet et al., 2021], selecting policies on the basis of an incautiously designed metric or of a biased human judgement can lead to biased policies [Henderson et al., 2018].

We present a thorough empirical study of this question in a number of challenging domains with high dimensional spaces of actions and observations. We train thousands of agents, using three imitation learning algorithms based on different paradigms, with large parameter sweeps, and empirically compare different HP selection strategies. In particular, we consider a number of metrics assessing how well the learned behaviors match the demonstrations. Moreover, we investigate how well the algorithms perform if their HPs are selected on a similar task where the reward signal is assumed to be available. As our key contributions, we:

- Highlight the fundamental question of HP selection in imitation learning without access to an external reward signal;
- Provide empirical evidence that the results obtained by IL algorithms depend heavily on how HPs are selected, confirming the prominence of the problem;
- Propose proxy metrics to define the task success;

- Perform a thorough comparison of these alternatives to the external reward signal in a large-scale study;
- Empirically assess the transferability of HPs across tasks for different imitation learning algorithms;
- Give practical recommendations for tuning HPs in imitation learning.

5.2 Hyperparameter Selection

In this chapter, we argue that HP selection should strictly follow the setup in which the algorithm is to be used. For example, when designing an offline RL algorithm, HPs should not be tuned through environment interactions but rather offline [Paine et al., 2020]. Similarly, in IL, one can interact with the environment but should choose HPs without access to the reward signal. We propose to choose these HPs by either (1) using proxy metrics or (2) transferring HPs from other environments that have an accessible reward.

5.2.1 Using proxy metrics

To select a model, one should use a different metric than the return under the supposedly unknown reward. We include a diverse set of metrics that can be used to measure the success of a policy in imitating the demonstrations.

Action MSE. The simplest way to measure similarity between agent behaviour and a given set of demonstrations is to compare actions of the agent and the demonstrator on the states provided in the demonstrations. In particular, for continuous control environments, we use the mean squared error (MSE) between the agent and the expert actions on the training expert states[1]. Notice that this can be done offline without any interaction with the environment. For every environment, we also keep some expert trajectories as a validation set and use it to compute the validation MSE. This will allow us to study if selecting HPs on validation data helps avoiding overfitting, notably as action MSE is actually the loss optimized by the Behavioral Cloning [Pomerleau, 1991] algorithm.

State distribution divergence. Another approach to measuring how well the learned policy imitates the expert is by comparing the distributions of states[2] encountered by both policies. In particular, we compute the Wasserstein distance [Villani, 2009] between the distribution of states in the demonstrations and that of the agent[3] (*i.e.* from generated trajectories). As before, we measure both the distance to the training set and to the validation set and thus have two distinct metrics. The Wasserstein metric computation assumes we can compute distances in the state space, to this end we use the Euclidean distance with state coordinates normalized to have the standard deviation equal to 1. While this is not directly applicable to vision-based observations, there are plenty of techniques which can be used to compute vector embedding for vision observations in RL/IL, e.g. Stooke et al. [2020], Lee et al. [2020].

[1] All action coordinates are rescaled to the $[-1, 1]$ range to make them comparable in magnitude.
[2] We only consider fully observable environments, but similar techniques may work in the partially observable case too.
[3] We approximate the Wasserstein distance using the entropy-regularized Sinkhorn algorithm from the POT library [Flamary and Courty, 2017] (leading to faster and more stable values) with a regularization parameter of 5.

Random Network Distillation (RND). As proposed by Burda et al. [2018], Wang et al. [2019], we define a support estimation metric by taking two randomly initialized networks, and train one to predict the *random* output of the other one on the expert training set. The metric is then given by the MSE between the prediction network and the frozen one on trajectories generated by the agent. Details on the metric training are given in Appx. C.1.4.

Imitation Return. Inverse RL algorithms recover a –learned– reward function. We can use the corresponding learned return to select HPs. We compute this metric as it should model the goal optimized by the agent, although this goal is generally non-stationary. Moreover, the scale of the metric often depends on the HPs which could make it harder to compare the value of this metric between different training runs.

Environment return. This is the sum of rewards obtained during a full episode according to the environment reward function. We include it as the gold standard.

A desired property for all these metrics is to preserve the ranking of policies induced by the oracle environment return. A metric that would satisfy this property would yield an optimal HP selection. Note that this is a sufficient but not a necessary condition. We show in Appx C.2.1, using both the Spearman rank correlation and the ROC-AUC of an additional task (classify good from poor policies), that the aforementioned metrics have good enough ranking properties to make them legitimate candidates for HP selection.

5.2.2 Using transfer

The HPs that work best in tasks for which a well-defined reward function is available can also be transferred to a new task. We expect the success of this procedure to highly depend on the similarity between the source and target domains.

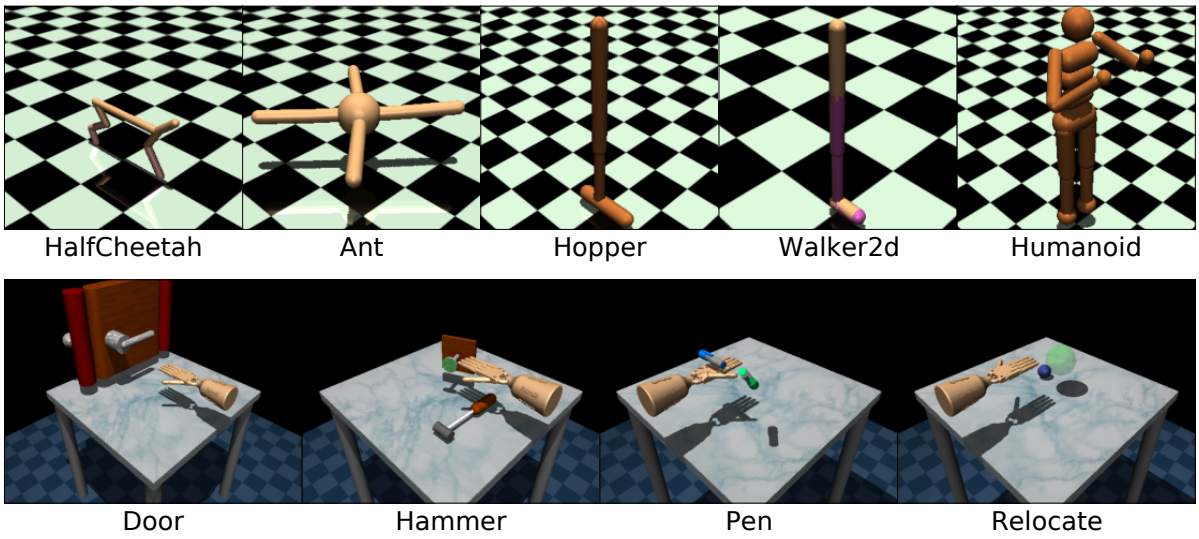


Figure 5.1: Environments: OpenAI Gym (top) and Adroit (bottom).

5.2.3 Environments and data

We focus on continuous-control benchmarks and consider five widely used environments from OpenAI Gym [Brockman et al., 2016a]: `Hopper-v2`, `Walker2d-v2`, `HalfCheetah-v2`, `Ant-v2`, and `Humanoid-v2` and four manipulation tasks from Adroit [Kumar, 2016]: `pen-v0`, `relocate-v0`, `door-v0`, and `hammer-v0`. The Adroit tasks respectively consist in aligning a pen with a target orientation, moving an object to a target location, opening a door and hammering a nail. These two benchmarks bring orthogonal contributions. The former focuses on locomotion but has 5 environments with different state/action dimensionality. The latter, more varied in term of tasks, has an almost constant state-action space. For the Gym tasks, we generate demonstrations with a Soft Actor-Critic (SAC) [Haarnoja et al., 2018b] agent trained on the environment reward. For the Adroit environments, we use the "expert" datasets from the D4RL dataset [Fu et al., 2020b].

For the OpenAI Gym environments, we use 11 training trajectories and keep 5 additional held-out trajectories for validation. For the Adroit environments, 20 training trajectories are used as well as 5 validation trajectories. The number of training trajectories corresponds to what IL algorithms are typically designed for. We chose a low number of validation trajectories as demonstrations are generally expensive and one wishes to use as many as possible for training.

5.2.4 Algorithms

We consider the three algorithms briefly described below. They use different approaches, from supervised learning, to Inverse RL via distribution matching, either through the primal or the dual form of a divergence. We consider them to be representative of different families of algorithms and thus good candidates to validate HP selection techniques.

Detailed description of the algorithms can be found in the original publications. We provide additional information on our implementations in Appx. C.1. We implemented our algorithms in the Acme framework Hoffman et al. [2020] using JAX Bradbury et al. [2018] for automatic differentiation and Flax Heek et al. [2020] for neural networks computation.

Behavioral Cloning (BC, Pomerleau [1991]) is the simplest approach to IL and relies on mapping the expert states to the expert actions in a supervised manner. In contrast to the other algorithms we consider, BC is an offline algorithm as it does not require any interactions with the environment.

Adversarial Imitation Learning (AIL, Ho and Ermon [2016]) is a family of algorithms stemming from the seminal GAIL paper [Ho and Ermon, 2016]. The overall design uses a classifier to discriminate the expert state-action pairs from the agent ones, and an RL algorithm trains the policy to maximize the confusion of this discriminator. Our implementation is mostly similar to Discriminator-Actor-Critic (DAC, Kostrikov et al. [2019]) but uses different reward functions depending on the HPs. See Appx. C.1.2 for details.

Primal Wasserstein Imitation Learning (PWIL, Dadashi et al. [2021a], Chapter 4) uses an RL algorithm to minimize a greedy upper bound on the Wasserstein distance between the state-action distributions of the expert and the agent. For both AIL and PWIL, we use Soft Actor-Critic (SAC, Haarnoja et al. [2018b]) to train the policy.

5.2.5 Experimental design

For each algorithms and 5 different random seeds[4], we sample 100 independent HP configurations from the HP sweeps detailed in Appx. C.1. We then use them to train a total of 500 agents per algorithm-environment pair. Online algorithms (AIL & PWIL) are run for 1M environment steps while BC is trained for 60k gradient steps. Each agent is evaluated 20 times throughout its training. At each evaluation, all the metrics are computed using 50 episodes[5].

We want to check whether the metrics introduced in Sec. 5.2 can be used for the problem of HP selection. To this end, we repeat the following experiment 20 times for each of the 5 seeds: (1) we uniformly sample 25 HP configurations from the set of all 100 configurations, (2) we take the final policies from the corresponding training runs, (3) we select the best one according to the techniques outlined in Sec. 5.2 and (4) we check how well it performs according to the environment reward This allows to simulate 5×20 practitioners that would run sweeps of size 25 and look for the best configuration possible.

We also investigate how the metrics described in Sec. 5.2 can be used for early stopping. To this end, we repeat the above experiment but this time we select the best policy not only from the fully trained ones, but also include the partially trained policies from the corresponding training runs.

Furthermore, we evaluate whether the performance on another task with a well-defined reward function can be used to select HPs. To this end, we choose a set of validation environments and repeat the experiment mentioned above but this time selecting the HP configuration with the best average normalized[6] return across the validation tasks. We then check how well it performs on the test environment.

5.3 Experimental Results

In this section, we try to answer the following questions:

1. Can we effectively choose an imitation learning algorithm and its HPs without access to the true reward function?
2. Which of the proxy metrics defined in Sec. 5.2 is the best?
3. Is early stopping important in IL algorithms?
4. Do HPs transfer well between different environments?

5.3.1 Hyperparameter selection via proxy metrics

Fig. 5.2 and Fig. 5.3 show how the performance of different algorithms varies as we change the metric used for HP selection on OpenAI Gym and Adroit environments respectively.

By looking at the first bar in each subplot (performance for HPs selected on the episode return), we can see that all algorithms, including BC[7], can achieve returns comparable to or even better than those of the expert on almost all environments if the HP selection and early stopping are performed using the

[4] The random seed fixes the episodes train/validation split.

[5] We use only 10 episodes for the state divergence as it is more computationally demanding.

[6] Rewards are normalized per task so that 0 corresponds to a random policy and 1 is the average return in the demonstrations.

[7] Many recent publications in IL (e.g. Ho and Ermon [2016]) *subsample* demonstrations by including only every n-th state-action pair to make the task harder for BC. We did not follow this practice as it has little justification from the practical point of view.

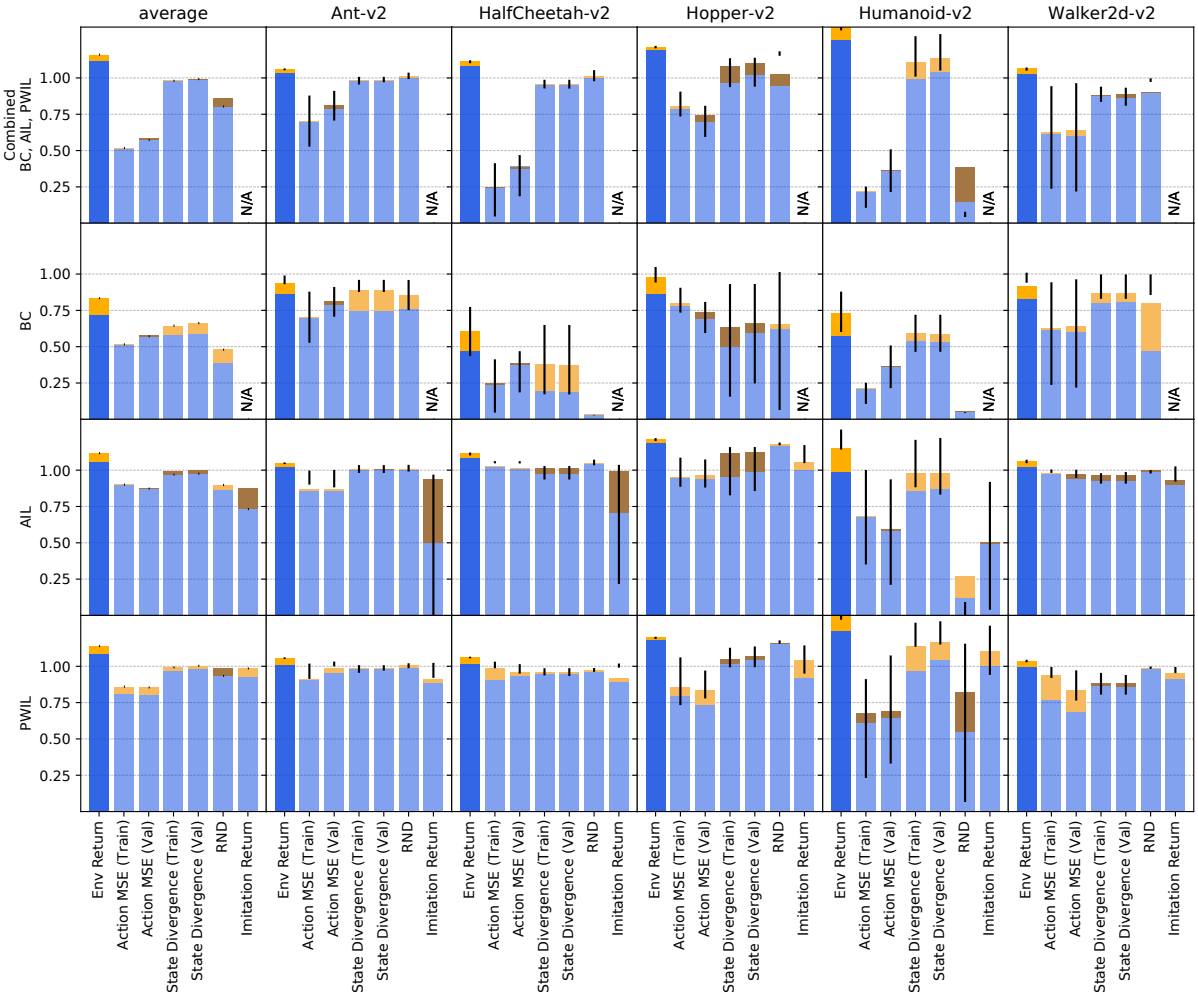


Figure 5.2: The episode return achieved by different algorithms if the HP selection and early stopping is performed using a proxy metric for OpenAI Gym tasks. Each subplot corresponds to a different algorithm and a different environment with the first column showing the results averaged across environments. The first row corresponds to the case when we choose HPs as well as the IL algorithm used based on the given metric. Episode returns are rescaled for each environment so that 0 corresponds to a random policy and 1 to the average episode return in the demonstration set. The lower (blue) part of each bar shows the episode returns in the case of no early stopping and the full bar (blue and yellow) shows the performance when using early stopping with the same metric as for the HP selection. The vertical lines show the 25-th and 75-th percentile of the episode return across reruning the whole HP selection process as described in Sec. 5.2.5 for the early stopping case. Brown color in the upper part of the bar means that the algorithm performs better without early stopping and shows how much performance is lost by using it. Each bar shows the mean performance across running the HP selection process 100 times.

oracle episode return. In some cases, the policies obtained this way can even perform significantly better than the expert (e.g. BC on `pen-expert`, Fig. 5.3). This should not happen as all the algorithms are trying to mimic the expert behaviour and we argue that this is an artifact of the policy selection process.

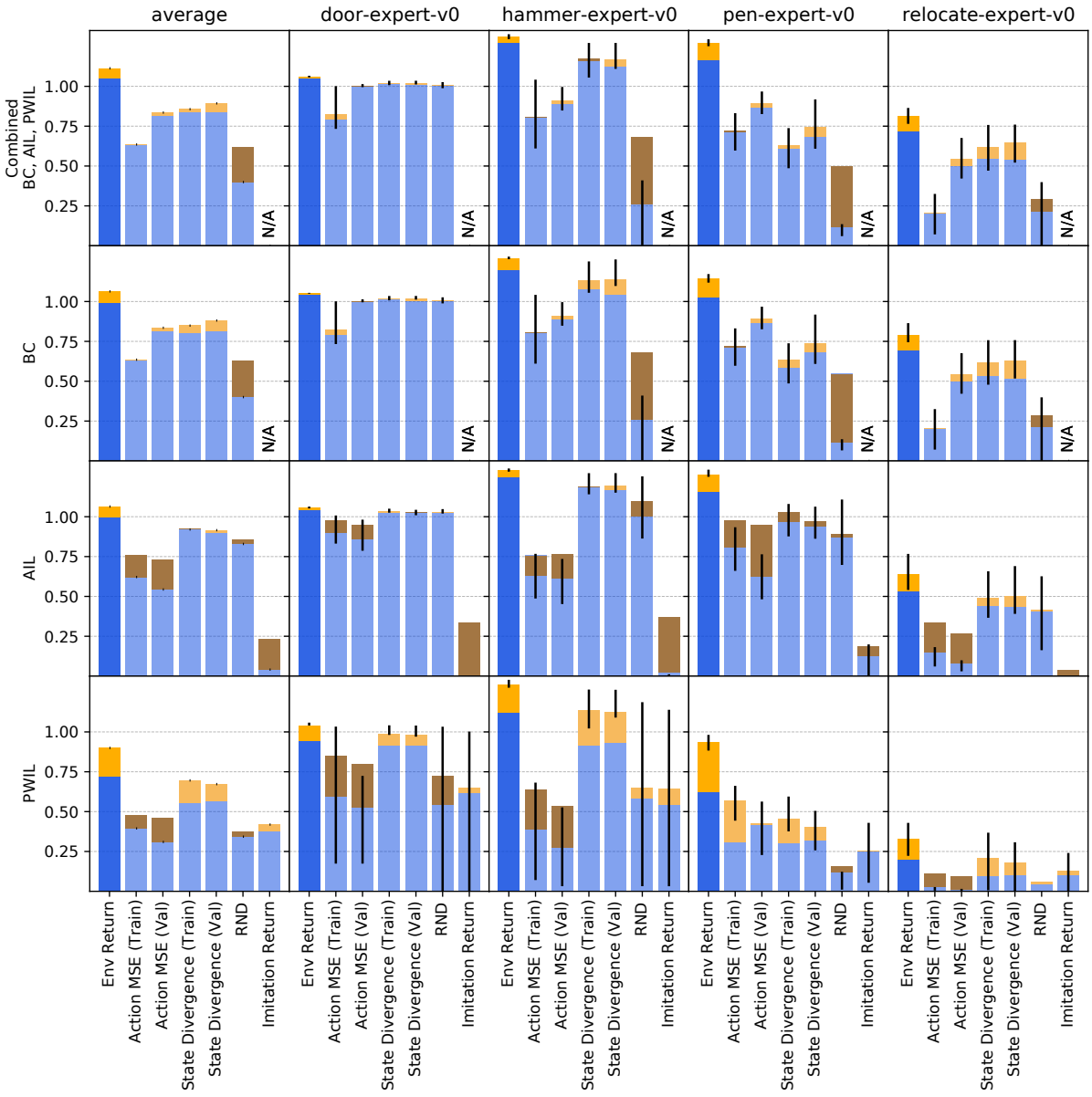


Figure 5.3: The episode return achieved by different algorithms if the HP selection and early stopping is performed using a proxy metric for Adroit environments. See the caption for Fig. 5.2 for the detailed description of this plot.

How much do we lose by choosing an IL algorithm and HPs using a proxy metric? The top-left subplot in Fig. 5.2 and Fig. 5.3 shows the performance averaged across the tasks in a given suite when the HPs *as well as* the IL algorithm are chosen using the proxy metrics. The selection by state divergence achieves episode returns similar to that of the demonstrator on both environment suites. This is a *very positive result*, suggesting that it is, in practice, possible to select HPs without access to a reward

function and still obtain a policy that performs well at the task.

On the other hand, the big gap between the performance of all three algorithms on Adroit (first column in Fig. 5.3) with HPs selected on the return *vs.* proxy metrics shows that the research practice of selecting HPs based on the return can lead to significant overestimation of an algorithm's performance when no reward function is available. This may limit the applicability of some IL algorithms to practical problems.

Which proxy metric is the best? The state divergence performs best for all algorithms. Action MSE performs worse than the state divergence but still achieves at least 75% of the expert score on some algorithm-environment pairs. The inferior performance of action MSE is expected as it is a fully offline metric unaware of the system dynamics. Whether the metrics are computed against the set of training demonstrations or a held-out set of demonstrations makes little difference in our experiments and suggests that it might be better to use all available demonstrations for training[8].

The metric given by the average RND score on the episode performs well to select HPs on OpenAI Gym environments but is outperformed by the state divergence. Perhaps suprisingly, the metric is not as performant on Adroit. This suggests that this support-estimation metric might have to be adapted for environments with more stochasticity in initial states. Imitation return (the sum of learned-reward collected in an episode), as defined by in AIL and PWIL also perform poorly and is usually worse than action MSE with the exception of PWIL on OpenAI Gym where both approaches perform similarly. The inferior performance of the imitation return for AIL can be explained by the fact that the reward function introduced in the algorithm is non-stationary. Therefore comparing the values from different training runs or different timesteps might be misleading. Concerning PWIL, we suspect that its inferior performance on Adroit compared to the state divergence metrics is caused by the fact that the upper-bound on the Wasserstein distance introduced by the algorithm is not tight.

Is early stopping important? The upper yellow (resp. brown) parts of the bars in Fig. 5.2 and Fig. 5.3 show how much is gained (resp. lost) by using early stopping based on the same metric as for the HP selection. We can see that early stopping almost always improves performances if a reliable metric (*i.e.*, state divergence) is used and the task was not already almost completely solved without it. The magnitude of the gain depends heavily on the environment and the algorithm, in particular we found early stopping to be particularly helpful for PWIL on Adroit environments.

Can we use metrics to choose the algorithm? The first row in Fig. 5.2 and Fig. 5.3 show the performance when we use the metrics to choose not only HPs but also the IL algorithm. We observe that when using the state divergence as the metric to select the algorithm, we achieve comparable performance to that of the best algorithm. Furthermore, selecting the algorithm based on action MSE results in similar performance as BC even if it is not the best algorithm. This may be due to the fact that BC optimizes the very same criterion, hence it is more likely to have the best results according to that metric. We provide additional evidence for this hypothesis in Appx. C.2.2.

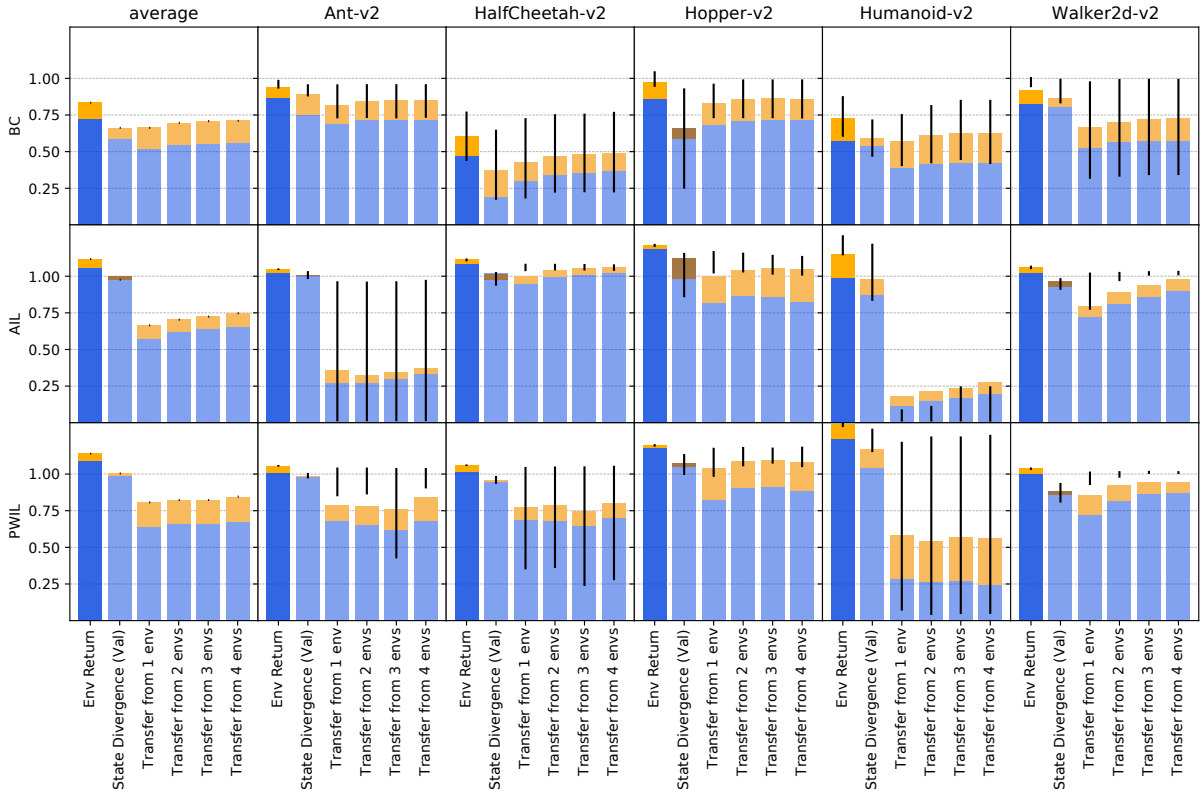


Figure 5.4: The episode return achieved by different algorithms if the HP selection is performed using transfer for OpenAI Gym environments. HPs are selected by choosing the HP configuration which performs best on a set of other tasks from the same benchmark (See Sec. 5.2.5 for the details). The results are averaged across all possible choices of the validation environments. Different bars in each subplot correspond to using a different number of environments to select HPs. We also include the selection based on the episode return and the state divergence for comparison. Early stopping is performed using the state divergence (validation) regardless of the HP selection metric. See the caption for Fig. 5.2 for additional information on this figure.

5.3.2 Hyperparameter selection via transfer

As an alternative to the metrics investigated in the previous section, we could select HPs which work best on a similar task(s) that actually has a well-defined reward function. Fig. 5.4 and 5.5 show the performance of different algorithms when we choose HPs using a set of validation environments and then test them on another environment from the same benchmark (*i.e.*, we do not consider the transfer from OpenAI Gym to Adroit here). While we could use transfer for early stopping (*i.e.*, select the timestep which worked best for the validation environments), we have noticed that it almost always resulted in worse performance than no early stopping and therefore we chose to use state divergence for early stopping

[8] For some algorithm-environment pairs, the train versions of some metrics performs better than the validation ones. We suspect that it is due to the training sets being bigger (11 trajectories for OpenAI Gym and 20 for Adroit) than the validation sets (5 trajectories). Moreover, except for BC and action MSE, the metric used to select HPs is not the metric being optimized, so it may be less crucial to have a validation set.

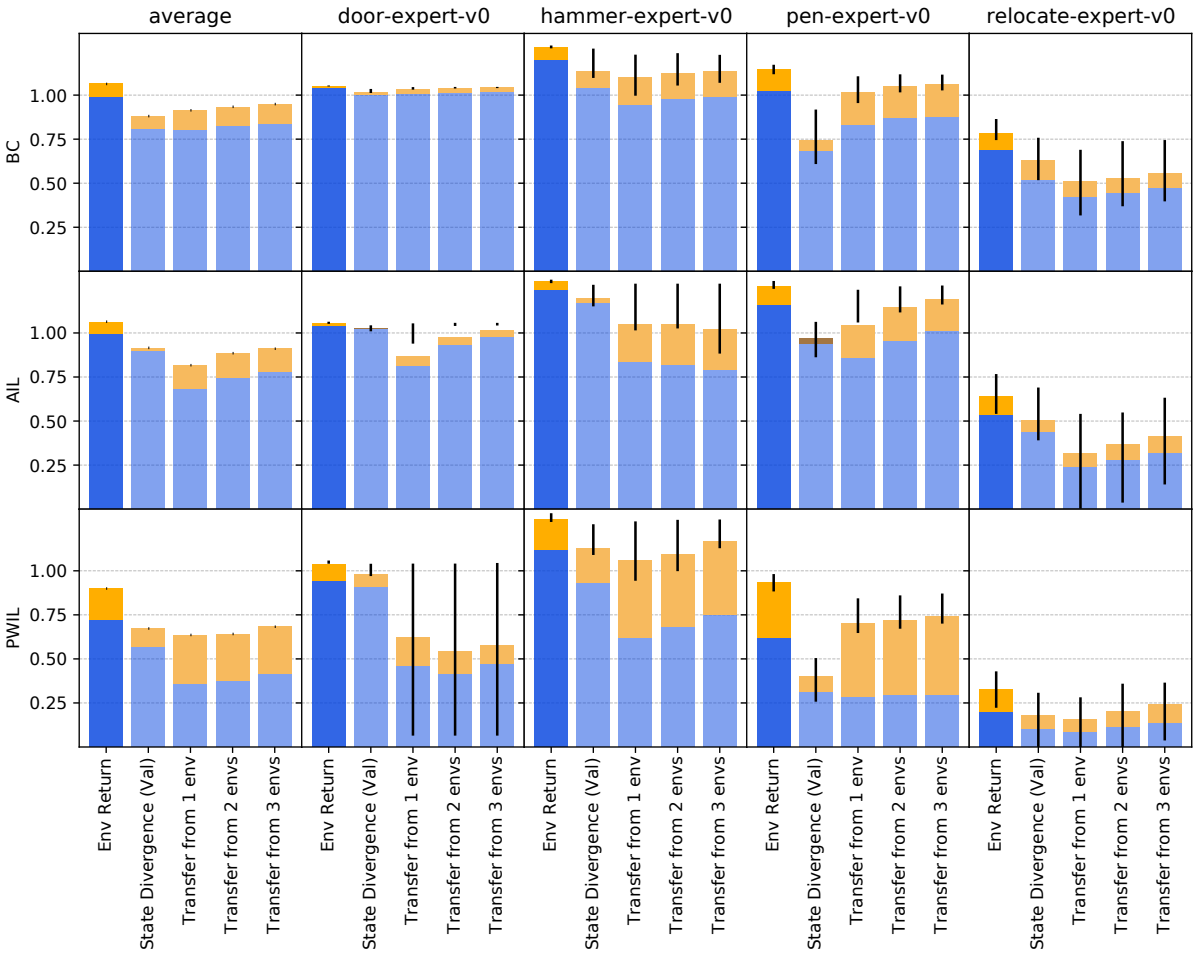


Figure 5.5: The episode return achieved by different algorithms if the HP selection is performed using transfer for Adroit environments. See Fig. 5.4 for additional information on this plot.

in the transfer experiments.

Transfer performance consistently improves with the number of validation environments used but selecting HPs based on the state divergence on the task of interest outperforms transfer even if we validate HPs on all the other tasks in the given task suite. The relatively poor transfer performance can be caused by two factors: (1) different HP configurations performing well on each task, or (2) the stochasticity of the algorithm (*i.e.*, the algorithm producing different results when run twice with the same HPs). Despite that, transferring HPs may still be preferred in some situations as it does not require tuning HPs for each new task.

Fig. 5.6 shows the transfer performance for individual pairs of validation-test environments. We observe that HPs transfer better within a suite, but the HP transfer between OpenAI Gym and Adroit often succeeds too. Especially, BC's HPs transfer very well from Adroit to Gym but not the other way around. Results also suggest that HPs transfer better to easier tasks than to harder ones: the two tasks on which the HP transfer performs worst are `Humanoid` and `relocate` which are the hardest tasks in their

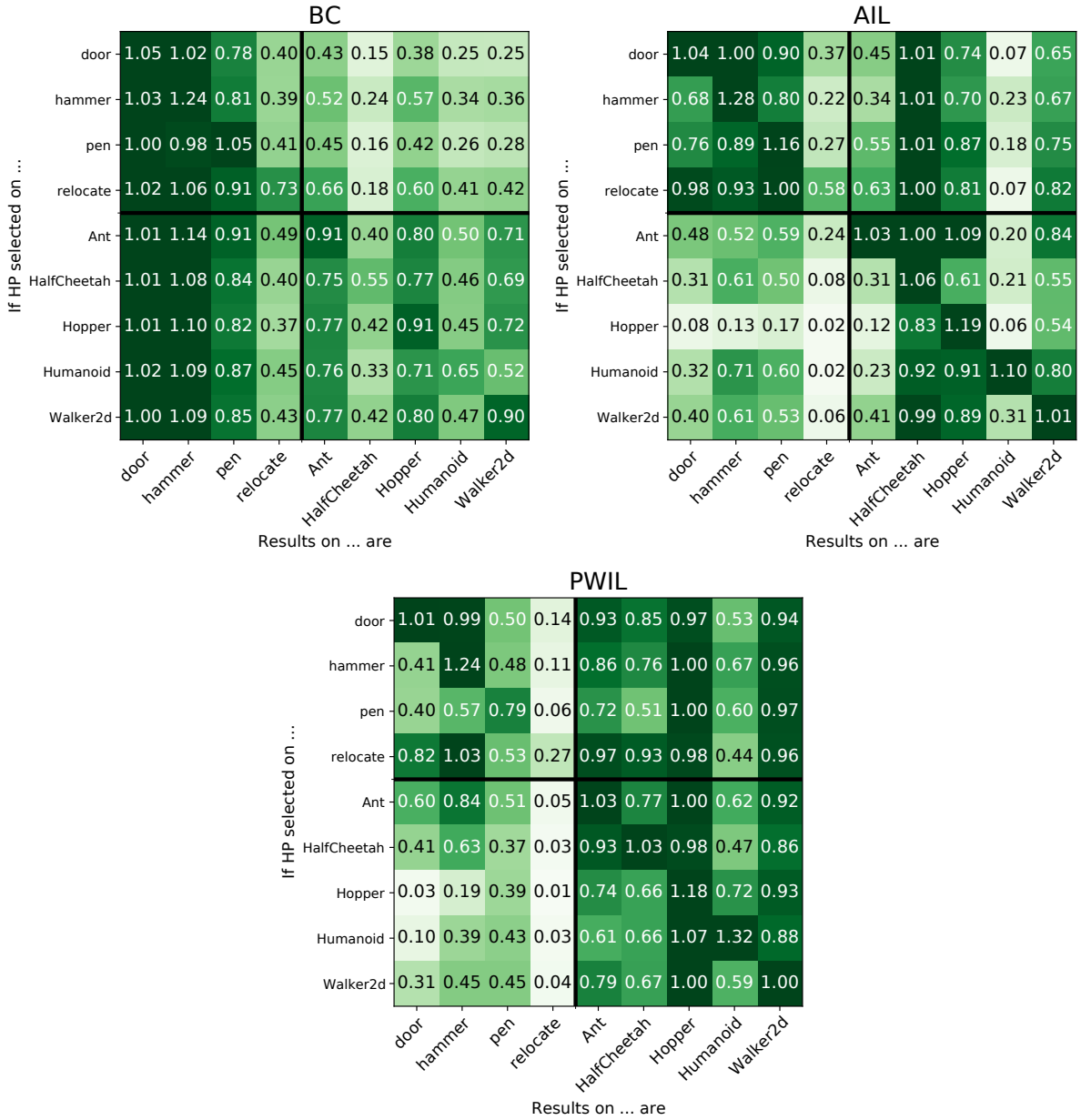


Figure 5.6: HP transfer results for individual validation-test environments pairs. Rows correspond to different validation environments and columns to different test environments. Early stopping is performed using the state divergence on the validation set of demonstrations.

respective suites.

The results suggest also that BC enjoys better HP transferability than AIL and PWIL which can be expected as it is a much simpler algorithm based on supervised learning.

A general conclusion from our experiments is that IL algorithms are quite sensitive to their HPs and

may require per-environment HP tuning for optimal performance. It is therefore important to compare IL algorithms not only in terms of their performance under optimal HPs but also in terms of their HP sensitivity and transferability.

5.4 Related Work

Hyperparameter selection is central to the performance of machine learning algorithms. In the context of supervised learning, it is common practice to select HPs on dedicated held-out data (validation) and subsequently estimate the performance of an algorithm on another set of held-out data (test). Previous work has looked into improving the selection of the best configurations using grid search [LeCun et al., 2012], random search [Larochelle et al., 2007, Bergstra and Bengio, 2012] or population based strategies Jaderberg et al. [2017].

In the context of reinforcement learning, the notions of training and validation can be conflated. This is the case if the goal is to design a policy that performs well in a single environment [Silver et al., 2016, Tesauro, 1995, Vinyals et al., 2019]. However, RL agents are typically evaluated on their ability to learn on multiple environments [Bellemare et al., 2013, Brockman et al., 2016a, Cobbe et al., 2020] with a single set of HPs. Henderson et al. [2018] highlights that poor evaluation protocols make RL algorithms hard to reproduce. For instance, although code level optimizations often explain a lot of the RL algorithms performance, they tend to be omitted in most of the recent RL publications Engstrom et al. [2020], Andrychowicz et al. [2020a], making it hard for practionners to compare and reproduce.

Imitation Learning adds another level of algorithmic complexity to the RL setting since the reward function is not available. Therefore, the learning of a reward function, which introduces its own set of extra HPs, is intertwined with the learning process of a direct RL agent [Finn et al., 2016, Ho and Ermon, 2016, Kostrikov et al., 2019]. Although previous work has identified measures of similarity not based on the reward functions Dadashi et al. [2021a], Ghasemipour et al. [2020], Ke et al. [2019], this work is, as far as we know, the first to propose a principled evaluation protocol to select HPs not based on the true but supposedly unknown reward function.

Paine et al. [2020] recently highlighted the same problem occurring in offline RL [Lagoudakis and Parr, 2003, Ernst et al., 2005, Riedmiller, 2005, Lange et al., 2012, Levine et al., 2020], where HPs are usually selected on the performance of the agent on the online environment (although the core constraint of offline RL prevents interactions with the environment). Some recent benchmarks [Gulcehre et al., 2020, Fu et al., 2020b] also propose evaluation protocols where HPs are selected by tuning on only a subset of environments.

5.5 Discussion

In this chapter, we highlighted a major flaw in current evaluation protocols of imitation learning methods. Although the promise is to design agents learning from demonstrations, the standard practice is to select agents on the reward of the task. In order to align research progress with the problem it attempts to solve, we advocate for a new evaluation protocol, where the HP selection is based on criteria available in the imitation learning setting. We investigated multiple proxies to the environment return for HP selection and early stopping. We evaluated, on 9 continuous control tasks, model selection using proxy metrics or through transfer. We demonstrated the brittleness of classical algorithms when the HP selection cannot be performed on the unknown environment return. We also showed that it is possible to select good HPs

by estimating the divergence between the distribution of states encountered by the demonstrator and the agent. This work opens the interesting question of new proxy metrics design, that can adapt to harder imitation learning settings including suboptimal demonstrations, partial observability or visual-based inputs.

So far, we studied adversarial imitation learning and the design choices that allowed to imitate human data as successfully as possible. We designed a new imitation learning algorithm and show its efficiency in the very-low data regime, showing that would could train Humanoid-v2 to run with a single sub-sampled trajectory (only 50 state-action pairs). Thereafter, we studied a key component of algorithms that is often neglected: how to select hyperparameters in this context without breaking the assumptions of the imitation learning setup.

In what follows, we do not assume anymore that the provided demonstrations are optimal. We rather consider that they may not be perfectly aligned with the task at hand, be it because the demonstrator is exploring in the environment, still learning, or just playing around without solving a specific task. However, even when demonstrations do not actually solve the goal task, or not in an optimal way, they still convey a lot of information concerning the environment. In particular, when the demonstrator is human, the demonstrations may carry all the priors, all the knowledge the human has on the task at hand. Even when solving a new task, a human never starts *tabula rasa*. They carry knowledge about, for example, the physics of the environment. They do not have to rediscover gravity every time they need to learn a physical world skill. Thus even when not actually solving the task, humans most of the time interact with their environment in a way that "makes sense".

We can therefore naturally expect that using such demonstrations may help transfer the demonstrator's priors to the agent and accelerate its learning. In the next part, we will try to make the most of these demonstrations to transmit knowledge from the human to the artificial agent. We will start by focusing on the extraction of the exploratory behavior from the demonstrator through a learned intrinsic motivation bonus reward. Afterwards, we will try to exploit the human demonstrations to discretize its continuous action space and constraint the agent to follow the human's priors.

Part II

Learning from Demonstrations: Transferring Motivations

Chapter 6

Transferring Exploratory Motivation to Artificial Agents

Contents

The study of exploration in the domain of decision making has a long history but remains actively debated. From the vast literature that addressed this topic for decades under various points of view (*e.g.*, developmental psychology, experimental design, artificial intelligence), intrinsic motivation emerged as a concept that can practically be transferred to artificial agents. Especially, in the recent field of Deep Reinforcement Learning (RL), agents implement such a concept (mainly using a novelty argument) in the shape of an exploration bonus, added to the task reward, that encourages visiting the whole environment. This approach is supported by the large amount of theory on RL for which convergence to optimality assumes exhaustive exploration. Yet, Human Beings and mammals do not exhaustively explore the world and their motivation is not only based on novelty but also on various other factors (*e.g.*, curiosity, fun, style, pleasure, safety, competition, *etc.*). They optimize for life-long learning and train to learn transferable skills in playgrounds without obvious goals. They also apply innate or learned priors to save time and stay safe. For these reasons, we propose in this chapter to learn an exploration bonus from demonstrations that could transfer these motivations to an artificial agent with little assumptions about their rationale. Using an inverse RL approach, we show that complex exploration behaviors, reflecting different motivations, can be learnt and efficiently used by RL agents to solve tasks for which exhaustive exploration is prohibitive.

6.1 Introduction

Intrinsic motivation [Deci and Ryan, 2010] has emerged as one explanation for humans' and animals' impressive learning capabilities. Steered by the need to explore their environment (whether this need is satiable or not has been a fierce debate in the behavioral psychological community [Hunt, 1965]), they are able to discover near-optimal strategies in very efficient ways. Designing artificial agents presenting such capabilities is a central goal of modern artificial intelligence and Reinforcement Learning (RL) is a popular candidate to do so. RL has addressed a variety of sequential-decision-making problems whether in games [Tesauro, 1995, Mnih et al., 2013, Silver et al., 2016] or robotics [Abbeel and Ng, 2004, Andrychowicz et al., 2020b]. Nevertheless, some simple problems remain unsolved. Current state-of-the-art methods struggle to find good policies in environments (1) where constant negative rewards may discourage the agent to explore (*e.g.*, the *Pitfall!* game from Atari), (2) where the reward is so sparse that an agent does not find any (*e.g.*, the *Montezuma's Revenge* Atari game), (3) where state and action space are big (*e.g.*, text worlds). These tasks remain fairly easy for humans, though. In order to tackle these specific problems, the use of reward bonuses, inspired by animal curiosity, was proposed to steer the agent's exploration [Şimşek and Barto, 2006, Strehl and Littman, 2008]. Even though different intrinsic bonuses have been proposed, a large majority rely on the same principle: reward for *novelty*. These methods mostly differ in how they compute this notion of *newness*. Count-based methods do it by counting how often the agent has encountered a given state [Strehl and Littman, 2008]. Pseudo-counts methods [Bellemare et al., 2016, Ostrovski et al., 2017] allow to approximate counts in large state spaces. Prediction error is also used to measure novelty, either by computing the agent's ability to predict the future [Pathak et al., 2017] or random statistics about the current state [Burda et al., 2018]. Some restrict novelty to state-action pairs that have an impact on the agent [Raileanu and Rocktäschel, 2020] or derive empowerment metrics [Mohamed and Rezende, 2015] using mutual information. All these methods naturally encourage the discovery of new states through exhaustive exploration. Yet, in most realistic environments, exhaustive exploration is (1) not feasible due to the size of the state-action space, (2) not desirable as most behaviors are unlikely to be relevant for the task at hand.

Nonetheless, human and more generally mammals exploration behaviors are governed by various motivations and constraints. Intelligent Beings do not have unlimited resources of time and energy. They optimize these resources to survive and reproduce but also to have fun [Holloway and Valentine, 2004], to help others [Byrne and Whiten, 1989] or to satisfy their curiosity. Oudeyer and Kaplan [2009] make the difference between homeostatic motivations (that encourage to stay in the "comfort zone" and generally correspond to desires that can be satiated) and heterostatic motivations (that push organisms out of equilibrium but cannot be satiated). These many desires shape the way organisms interact with their environment, encouraging them to discover new things but also to protect themselves, avoiding over-surprising events with mechanisms like fear [Lang et al., 2000]. Berseth et al. [2019] exemplified how to exploit such priors by implementing a "homeostasis" objective for RL, thereby showing how different from "novelty seeking" these priors can be. Eventually, the resource constraints stop organisms from exploring exhaustively their environment and push them to transfer knowledge from past experience. Hunt [1965] developed the idea of optimal incongruity: high-novelty is not rewarded as much as intermediate-level novelty, suggesting how curiosity is tightly connected to fear. More recently, Kidd et al. [2012] supported this hypothesis with experiments on children curiosity. Overall, novelty methods fail to model correctly human curiosity as they consider that "the newer, the better". This failure calls for a new way of definingour agent's intrinsic motivations.

In an arbitrary environment, exhaustive exploration is desirable and leads to convergence with

theoretical guarantees [Strehl et al., 2009]. But when the exploration presents some structure, one can transfer skills and priors from similar environments. Dubey et al. [2018] exemplified, in the case of simple video games, how humans priors help us to solve new problems. The authors enlighten how humans struggle to play the same underlying video game with change of the object semantics, physics modifications (*e.g.*, the gravity is rotated) or with visual similarities transformations. Overall, they show how much of the human's ability to solve a new game in a zero-shot manner is due to their prior on the environment.

In this chapter, instead of hard coding what we think an agent's motivations should be (e.g. novelty), we propose to learn a bonus that captures these sources of motivation from demonstrations. By adopting this approach, we expect to learn a bonus that implicitly helps reproducing a structured exploration behavior (*i.e.* using priors from the demonstrations to reduce the search space), in lieu of an exhaustive one. We also argue that, to a certain extent at least, this can happen without the need of extra modelling inspired by cognitive or behavioral research. To do so, we cast this problem as an inverse RL problem with the difference that only some fraction of the reward optimized by an observed agent is hidden: the intrinsic motivation bonus. The task-related reward remains provided by the environment. We then build upon Klein et al. [2013] to propose a method that allows us to recover the intrinsic motivation from demonstrations.

Therefore, our contributions are the following:

1. a modelling that allows for disentangling the reward optimized by a demonstrator from its intrinsic motivation bonus;

2. an architecture, that we call "*Show me the Way*" (SmtW), based on a cascade of supervised learning methods that extracts that exploration bonus from demonstrations;

3. an empirical evaluation showing that SmtW is able to capture different exploration priors explained by various types of motivations.

To evaluate SmtW, we validate a set of hypotheses on a controlled environment. We notably find that our method can learn structures and styles, transfer useful priors and encourages long-term planning.

6.2 Background

Markov Decision Processes. We refer to sections 2.1, 2.2 for a background on MDPs and value functions.

Exploration Bonus. For MPDs, a common strategy to encourage exploration is to augment the reward function with a bonus. This bonus generally depends on past history. For example, a bonus rewarding novelty requires remembering what has been experienced so far. Write $h_t = (s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t)$ the history up to time t, and H the set of all histories. Generally speaking, we abstract a bonus as $\mathrm{B} : \mathrm{H} \times \mathcal{A} \to \mathbb{R}$, and use it for addressing the dilemma between exploration and exploitation, which thus amounts for the agent to optimize for $R(s_t, a_t) + B(h_t, a_t)$ instead of simply $R(s_t, a_t)$. This state-of-the-art exploration bonuses all rely on memory, *e.g.* by counting the state visitation [Strehl and Littman, 2008] or through updates of a neural networks [Burda et al., 2018, Ostrovski et al., 2017]. These exploration bonuses are designed to express the prior that any source of novelty is good for exploration.

Such a prior on what is good for exploration is task-specific. Taiga et al. [2020] showed that state-of-the-art bonuses were degrading performances in most Atari games.

6.3 Show me the Way

Rather than handcrafting a bonus that encodes what we think intrinsic motivation should be (*e.g.* using novelty), we propose to learn it from demonstrations of exploratory behaviours.

We thus assume that the demonstrator learns to solve a task by exploring its environment and a simple solution would be to perform behavior cloning. Because the demonstrator is likely to use past interactions to make decisions (remembering what has been already tried so far), we could frame our problem as learning, in a supervised manner, a mapping from histories to actions. Yet, behavioral cloning suffers the behavioral drift [Ross and Bagnell, 2010], which would be exacerbated in the case of history dependent policies. Moreover, we would like to transfer this behavior to new environments and possibly to new tasks.

Imitation learning classically assumes that experts are optimizing an MDP with an unknown reward function $r_E(s,a)$. Note that this introduces a modelling bias, *i.e.* a human performing a task is not necessarily explicitly solving an MDP. In this study, we do not tackle the standard problem of inferring an optimal behavior from demonstrations, but of estimating an exploratory behavior. Yet, the latter can be reduced to the former. Taking inspiration from RL, and especially from the body of work about the exploration-exploitation dilemma, we assume that our demonstrator is optimizing for an unknown reward function $r_E(s,a)$, standing for the task objective, augmented with a trajectory-dependent intrinsic bonus $\mathrm{B}_E(h,a)$, standing for how the environment is explored. Making this bonus depend on past interactions is important as we can reasonably assume that exploration is based on memory (one would not try to always reproduce situations that were already seen). Then, we assume that the expert is optimal for the bonus-augmented reward $r_E(s,a) + \mathrm{B}_E(h,a)$, in the augmented MDP $\{\mathrm{H}, A, P, r_E + \mathrm{B}_E, \gamma\}$. That is the original MDP, with the state space being replaced by the set of all trajectories and the reward function being augmented with a bonus function. So, we reduced our original problem to Inverse Reinforcement Learning (IRL): learn a function $\widehat{RB} : \mathrm{H} \times \mathcal{A} \to \mathbb{R}$ such that the demonstrator is the (unique) optimal policy. By design, $\widehat{RB} = r_E + \mathrm{B}_E$ is a solution to this problem (even if it is not learnable exactly, as the optimal policy is invariant to many reward transformations [Ng et al., 1999]). Yet, we also assume that we know the task's reward R, or at least that we observe it in the demonstrations. Formally, it may be different from the reward R_E (even if we assume that it leads to a similar optimal behavior), but we can leverage it to disentangle the task contribution and the exploration one. For doing so, we propose to learn a bonus function $\hat{B} : \mathcal{H} \times A \to \mathbb{R}$ such that the demonstrator is optimal for $R + \hat{B}$. Notice that it does not change the problem, as it is just a reparameterization of the previous one (by setting $\hat{B} = \widehat{RB} - r$).

Thus, our IRL problem has additional constraints. First, we want to recover the bonus, for transferring to new environments or new tasks, this preclude using imitation learning methods that do not explicitly recover rewards (IRL is mandatory). Second, our specific parameterization $(r + \mathrm{B})$ precludes using IRL methods that would not allow using the observation of the task reward r along the expert trajectories. Third, the function we want to estimate is history-dependent which requires an IRL method able to use sequences as inputs.

Formalism. We assume to have access to demonstrations that are optimal according to the (known) reward of the environment *plus* an (unknown) intrinsic bonus. The environment being assumed Markovian, knowing the current state is enough to act optimally according to the task (optimizing for the environment's reward). Yet, the demonstrator also optimizes its exploration bonus, that depends on the past. To formalize things, we consider that the demonstrations are provided by a policy $\pi_E : \mathrm{H} \to \mathcal{A}$, and that the policy is optimal for the augmented MDP $(\mathrm{H}, \mathcal{A}, \mathcal{P}, r_E + \mathrm{B}_E)$, where H replaces $\mathcal{S}$ and $r_E + \mathrm{B}_E$ replaces r.

We frame our problem as learning the bonus B_E from trajectories sampled from π_E.

Our approach. If we cannot naively apply any existing IRL algorithm to our problem, it can be a source of inspiration. Especially, one suits well our problem: the set-policy framework [Piot et al., 2016]. It shows that a formal bijection between supervised learning and IRL exists. Among the covered algorithms, the *Cascaded Supervised approach to IRL* (CSI) [Klein et al., 2013] is of particular interest to us. We refer the reader to these papers for more details and we explain in details here how the CSI paradigm can be readily applied to our setting.

The demonstrator's policy, π_E, is assumed optimal for $r_E + \mathrm{B}_E$ (which is unknown), so

$$\pi_E = \pi^*_{r_E+\mathrm{B}_E}.$$

Write $Q_E(h,a) = Q^*_{r_E+\mathrm{B}_E}(h,a)$ the associated optimal Q-function. It satisfies the Bellman optimality equation (writing $h = (\dots, s)$, that is s the last state of the trajectory h and $h' = (h, a, s')$):

$$\begin{aligned} Q_E(h,a) &= r_E(s,a) + \mathrm{B}_E(h,a) + \gamma \mathbb{E}_{s'|s,a}[\max_{a'} Q_E(h',a')] \\ &= r_E(s,a) + \mathrm{B}_E(h,a) + \gamma \mathbb{E}_{s'|s,a}[Q_E(h',\pi_E(h'))]. \end{aligned}$$

Would the optimal policy and Q-function be known, we could use them to recover the optimized bonus-augmented reward using this Bellman equation:

$$r_E(s,a) + \mathrm{B}_E(h,a) = Q_E(h,a) - \gamma \mathbb{E}_{s'|s,a}[Q_E(h',\pi_E(h'))].$$

Now, the quantities in the right hand side are unknown, but they can be estimated in an indirect way.

Assuming that the actions are discrete, we can learn the policy π_E by mapping histories to actions (using, for example, an LSTM network). Write $\hat{\pi} : \mathrm{H} \to \mathcal{S}$ the resulting policy, or classifier. If we train it by minimizing a cross-entropy loss, what we learn indeed are logits $\hat{Q}(h,a)$, and the classifier is $\hat{\pi}(h) = \mathrm{argmax}_a \hat{Q}(h,a)$. Said otherwise, $\hat{\pi}$ is greedy with respect to $\hat{Q}$, that can thus be interpreted as an optimal Q-function for an unknown reward. Using the Bellman equation, we can recover this reward:

$$r(s,a) + \hat{\mathrm{B}}(h,a) = \hat{Q}(h,a) - \gamma \mathbb{E}_{s'|s,a}[\hat{Q}(h',\hat{\pi}(h'))]. \tag{6.1}$$

By Bellman, as $\hat{\pi}$ is greedy w.r.t. $\hat{Q}$, we have that $\hat{\pi}$ and $\hat{Q}$ are respectively the optimal policy and Q-function for the reward $r + \hat{\mathrm{B}}$. We cannot use directly Eq. (6.1), the model being unknown, but we can sample the right hand side and estimate $\hat{\mathrm{B}}$ by solving a regression problem.

Therefore, we have reduced our initial problem to a sequence of supervised learning problem. Our algorithm is indeed CSI, up to the fact that we consider trajectories instead of states, and parameterize the bonus with the reward task. As such, the theoretical results of Klein et al. [2013] applies to our setting. Notably, we would have that

$$0 \leq \mathbb{E}_{h\sim\pi_E}[\max_a Q^*_{R+\hat{B}}(h,a) - Q^{\pi_E}_{R+\hat{B}}(h,\pi_E(h)] \leq \mathcal{O}\left(\frac{\epsilon_1+\epsilon_2}{1-\gamma}\right),$$

with ϵ_1 the classification error and ϵ_2 the regression error. This means that the demonstrator policy is close to optimal if these errors are small, for the learnt bonus function. One could argue that this bound trivially holds for $R + \hat{B} = 0$, when all behaviors are optimal. Yet, this is unlikely, as for having the classification error ϵ_1 small, we must have $\hat{Q}(h,\pi_E(h)) > \hat{Q}(h, a \neq \pi_E(h))$ w.h.p., and thus learn an informative bonus.

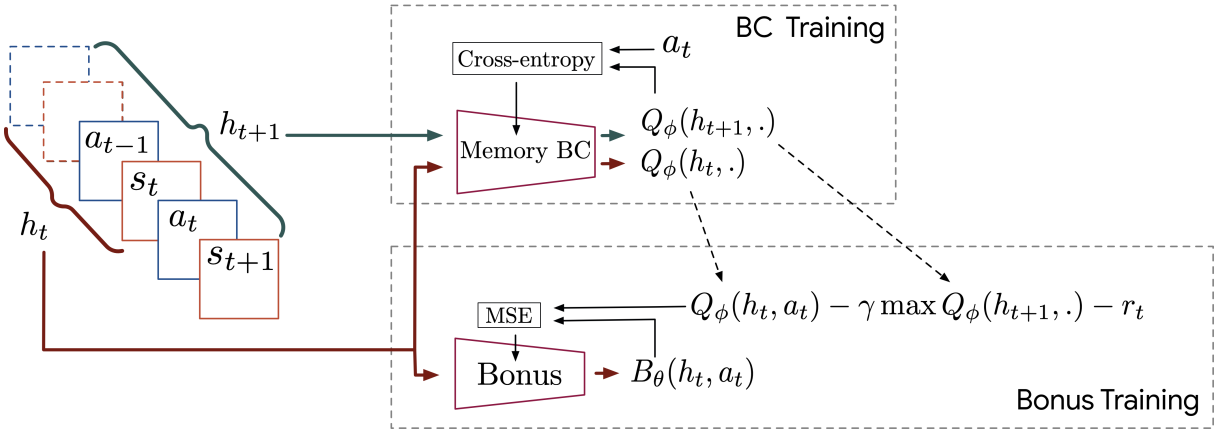


Figure 6.1: Trajectories $(s, a, \dots)$ are generated by a demonstrator exploring its environment. In order to recover a bonus that can explain its behavior, a BC policy parameterized with an LSTM is trained to predict the actions of the demonstrator from its trajectories of states, by minimizing $\mathcal{L}^{\text{BC}}$. The policy's logits Q_ϕ are interpreted as optimal Q-values and used to compute a regression target. A bonus function B_θ, parameterized with an LSTM, is then trained to predict it, by minimizing $\mathcal{L}^{\text{reg}}$.

Implementation. More concretely, we consider $\text{softmax}(Q_\phi)$ to be a neural network classifier with LSTM [Hochreiter and Schmidhuber, 1997] units, ϕ being the set of parameters and Q_ϕ being the logits. We train π_ϕ to do behavioral cloning, that is to predict the demonstrator actions a_E based on its past interactions h_E, by minimizing a cross-entropy loss:

$$\mathcal{L}^{\text{BC}} = -\ln(\text{softmax}(Q_\phi(h_E, a_E)),$$

with $Q_\phi(h, a)$ the a^{th} logit for input h. If the classifier learns correctly, the logits of the resulting network should satisfy $Q_\phi(h_E, a_E) > Q_\phi(h_E, a)$ for $a \neq a_E$, and the class predicted by the classifier will be $\pi_\phi(h) = \text{argmax}_a\, Q_\phi(h, a)$. Hence, as explained above, one can interpret Q_ϕ as an optimal Q-function (hence the notation), and π_ϕ as the associated optimal policy. Recall that these quantities can be related to the bonus-augmented reward through Bellman:

$$Q_\phi(h, a) = r(s, a) + \text{B}(h, a) + \mathbb{E}_{s'|s,a}[Q_\phi(h', \pi_\phi(h'))].$$

Then, we can learn a network B_θ (parameterized by θ, with LSTM units) by minimizing a square-loss, the regression target being $Q_\phi(h_E, a_E) - \gamma Q_\phi(h_E{}', \pi_\phi(h_E{}')) - R(s_E, a_E)$, an unbiased sample of what would give the true Bellman equation. However, we only observe optimal actions (according to $r + \text{B}$), so this alone would hardly generalize to suboptimal ones. Therefore, we propose a heuristic, that consists in regressing for suboptimal actions towards B_{min}, a hyperparameter of the algorithm. For example, it could be set to $\min(Q_\phi(h_E, a_E) - \gamma Q_\phi(h_E{}', \pi_\phi(h_E{}')) - R(s_E, a_E))$, the minimum being over transitions in the dataset. This gives the following loss, for a transition $(h_E, a_E, h_E{}')$, and for $\overline{a_E}$ being sampled randomly

in $\mathcal{A} \setminus \{a_E\}$:

$$\mathcal{L}^{\text{reg}} = \Big(Q_\phi(h_E, a_E) - \gamma Q_\phi(h_E{}', \pi_\phi(h_E{}')) - R(s_E, a_E) - \text{B}_\theta(h_E, a_E)\Big)^2 + \Big(\text{B}_{\min} - \text{B}_\theta(h_E, \overline{a_E})\Big)^2.$$

To sum up, we train a BC policy by minimizing $\mathcal{L}^{\text{BC}}$. The implicit resulting logits are considered optimal Q-values, that are in turn used to learn the bonus B_θ by minimizing the loss $\mathcal{L}^{\text{reg}}$ (Figure 6.1).

6.4 Experiments

We aim at providing insights on what *priors* SmtW is able to extract from the demonstrations and specifically, we wish to verify that SmtW is able to encourage a *structured exploration* of the environment. In order to thoroughly study the method, we test it on a grid-world where we are able to design controllers with specific behaviors. As in IRL, studying the return of an agent trained with our bonus is only a proxy to evaluate SmtW's quality and is not informative on the priors the bonus conveys. We thus focus our experiments on analyzing the priors that were extracted from the demonstrations by the method. More specifically we wish to answer the following questions: (1) Is SmtW encouraging the demonstrator's behavior more than a random one? (2) Is SmtW capturing the demonstrator's style, its way of exploring the environment? (3) Does SmtW capture the skills required to solve the task? (4) Does SmtW encourage novelty seeking? (5) Does SmtW capture the constraints the demonstrator may be submitted to? To do so, we design controlled behaviors and study the bonus returned along these specific behaviors by SmtW, as described in Fig. 6.3. Given a behavior A and a behavior B, this allows to check if a given bonus encourages behavior A over B or vice versa or rewards them equivalently.

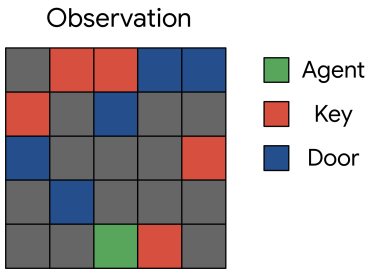


Figure 6.2: KeysDoors(N=5).

After addressing these questions, we also verify that a simple agent can benefit from SmtW to actually solve efficiently a task.

The environment. We introduce a specific environment to answer these. We require this environment to be procedurally-generated in order to test SmtW's ability to generalize to unseen environments. We require the environment to be complex enough so that exhaustive exploration is prohibitively expensive. To achieve this, we introduce the KeysDoors grid-world of size $N \times N$.

It contains N keys and N doors, modeled by two different colors. The agent has a third color. The goal is to find the correct key and to open the correct door with it. As doors (resp. keys) are indistinguishable (except by their locations), an explorer has to try the different keys on the different doors. Actions available are {*go left, go right, go up, go down, take, open, wait*}. When an agent makes the action "take" on a key, it is then able to move with it. Actions "open" or "take" make the agent lose the key it was previously holding. To solve the task, the agent has to go to the correct key, take it, go to the correct door without doing action "take" or "open" on the way (so as not to lose the key), and then "open" the door. We need the environment to require *perseverance* so we made the reward function -1 for any actions

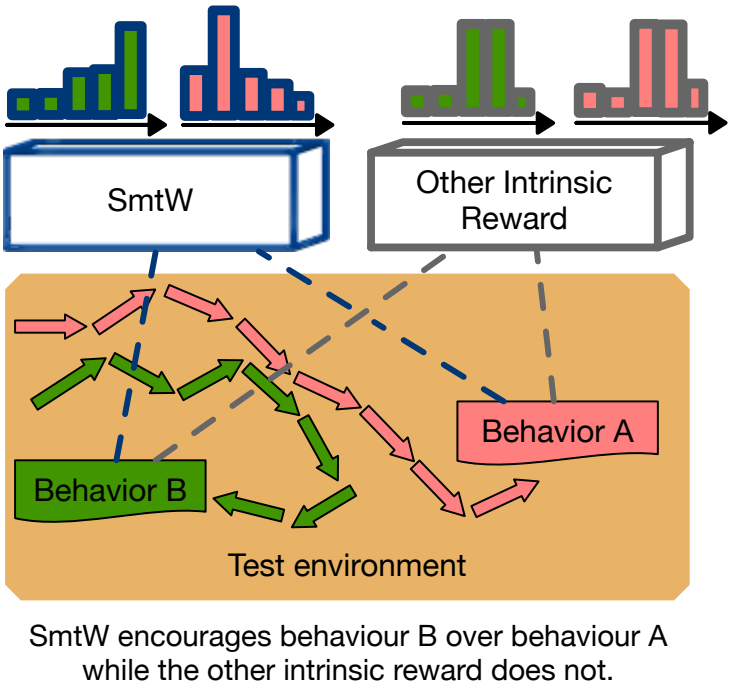


Figure 6.3: Comparing intrinsic rewards on what behavior they encourage or discourage on new unseen environments.

but the *wait* action, that is rewarded 0. Opening the correct door with the correct key gives a reward of 100 and terminates the episode. It requires perseverance as a "lazy" policy would get a return of 0 whereas trying to find the 100 reward gives -1 at each step. This is a well known issue in RL that simple exploration leads to such lazy solutions.

The KeysDoors environment is generated procedurally. For each column, locations for a door and a key are sampled uniformly without replacement. Thus, there is exactly one key and one door on each column and these cannot be at the same location. The "correct" key is then uniformly sampled among the keys and the "correct" door is sampled uniformly among the doors. The initial position of the agent is sample uniformly on the grid. The environment gives both a ground-truth-state (an integer representing the current state), unused by SmtW as well as an RGB observation (as shown in Fig. 6.2), used by SmtW. Figure 6.4 shows a trajectory in one possible instance of the KeysDoors environment with $N = 5$. Every observation x (an $N \times N \times 3$ tensor) is normalized between 0 and 1 by dividing by 255.

The demonstrations . For a given instance of the environment, the demonstrator navigates between keys and doors and tries key/door pairs in a precise order. It takes the first key on the left and tries it on the first door on the left, then it tries the same key on the second door etc. Once it has tried the first key on every door, it repeats the operation with the second key and proceeds further this way. The episode ends when the demonstrator finds the right key/door pair and obtains the reward. Then it "exploits", taking the correct key and opening directly the correct door five consecutive times. Note that this also simulates the non-stationnarity happening in most goal-directed task solving processes. One first mainly explores and then exploits more and more.

Train vs. Test. The bonus is always used in new test environments, unseen in the demonstrations. SmtW's ability to generalize to new environments is thus tested in all the following experiments. Given the possible positions of the keys, of the doors and then of the correct key and the correct door, there are $(N-1)N^3$ possible instances of the environment.

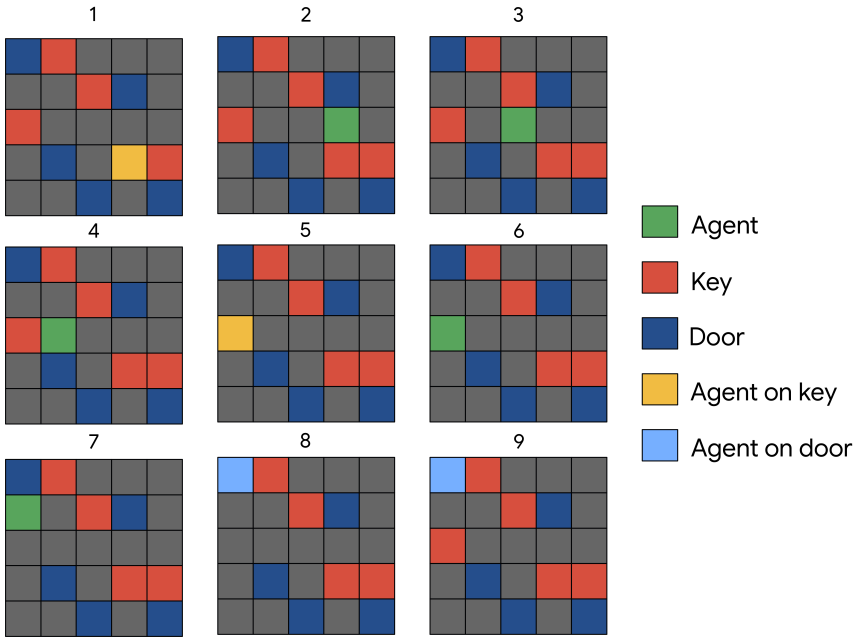


Figure 6.4: A trajectory of length 9 in an instance of the KeysDoors(N=5) environment.

The behaviors that are designed to study what is actually encouraged or discouraged by SmtW are the following. Their associated bonus is always studied in test instances of the environment.

- The demonstrator behavior acts as described previously, sequentially trying key/door pairs.
- The *random* behavior takes random actions. Trajectories are limited to 1000 steps.
- The *demonstrator inverse* behavior is similar to the demonstrator as it navigates to a key, takes it, navigates to a door and opens it. However, the key/door pairs are tried in the reverse order to the demonstrations.
- The *demonstrator random* behavior is also similar but tries the key/door pairs in a random order.
- The *dummy demonstrator* behavior navigates exactly like the demonstrator but drops the key at a random time on the way to the door (uniformly sampled on the path to the door) by taking action *open*. The trajectories are limited to 1000 steps.
- The *standing still* behavior remains in its original position by only taking the *wait* action.
- The *waiting demonstrator* behavior acts like the demonstrator but has a probability 0.1 of waiting at each step.
- The *unsafe demonstrator* acts like the demonstrator but takes this action *take* each time it moves until it has a key. Taking action *take* somewhere else than on a key can be considered as breaking a safety constraint that the demonstrator respects strictly.

A trajectory of an agent moving to a key, taking it, moving to a door and trying to open it with the key is shown in Fig. 6.4.

6.4.1 Bonus analysis

We train the SmtW bonus on 200 KeysDoors(N=5) training environments with 10 demonstrations for each of them. The implementation choices are detailed in Sec. 6.4.3. In order to study the priors extracted from the demonstrations, we study the bonus given by SmtW along various trajectories following a given behavior. We thereafter plot the distribution of received bonus along the various controlled behaviors on 20 test environments, unseen during the training of SmtW. We compare the bonus given by SmtW along these trajectories to the one that would be given by a count-based [Strehl and Littman, 2008] and a random network distillation bonus [Burda et al., 2018]. The very same trajectories are presented to each bonus.

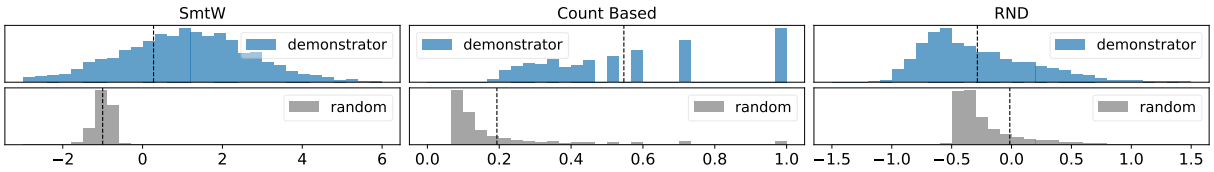


Figure 6.5: Bonus distribution received by an demonstrator's behavior (top) and a random behavior (bottom). We can say that a bonus encourages more a behavior A than a behavior B if the distribution of bonus along trajectories following A are globally higher than the one along trajectories following B. SmtW encourages the demonstrator behavior over the random behavior.

Does SmtW encourage a structured exploration more than a random one? We compare in Figure 6.5 the distribution of bonus received along random trajectories to the ones obtained by the demonstrator's behavior. Recall that SmtW has been trained on similar environments but is here tested on different ones. It is thus provided with trajectories unseen during training.

As shown on Figure 6.5, the demonstrator's behavior (top) is more rewarded by SmtW than the random behavior (bottom). We can conclude that SmtW encourage the agent to follow a demonstrator-like behavior on new unseen environments more than a random behavior. The count-based bonus also rewards the demonstrator's behavior more than a random one as a random behavior explores the environment very locally. Surprisingly, RND rewards the random behavior more than the demonstrator's one. This might be explained by the fact that the demonstrator visits several times the same state in order to explore correctly. Indeed the demonstrator has to go several times to the same key to take it and try it on the several doors.

Does SmtW capture the demonstrator's style, his way of exploring the environment? We show in Figure 6.6 the distribution of bonus received along different behaviors: the *demonstrator* one, the *demonstrator inverse* one as well as the *demonstrator random* one. These three behaviors lead to the same outcome but we hope to capture the demonstrator's exploration bias and see if it encourages the behaviors that tries the key/door pair in the same order as in the demonstrations.

As shown on Figure 6.6, the count-based bonus and RND reward similarly the three behaviors, as they lead to the same amount of novelty. Only the order in which the key/door pairs are tried is change.

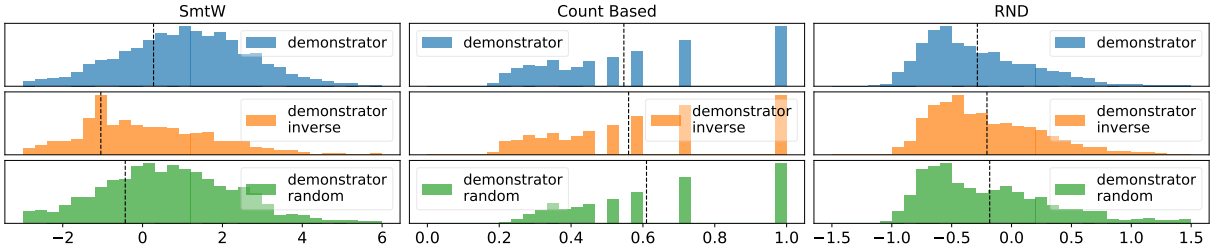


Figure 6.6: Bonus distribution received by the *demonstrator* behavior (top), the *inverse demonstrator* behavior (middle), and the *random demonstrator* behavior. (bottom).

SmtW, on the contrary, encourages to reproduce the demonstrator bias. It rewards more the behavior trying the key/door pairs in the same order as in the demonstrations.

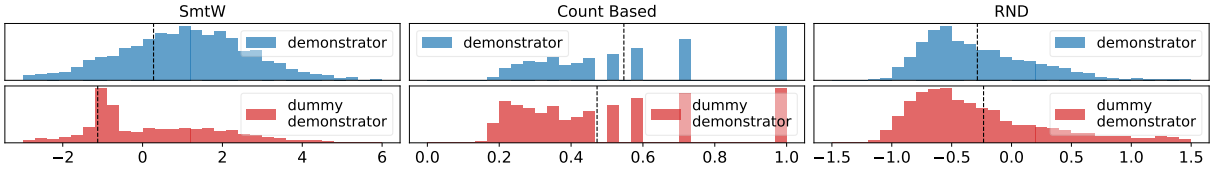


Figure 6.7: Bonus distribution received by the *demonstrator* behavior (top) and by the *dummy demonstrator* behavior, acting (almost) like the demonstrator but releasing the key on the way to the door (bottom).

Does SmtW capture the priors useful to solve the task? Figure 6.7 shows the distribution of bonus received by the *demonstrator* behavior and compares the bonus received to the one received when following the *dummy demonstrator* one.

As shown on Figure 6.7, the count-based bonus and RND reward equivalently these two behaviors as they bring the same amount of novelty (both in term of ground-truth-state and observations). SmtW does not reward the *dummy demonstrator* behavior as much as the expert one and we can interpret the lower distribution mode (SmtW-bottom) as the bonus obtained after loosing the key. We can argue that SmtW has somehow captured the prior that it is useful to navigate from the key to the door without loosing the key, as it rewards more the *demonstrator* behavior than the *dummy demonstrator* one.

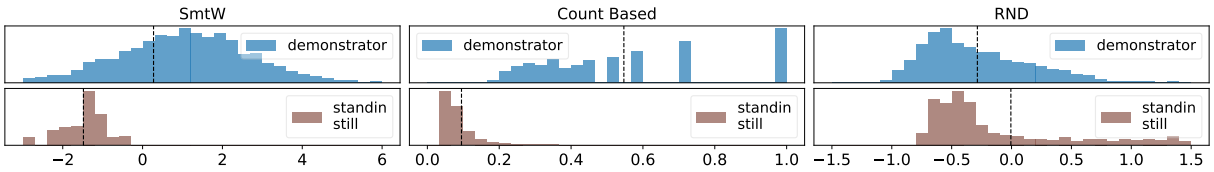


Figure 6.8: Bonus distribution received by the *demonstrator* behavior (top) and by the *standing still* behavior (bottom).

Does SmtW encourage long-term exploration? As the environment gives a reward of -1 for taking any action but the *wait* action, an agent not exploring sufficiently would quickly converge to the policy only taking action *wait* to avoid negative rewards (verified in Figure 6.11). This same problem is

visible in the *Pitfall!* game, where the best agents learn a policy obtaining 0 reward, while persevering humans get much higher scores. We show in Figure 6.8 the distribution of bonus obtained by the *standing still* behavior.

As shown on Figure 6.8, SmtW rewards much less a behavior not seeking novelty. As expected the count based gives a bonus very close to 0 for such a behavior. Perhaps surprisingly, RND rewards negatively this behavior but not with an average bonus lower than the demonstrator's behavior. This might be also due to the designed bonus normalization that RND uses (zero-mean unit-variance).

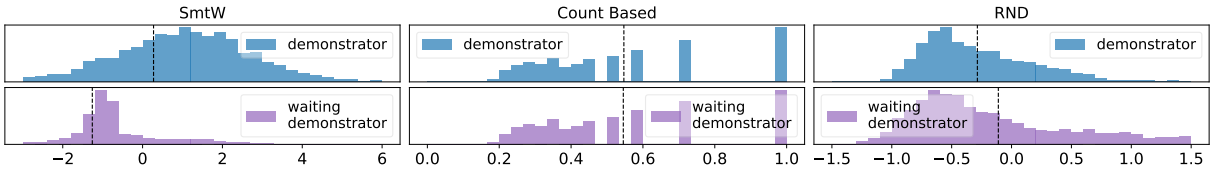


Figure 6.9: Bonus distribution received by the *demonstrator* behavior (top) and by the *waiting demonstrator* behavior (bottom).

Does SmtW capture the constraints the demonstrator may be submitted to? A demonstrator can be subject to time or energy constraints. In the demonstrations, the demonstrator tries to explore the environment as fast as possible and does not take action *wait* on his way to keys and doors. We compare the bonus distribution obtained by the *waiting demonstrator* behavior to the one obtained by the *demonstrator* one. As shown on Figure 6.9, RND and the count-based bonus reward equivalently these two behaviors. On the other hand, SmtW rewards less the *waiting demonstrator* behavior. We argue it has somehow captured the prior resulting from the resource constraint that leads the demonstrator to try the key/door pairs as fast as possible. In other words, it favors behaviors that, as shown in the demonstrations, discard the *wait* action to simplify exploration of the MDP. What is more, a demonstrator might be subject to safety constraints. As example, it might be dangerous for a robot to try an action in an inappropriate place. The demonstrations minimize the number of time they use the action "take" and only do it when on keys. We can consider that the demonstrator's behavior complied with safety constraints. We show in Figure 6.10 the bonus distribution obtained by the demonstrator's behavior and compare it with the one obtained by the *unsafe demonstrator*. As shown on Figure 6.10, the RND and

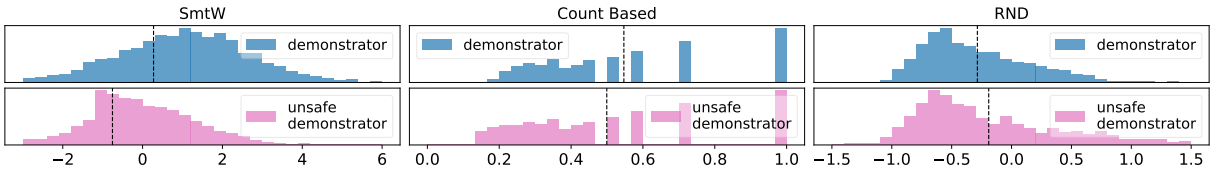


Figure 6.10: Bonus distribution received by the *demonstrator* behavior (top) vs. by the *unsafe demonstrator* behavior (bottom).

the count-based bonuses reward equivalently these two behaviors. This is expected as they bring the same amount of novelty. In contrast, SmtW rewards less the *unsafe demonstrator* behavior, capturing the safety prior the demonstrator have been subject to.

Overall, we argue that SmtW is able to recover some important bias and constraints inherent to the demonstrations. Hand-crafting a bonus expressing these motivations could be extremely complicated and

we demonstrated that SmtW is able to generalize these motivations to unseen environments.

6.4.2 Training an agent on the bonus

We now wish to check that an agent can benefit from SmtW. We thus train a Q-learning agent with SmtW and compare the results with that of a simple ϵ-greedy (ϵ=0.1) exploration strategy and a count-based bonus with $B(s,a) = N(s,a)^{-1/2}$.

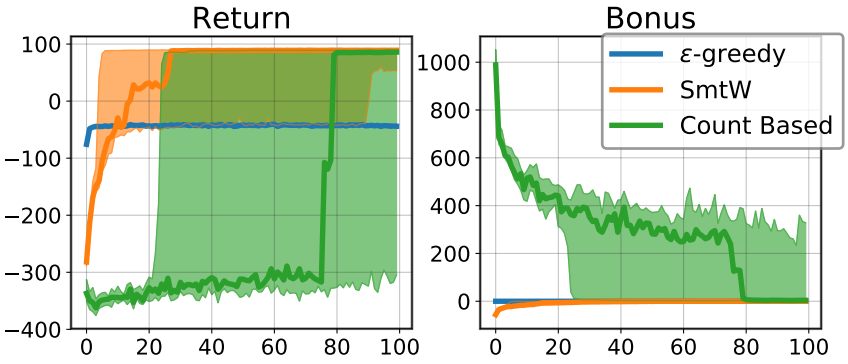


Figure 6.11: Median and min/max values of the return per episode (left) and of the total bonus per episode(right).

The results are averaged over 10 newly generated environments, unseen during SmtW training.

For each of these environments, the experiment is repeated twice. We present, for each algorithm, the best result after a hyper-parameter search. The bonus given by our method is computed to capture the exploratory behavior of the demonstrator. In order for the agent not to keep exploring forever, our bonus is here divided by $\sqrt{k}$ with k the number of step of training.

As Figure 6.11 shows, the Q-learning with an ϵ-greedy exploration strategy quickly gets stuck in "waiting" at each timestep. SmtW encourages the agent to visit its environment and solves the 10 new environments much faster than the count-based method that push for exhaustive exploration.

6.4.3 Implementation details

Our method works directly with visual inputs, as shown in Fig. 6.2. The network used for the behavioral cloning policy π_ϕ has the following architecture: an LSTM with 64 units, a fully-connected layer with 512 units and relu activation and an output layer with as many units as there are actions available in the environment (7 for KeysDoors). It is trained with the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 10^{-3} and a batch size of 1. It uses the visual input from the environment and not the ground-truth state.

The network used for the regression of the bonus B_θ has the same architecture but an output layer with a single unit. It is trained with the Adam optimizer, a learning rate of 10^{-4} and a batch size of 1. The discount factor used in SmtW is set to $\gamma = 0.99$.

For experiment shown in Figure 6.11, the tabular Q-learning is trained on the 10 test environments twice and the figure shows the median and the min/max values. For each of the compared algorithms, we sweep over the agent learning rate over the following values: $[0.01, 0.1, 0.5, 0.7]$. Only the result of

the learning rate with the highest median return over the 10×2 runs is shown for each algorithm. The ε-greedy strategy is used for all methods with $\varepsilon = 0.1$. Even though the agent is tabular, we recall that SmtW itself does not access the ground-truth state of the environment. It works from observations. The count-based bonus, on the contrary, counts ground-truth states.

6.5 Related Work

Intrinsic Motivation. Intrinsic motivation is essential to mental development [Oudeyer et al., 2007] and we can argue that this may, in consequence, be an essential component for computational learning. Oudeyer and Kaplan [2009] argue that all humans respond to intrinsic motivations. Young infants motivations can be qualified as more chaotic as they push children to bite, throw, grasp or shout in order to learn. Adults, in contrast, have more structured intrinsic motivations, activated, for instance, when they play games, read novels or watch movies. Correctly using these numerous intrinsic motivations can be key to train agents that solve more and more difficult tasks. Instead of modeling such intrinsic motivations to mimic cognitive processes, we learn them from demonstrations.

Exploration. In order to provide an exploration signal to the agent, [Strehl and Littman, 2008] proposed the very intuitive count-based method in order to measure novelty. Counting how many times the agent has been in a given state, it rewards less visited states. Several methods extended this idea to large state-space problems [Ostrovski et al., 2017, Bellemare et al., 2016, Tang et al., 2017, Machado et al., 2018], where it is not possible to count state occupancy. Intrinsic curiosity is also commonly computed as a prediction error, either trying to predict the environment's dynamics [Pathak et al., 2017, Raileanu and Rocktäschel, 2020] or random statistics about the current state [Burda et al., 2018]. Different methods try also to measure surprise as a prediction gain [Schmidhuber, 1991, Houthooft et al., 2016]. Instead of designing such a bonus, we aim at learning one from demonstrations.

Learning from demonstrations. We refer to section 2.5 for an introduction on Learning from Demonstrations. Our methods differs from imitation learning as it does not assume that demonstrations are optimal but rather try to answer the question: "In what way is the demonstrator's behavior deviating from an optimal policy?". Moreover, we do not seek to recover a reward as in IRL but rather to recover a bonus explaining which, added to the environment reward, explains the demonstrator's behavior. Facing the same problem that the usual proxy to control the algorithm quality (training an agent on the inferred bonus) is not informative, we decided to study our method through its response to various behaviors.

6.6 Discussion

In this work, we present a novel method for extracting an intrinsic bonus from the demonstrations. The method we introduce is offline and does not require environment interactions to recover the bonus, unlike recent adversarial imitation methods who need numerous interaction in order to recover a reward function. Anyway, those methods could not be readily applied to our problem, as they do not explicitly compute a reward function. Moreover, to the best of our knowledge, this is one of the very first method to recover some kind of reward that is history-dependent. We show how this bonus generalizes to unseen environments and is able to convey long-term priors. We exemplified the approach on a simple yet didactic and challenging example. Yet, testing the method on a larger-scale environment would require human

exploratory demonstrations. Gathering such a dataset is costly and very few are already available, none of them really covering our setting. Even though the given example is simple, this novel approach of capturing the demonstrator's bias could potentially lead to new lines of work in RL. For instance, one could use our method to implement *behavioral style-transfer* in RL and show to an agent a specific way to solve the task thanks to demonstrations. Combining a reward and biases extracted from demonstrations may also help for robotic tasks, where some aspects of the task are easily programmable with a reward but some expectations on how to solve the task may be easier to transmit thanks to demonstrations. This could also lead to some advances in tackling mispecified rewards. Using both a reward, that would contain information on the task to solve but not fully describe the constraints of the problem and demonstrations to correct the reward can be key to train sequential controllers in complex dynamics.

Using intrinsic motivation is an indirect way to convey human priors. As part of the reward, the learning agent has to slowly integrate the priors on the environment through interactions with the environment. In the following chapter, will we focus on developing a more direct way of transferring human priors to artificial agents, and show its efficiency on realistic, large-scale robotics tasks.

Chapter 7

Transferring Human Priors from Demonstrations through Discretization

Contents

In reinforcement learning, discrete actions, as opposed to continuous actions, result in less complex exploration problems and the immediate computation of the maximum of the action-value function which is central to dynamic programming-based methods. In this chapter, we propose a novel method: Action Quantization from Demonstrations (AQuaDem) to learn a discretization of continuous action spaces by leveraging the priors of demonstrations. This dramatically reduces the exploration problem, since the actions faced by the agent not only are in a finite number but also are plausible in light of the demonstrator's behavior. By discretizing the action space we can apply any discrete action

deep RL algorithm to the continuous control problem. We evaluate the proposed method on three different setups: RL with demonstrations, RL with play data –demonstrations of a human playing in an environment but not solving any specific task– and Imitation Learning. For all three setups, we only consider human data, which is more challenging than synthetic data. We found that AQuaDem consistently outperforms state-of-the-art continuous control methods, both in terms of performance and sample efficiency on a variety of hard manipulation tasks. We provide visualizations and videos on the website: https://google-research.github.io/aquadem/.

7.1 Introduction

With several successes on highly challenging tasks including strategy games such as Go [Silver et al., 2016], StarCraft [Vinyals et al., 2019] or Dota 2 [Berner et al., 2019] as well as robotic manipulation [Andrychowicz et al., 2020b], reinforcement learning holds a tremendous potential for solving sequential decision making problems. RL relies on Markov Decision Processes [Puterman, 2014] as its cornerstone, a general framework under which vastly different problems can be casted.

There is a clear separation in the class of MDPs between the finite discrete action setup, where an agent faces a finite number of possible actions, and the continuous action setup, where an agent faces an infinite number of actions. When the number of actions is small, the former is arguably simpler, since exploration is more manageable with a finite, reasonable number of actions, and computing the maximum of the action-value function is straightforward (and implicitly defines a greedily-improved policy). In the continuous action setup, the parametrized policy either directly optimizes the expected value function that is estimated through Monte Carlo rollouts [Williams, 1992], which makes it demanding in interactions with the environment, or optimizes a parametrized state-action value function [Konda and Tsitsiklis, 2000] hence introducing additional sources of approximations.

Therefore, a workaround consists in turning a continuous control problem into a discrete one. The simplest approach is to naively (*e.g.* uniformly) discretize the action space, an idea which dates back to the "bang-bang" controller [Bushaw, 1953]. However, such a discretization scheme suffers from the curse of dimensionality. Various methods have addressed this limitation by making the strong assumption of independence [Tavakoli et al., 2018, Tang and Agrawal, 2020, Andrychowicz et al., 2020b] or of causal dependence [Metz et al., 2017, Vinyals et al., 2019, Sakryukin et al., 2020, Tavakoli et al., 2021] between the action dimensions which are typically complex and task-specific (e.g. autoregressive policies, pointer networks based architectures).

In this work, we introduce a novel approach leveraging the prior of human demonstrations for reducing a continuous action spaces to a discrete set of meaningful actions. The proposed method does not suffer from the curse of dimensionality and does not require any task specific assumption. Demonstrations typically consist of transitions experienced by a human in the targeted environment, performing the task at hand or not. They are of particular interest in cases where the reward function is hard to define [Russell, 1998, Ng et al., 1999], to facilitate exploration [Salimans and Chen, 2018, Nair et al., 2018] or to build behavioral priors [Singh et al., 2020].

We thus propose Action Quantization from Demonstrations, or AQuaDem, a novel paradigm where we learn a state dependent discretization of a continuous action space using demonstrations, enabling the use of discrete-action deep RL methods by virtue of this learned discretization. We formalize this paradigm, provide a neural implementation and analyze it through visualizations in simple grid worlds. We empirically evaluate this discretization strategy on three downstream task setups: Reinforcement

Learning with demonstrations, Reinforcement Learning with play data, and Imitation Learning. We test the resulting methods on robotics tasks and show that they outperform state-of-the-art continuous control methods both in terms of sample-efficiency and performance on every setup.

7.2 Preliminaries

Markov Decision Process. We model the sequential decision making problem as a Markov Decision Process (MDP) [Puterman, 2014, Sutton and Barto, 2018]. Details can be found in sections 2.1 and 2.2.

Value-based RL. The Bellman [1957] operator $\mathcal{T}$ connects an action-value function Q for the state-action pair (s, a) to the action-value function in the subsequent state s':

$$\mathcal{T}^{\pi}(Q)(s,a) := r(s,a) + \gamma \mathbb{E}\big[Q(s',a') | a' \sim \pi(s), s' \sim \mathcal{P}(s,a)\big].$$

Value Iteration (VI) [Bertsekas, 2000] is the basis for methods using the Bellman equation to derive algorithms estimating the optimal policy π^*. As stated in section 2.2,the prototypical example is the Q-learning algorithm [Watkins and Dayan, 1992], which is the basis of *e.g.* DQN [Mnih et al., 2013], and consists in the repeated application of a stochastic approximation of the Bellman operator $Q(s,a) := r(s,a) + \gamma \max_{a'} Q(s',a')$, where (s,a,s') is a transition sampled from the MDP. The Q-learning algorithm exemplifies two desirable traits of VI-inspired methods in discrete action spaces that are **1)** bootstrapping: the current Q-value estimate at the next state s' is used to compute a finer estimate of the Q-value at state s, and **2)** the exact derivation of the maximum Q-value at a given state. For continuous action spaces, state-of-the-art methods [Haarnoja et al., 2018a, Fujimoto et al., 2018] are also fundamentally close to a VI scheme, as they rely on Bellman consistency, with the difference being that the argument maximizing the Q-value, in other words the parametrized policy, is approximate.

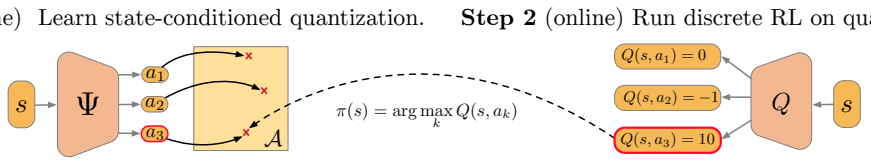
Demonstration data. Additional data consisting of transitions from an agent may be available. These demonstrations may contain the reward information or not. In the context of Imitation Learning [Pomerleau, 1991, Ng et al., 1999, 2000, Ziebart et al., 2008], the assumption is that the agent generating the demonstration data is near-optimal and that demonstration rewards are not provided. The objective is then to match the distribution of the agent with the one of the expert. In the context of Reinforcement Learning with demonstrations (RLfD) [Hester et al., 2018, Vecerik et al., 2017], demonstration rewards are provided. They are typically used in the form of auxiliary objectives together with a standard learning agent whose goal is to maximize the environment reward. In the context of Reinforcement Learning with play [Lynch et al., 2020], demonstration rewards are not provided as play data is typically not task-specific.

Demonstration data can come from various sources, although a common assumption is that it is generated by a single, unimodal Markovian policy. However, most of available data comes from agents that do not fulfill this condition. In particular, for human data, and even more so when coming from several individuals, the behavior generating the episodes may not be unimodal nor Markovian.

7.3 Method

In this section, we introduce the AQuaDem framework and a practical neural network implementation together with an accompanying objective function. We provide a series of visualizations to study the

candidate actions learned with AQuaDem in gridworld experiments.



Visualization of the AQuaDem framework (offline) with a downstream algorithm (online).

7.3.1 AQuaDem: Action Quantization from Demonstrations

Our objective is to reduce a continuous control problem to a discrete action one, on which we can apply discrete-action RL methods. Using demonstrations, we wish to assign to each state $s \in \mathcal{S}$ a set of K candidate actions from $\mathcal{A}$. The resulting action space is therefore a discrete finite set of K state-conditioned vectors. In a given state $s \in \mathcal{S}$, picking action $k \in \{1, \dots, K\}$ stands for picking the k^{th} candidate action for that particular state. The AQuaDem framework refers to the discretization of the action space, and the resulting discrete action algorithms used with AQuaDem on continuous control tasks are detailed in Section 7.4. We propose to learn the discrete action space through a modified version of the Behavioral Cloning (BC) [Pomerleau, 1991] reconstruction loss that captures the multimodality of demonstrations. Indeed the typical BC implementation consists in building a deterministic mapping between states and actions $\Phi : \mathcal{S} \mapsto \mathcal{A}$. But in practice, and in particular when the demonstrator is human, the demonstrator can take multiple actions in a given state (we say that its behavior is *multimodal*) which are all good candidates for AQuaDem. We thus learn a mapping $\Psi : \mathcal{S} \mapsto \mathcal{A}^K$ from states to a set of K candidate actions and optimize a reconstruction loss based on a soft minimum between the candidate actions and the demonstrated action.

Suppose we have a dataset of expert demonstrations $\mathcal{D} = \{(s_i, a_i)\}_{1:n}$. In the continuous action setting, the vanilla BC approach consists in finding a parametrized function f_Φ that minimizes the reconstruction error between predicted actions and actions in the dataset $\mathcal{D}$. To ease notations, we will conflate the function f_Φ with its parameters Φ and simply note it $\Phi : \mathcal{S} \mapsto \mathcal{A}$. The objective is thus to solve:

$$\min_{\Phi} \mathbb{E}_{s,a\sim\mathcal{D}} \|\Phi(s) - a\|^2.$$

Instead, we propose to learn a set of K actions $\Psi_k(s)$ for each state by minimizing the following loss:

$$\min_{\Psi} \mathbb{E}_{s,a\sim\mathcal{D}} \Big[-T \log \Big(\sum_{k=1}^{K} \exp \big(\frac{-\|\Psi_k(s) - a\|^2}{T} \big) \Big) \Big], \tag{7.1}$$

where the temperature T is a hyperparameter. Equation (7.1) corresponds to minimizing a soft-minimum between the candidates actions $\Psi_1(s), \dots, \Psi_K(s)$ and the demonstrated action a. Note that with $K = 1$, this is exactly the BC loss. The larger the temperature T is, the more the loss imposes all candidate actions to be close to the demonstrated action a thus reducing to the BC loss. The lower the temperature T is, the more the loss only imposes a single candidate action to be close to the demonstrated action a. We provide empirical evidence of this phenomenon in Section 7.3.2 and provite a formal justification in Appendix 7.5.2. Equation (7.1) is also interpretable in the context of Gaussian mixture models (see

Appendix 7.5.1). The Ψ function enables us to define a new MDP where the continuous action space is replaced by a discrete action space of size K corresponding to the K action candidates returned by Ψ at each state.

7.3.2 Visualization

In this section, we analyze the actions learned through the AQuaDem framework, in a toy grid world environment. We introduce a continuous action grid world with demonstrations in Figure 7.1.

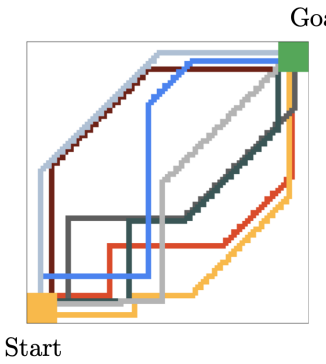


Figure 7.1: Grid world environment where the start state is in the bottom left, and the goal state is in the top right. Actions are continuous (2-dimensional), and give the direction in which the agent take a step. These steps are normalized by the environment to have fixed L2 norm. The stochastic demonstrator moves either right or up in the bottom left of the environment then moves diagonally until reaching the edge of the grid, and goes either up or right to reach the target. The demonstrations are represented in the different colors.

We define a neural network Ψ and optimize its parameters by minimizing the objective function defined in Equation (7.1) (implementation details can be found in Appendix D.4.1). We display the resulting candidate actions across the state space in Figure 7.2. As each color of the arrows depicts a single head of the Ψ network, we observe that the action candidates are *smooth*: action candidates slowly vary as the state vary, which prevents to have inconsistent action candidates in nearby states. Note that BC actions tend to be *diagonal* even in the bottom left part of the action space, where the demonstrator only takes horizontal or vertical actions. On the contrary, the action candidates learned by AQuaDem include the actions taken by the demonstrator conditioned on the states. Remark that in the case of $K = 2$, the action *right* is learned independently of the state position (middle plot in Figure 7.2) although it is only executed in a subspace of the action space. In the case of $K = 3$, actions are completely state-independent. In non-trivial tasks, the state dependence induced by the AQuaDem framework is essential, as we show in the ablation study in Appendix D.1 and in the analysis of the actions learned in a more realistic setup in Appendix 7.3.3.

Influence of the temperature. The temperature controls the degree of smoothness of the soft-minimum defined in Equation (7.1). We show that with larger temperatures, the soft-minimum converges to the average which is well represented in Figure 7.3 rightmost plot where the profile of AQuaDem's action candidates conflate with actions learned by BC. With lower temperatures, the actions taken by the demonstrator are recovered, but if the temperature is too low ($T = 0.001$), some actions that are not taken by the demonstrator might appear as candidates (blue arrows in the leftmost figure). This occurs because the soft minimum converges to a hard minimum with lower temperatures meaning that as long as one candidate is close enough to the demonstrated action, the other candidates can be arbitrarily far off. In this work, we treat the temperature as a hyperparameter, although a natural direction for future work is to aggregate actions learned for different temperatures.

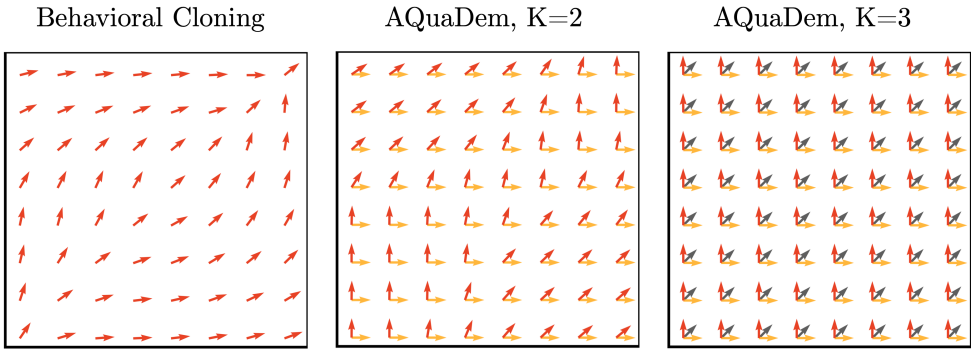


Figure 7.2: Visualisation of the action mapping learned by BC and the candidate actions learned with AQuaDem for $K = 2$ and $K = 3$ and $T = 0.01$. Each color represents a head of the Ψ network.

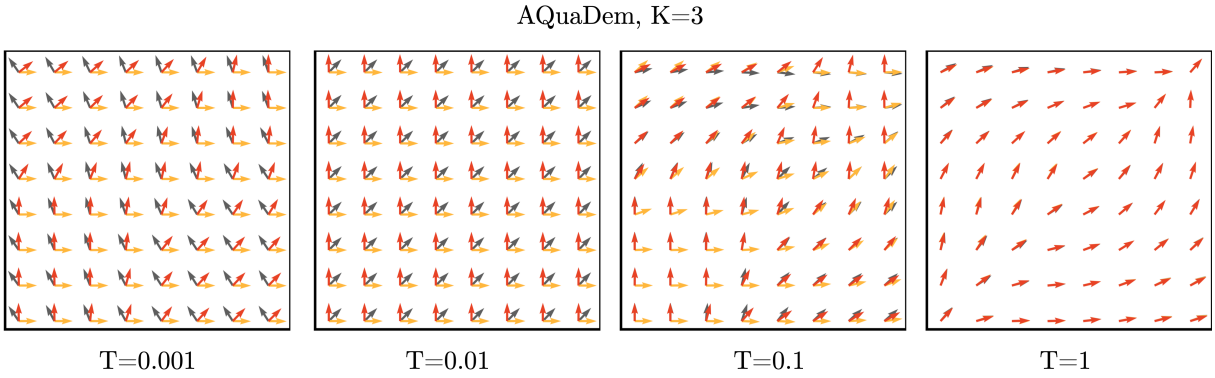


Figure 7.3: Influence of the temperature on resulting candidate actions learned with AQuaDem.

7.3.3 Action visualization in a high-dimensional environment

For the Door environment (see Figure D.2) we represent the actions candidates learned using the AQuaDem framework with videos that can be found on the website (1st video on https://google-research.github.io/aquadem/). As the action space is of high dimensionality, we choose to represent each action dimension on the x-axis, and the value for each dimension on the y-axis. We connect the dots on the x-axis to facilitate the visualization through time. We replay a trajectory from the human demonstrations and show at each step the 10 actions proposed by the AQuaDem network, and the action actually taken by the human demonstrator. Each action candidate has a color consistent across time (meaning that the blue action always correspond to the same head of the Ψ network). Interestingly, the video shows that actions are very state dependent (except some default 0-action) and evolve smoothly through time.

7.3.4 Discussion

On *losing* the optimal policy. In any form of discretization scheme, the resulting class of policies might not include the optimal policy of the original MDP. In the case of AQuaDem, this mainly depends on the quality of the demonstrations. For standard continuous control methods, the parametrization of the policy also constrains the space of possible policies, potentially not including the optimal one. This is a lesser problem since policies tend to be represented with functions with universal approximation capabilities. Nevertheless, for most continuous control methods, the policy improvement step is approximate, while in the case of AQuaDem it is exact, since it amounts to selecting the argmax of the Q-values.

On the multimodality of demonstrations. The multimodality of demonstrations enables us to define multiple plausible actions for the agent to take in a given state, guided by the priors of the demonstrations. We argue that the assumption of multimodality of the demonstrator should actually be systematic [Mandlekar et al., 2021]. Indeed, the demonstrator can be *e.g.* non-Markovian, optimizing for something different than a reward function like curiosity [Barto et al., 2013], or they can be in the process of learning how to interact with the environment. When demonstrations are gathered from multiple demonstrators, this naturally leads to multiple modalities in the demonstrations. And even in the case where the demonstrator is optimal, multiple actions might be equally good (*e.g.* in navigation tasks). Finally, the demonstrator can interact with an environment without any task specific intent, which we refer to as *play* [Lynch et al., 2020] and also induces a multimodal behavior.

7.4 Experiments

In this section, we evaluate the AQuaDem framework on three different downstream tasks setups: RL with demonstrations, RL with play data and Imitation Learning. For all experiments, we detail the networks architectures, hyperparameters search, and training procedures in the Appendix. We also provide results for Offline Reinforcement Learning in the Appendix D.3. We provide videos of all the agents trained on the website.

7.4.1 Reinforcement Learning from Demonstrations

Setup. In the Reinforcement Learning from Demonstrations setup (RLfD), the environment of interest comes with a reward function and demonstrations (which include the reward), and the goal is to learn a

policy that maximizes the expected return. This setup is particularly interesting for sparse reward tasks, where the reward function is easy to define (say reaching a goal state) and where RL methods typically fail because the exploration problem is too hard. We consider the Adroit tasks [Rajeswaran et al., 2017] represented in Figure D.2, for which human demonstrations are available (25 episodes acquired using a virtual reality system). These environments come with a dense reward function that we replace with the following **sparse reward**: 1 if the goal is achieved, 0 otherwise.

Algorithm & baselines. The algorithm we propose is a two-fold training procedure: **1)** we learn a discretization of the action space in a fully offline fashion using the AQuaDem framework from human demonstrations; **2)** we train a discrete action deep RL algorithm on top of this this discretization. We refer to this algorithm as AQuaDQN. The RL algorithm considered is Munchausen DQN [Vieillard et al., 2020] as it is the state of the art on the Atari benchmark [Bellemare et al., 2013] (although we use the non-distributional version of it which simply amounts to DQN [Mnih et al., 2013] with a regularization term). To make as much use of the demonstrations as possible, we maintain two replay buffers: one containing interactions with the environment, the other containing the demonstrations that we sample using a fixed ratio similarly to DQfD [Hester et al., 2018], although we do not use the additional recipes of DQfD (multiple n-step evaluation of the bootstrapped estimate of Q, BC regularization term) for the sake of simplicity. When sampling demonstrations, the actions are discretized by taking the closest AQuaDem action candidate (using the Euclidean norm). We consider SAC and SAC from demonstrations (SACfD) –a modified version of SAC where demonstrations are added to the replay buffer [Vecerik et al., 2017]– as baselines against the proposed method. We do not include naive discretization baselines here, as the dimension of the action space is at least 24, which would lead to a $2^{24} \simeq 16$M actions with a binary discretization scheme, which is prohibitive without additional assumptions on the structure of the action-value function.

Evaluation & results. We train the different methods on 1M environment interactions on 10 seeds for the chosen hyperparameters (a single set of hyperameters for all tasks) and evaluate the agents every 50k environment interactions (without exploration noise) on 30 episodes. An episode is considered a success if the goal is achieved during the episode. The AQuaDem discretization is trained offline using 50k gradient steps on batches of size 256. The number of actions considered were $10, 15, 20$ and we found 10 to be performing the best. Figure 7.4 shows the AQuaDem loss through the training procedure of the discretization step, and the Figure 7.5 shows the returns of the trained agents as well as their success rate. On Door, Pen, and Hammer, the AQuaDQN agent reaches high success rate, largely outperforming SACfD in terms of success and sample efficiency.

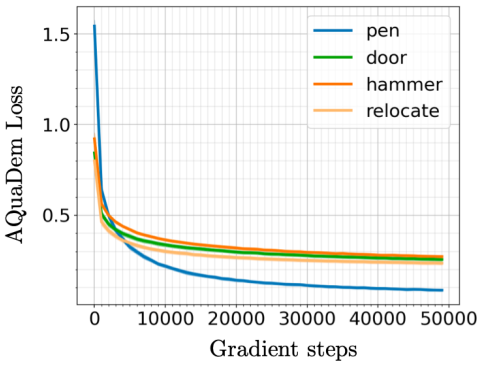


Figure 7.4: AQuaDem discretization loss.

On Relocate, all methods reach poor results (although AQuaDQN slightly outperforms the baselines). The task requires a larger degree of generalisation than the other three since the goal state and the initial ball position are changing at each episode. We show in Figure 7.6 that when tuned uniquely on the Relocate environment and with more environment interactions, AQuaDQN manages to reach a 50%

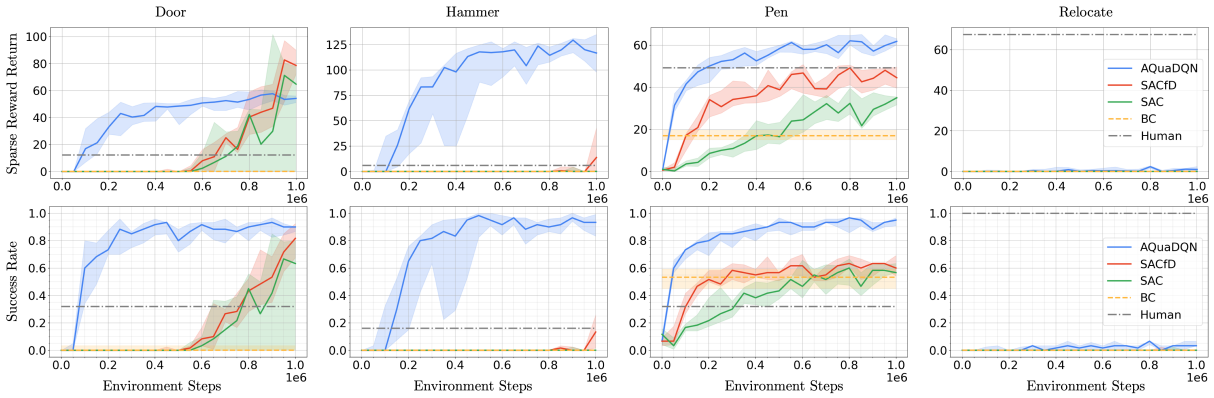


Figure 7.5: Performance of AQuaDQN against SAC and SACfD baselines. Agents are evaluated every 50k environment steps over 30 episodes. We represent the median performance in terms of success rate (bottom) and returns (top) as well as the interquartile range over 10 seeds.

success rate where other methods still fail. Notice that on the Door environment, the SAC and SACfD agents outperform the AQuaDQN agent in terms of final return (but not in term of success rate). The behavior of these agents are however different from the demonstrator since they consist in slapping the handle and abruptly pulling it back. We provide videos of all resulting agents on the website (one episode for each seed which is not cherry picked) to demonstrate that AQuaDQN consistently learns a behavior that is qualitatively closer to the demonstrator.

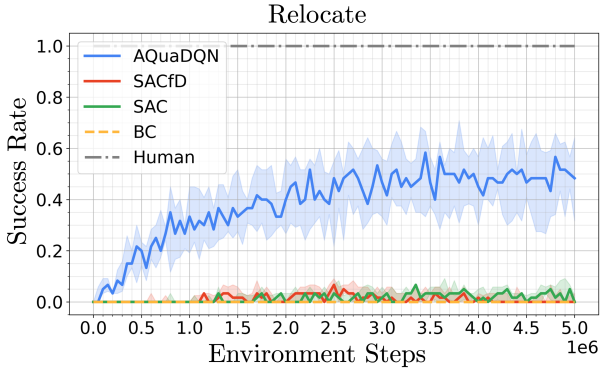


Figure 7.6: Performance of AQuaDQN against SAC and SACfD baselines when all are tuned on the Relocate environment. We represent the median performance in terms of success rate as well as the interquartile range over 10 seeds.

7.4.2 Imitation Learning

Setup. In Imitation Learning, the task is not specified by the reward function but by the demonstrations themselves. The goal is to mimic the demonstrated behavior. There is no reward function and the notion of success is ill-defined [Hussenot et al., 2021b]. A number of existing works [Ho and Ermon, 2016, Ghasemipour et al., 2020, Dadashi et al., 2021a] cast the problem into matching the state distributions

of the agent and of the expert. Imitation Learning is of particular interest when designing a satisfying reward function –one that would lead the desired behavior to be the only optimal policy– is harder than directly demonstrating this behavior. In this setup, there is no reward provided, not in the environment interactions nor in the demonstrations. We again consider the Adroit environments and the human demonstrations which consist of 25 episodes acquired via a virtual reality system.

Algorithm & baselines. Again, the algorithm we propose has two stages. **1)** We learn –fully offline– a discretization of the action space using AQuaDem. **2)** We train a discrete action version of the GAIL algorithm [Ho and Ermon, 2016] in the discretized environment. More precisely, we interleave the training of a discriminator between demonstrations and agent experiences, and the training of a Munchausen DQN agent that maximizes the confusion of this discriminator. The Munchausen DQN takes one of the candidates actions given by AQuaDem. We call this algorithm AQuaGAIL. As a baseline, we consider the GAIL algorithm with a SAC [Haarnoja et al., 2018a] agent directly maximizing the confusion of the discriminator. This results in a very similar algorithm as the one proposed by Kostrikov et al. [2019]. We also include the results of BC [Pomerleau, 1991].

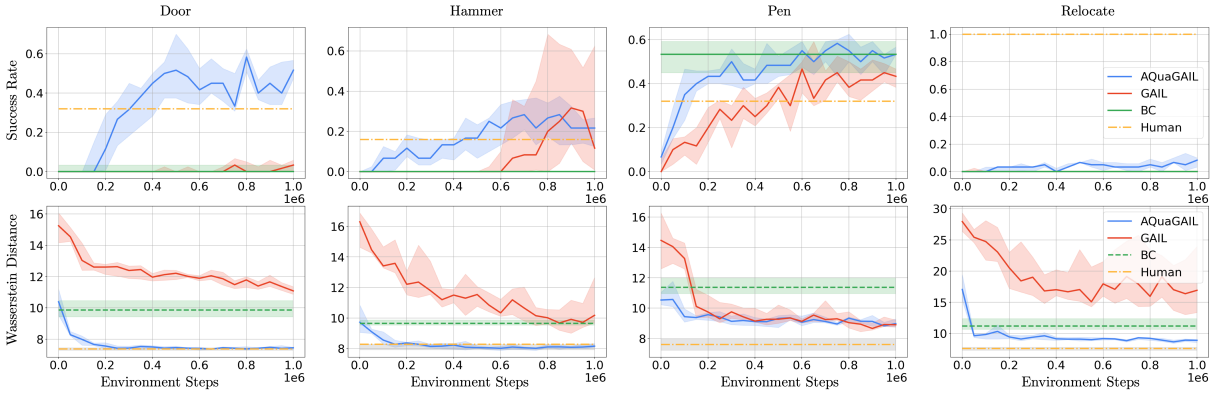


Figure 7.7: Performance of AQuaGAIL against GAIL and BC baselines. Agents are evaluated every 50k environment steps over 30 episodes. We represent the median success rate (top row) on the task as well as the Wasserstein distance (bottom row) of the agent's state distribution to the expert's state distribution as well as the interquartile range over 10 seeds.

Evaluation & results. We train AQuaGAIL and GAIL for 1M environment interactions on 10 seeds for the selected hyperparameters (a single set for all tasks). BC is trained for 60k gradient steps with batch size 256. We evaluate the agents every 50k environment steps during training (without exploration noise) on 30 episodes. The AQuaDem discretization is trained offline using 50k gradient steps on batches of size 256. The results are provided in Figure 7.7. Evaluating imitation learning algorithms has to be done carefully as the goal to "mimic a behavior" is ill-defined. Here, we provide the results according to two metrics. On top, the success rate is defined in Section 7.4.1. Notice that the human demonstrations do not have a success score of 1 on every task. We see that, except for Relocate, which is a hard task to solve with only 25 human demonstrations due to the necessity to generalize to new positions of the ball and the target, AQuaGAIL solves the tasks as successfully as the humans, outperforming GAIL and BC. Notice that our results corroborate results presented in Chapter 3 [Orsini et al., 2021] that showed poor

performance of GAIL on human demonstrations after 1M steps. The second metric we provide, on the bottom, is the Wasserstein distance between the state distribution of the demonstrations and the one of the agent. We compute it using the POT library [Flamary et al., 2021] and use the Sinkhorn distance, a regularized version of the Wasserstein distance, as it is faster to compute. The "human" Wasserstein distance score is computed by randomly taking 5 episodes out of the 25 human demonstrations and compute the Wasserstein distance to the remaining 20. We repeat this procedure 100 times and plot the median (and the interquartile range) of the obtained values. Remark that AQuaGAIL is able to get much closer behavior to the human than BC and GAIL on all four environments in terms of Wasserstein distance. This supports that AQuaDem leads to policies much closer to the demonstrator. We provide videos of the trained agents as an additional qualitative empirical evidence to support this claim.

7.4.3 Reinforcement Learning with play data

Setup. The Reinforcement Learning with play data is an under-explored yet natural setup [Gupta et al., 2019]. In this setup, the environment of interest has multiple tasks, a shared observation and action space for each task, and a reward function specific to each of the tasks. We also assume that we have access to *play data*, introduced by Lynch et al. [2020], which consists in episodes from a human demonstrator interacting with an environment with the sole intention to play with it. The goal is to learn an optimal policy for each of the tasks. We consider the Robodesk tasks [Kannan et al., 2021] shown in Figure D.2, for which we acquired play data. We expand on the environment as well as the data collection procedure in the Appendix D.4.2.

Algorithm & baselines. Similarly to the RLfD setting, we propose a two-fold training procedure: **1)** we learn a discretization of the action space in a fully offline fashion using the AQuaDem framework on the play data, **2)** we train a discrete action deep RL algorithm using this discretization on each tasks. We refer to this algorithm as AQuaPlay. Unlike the RLfD setting, the demonstrations do not include any task specific reward nor goal labels meaning that we cannot incorporate the demonstration episodes in the replay buffer nor use some form of goal-conditioned BC. We use SAC as a baseline, which is trained to optimize task specific rewards. Since the action space dimensionality is fairly low (5-dimensional), we can include naive uniform discretization baselines that we refer to as "bang-bang" [Bushaw, 1953]. The original "bang-bang" controller (BB-2) is based on the extrema of the action space, we also provide a uniform discretization scheme based on 3 and 5 bins per action dimension, that we refer to as BB-3 and BB-5 respectively.

Evaluation & results. We train the different methods on 1M environment interactions on 10 seeds for the chosen hyperparameters (a single set of hyperameters for all tasks) and evaluate the agents every 50k environment interactions (without exploration noise) on 30 episodes. The AQuaDem discretization is trained offline on play data using 50k gradient steps on batches of size 256. The number of actions considered were $10, 20, 30, 40$ and we found 30 to be performing the best. It is interesting to notice that it is higher than for the previous setups. It aligns with the intuition that with play data, several behaviors needs to be modelled. The results are provided in Figure 7.8. The AQuaPlay agent consistently outperforms SAC in this setup. Interestingly, the performance of the BB agent decreases with the discretization granularity, well exemplifying the curse of dimensionality of the method. In fact, BB with a binary discretization (BB-2) is competitive with AQuaPlay, which validates that discrete action RL algorithms are well performing if the discrete actions are sufficient to solve the task. Note however that

the Robodesk environment is a relatively low-dimensional action environment, making it possible to have BB as a baseline, which is not the case of *e.g.* Adroit where the action space is high-dimensional.

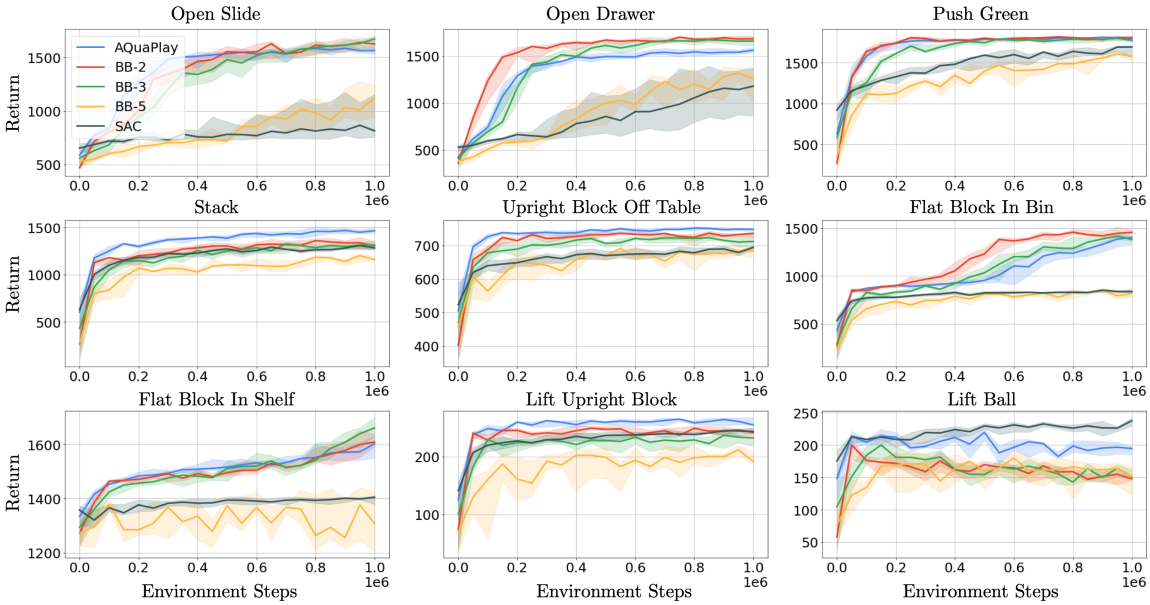


Figure 7.8: Performance of AQuaPlay against SAC and "bang-bang" baselines. Agents are evaluated every 50k environment steps over 30 episodes. We represent the median return as well as the interquartile range over 10 seeds.

7.5 Theoretical aspects of the discretization

In this section, we discuss two theoretical aspects of the discretization: the connection of the proposed loss to gaussian mixture models and its asympotic behavior with respect to the number of candidate actions.

7.5.1 Connection to Gaussian mixture models

The BC loss can be interpreted as a maximum likelihood objective under the assumption that the demonstrator data comes from a Gaussian distribution. Similarly to Mixture density networks [Bishop, 1994], we propose to replace the Gaussian distribution by a mixture of Gaussian distributions. Suppose we represent the probability density of an action conditioned on a state by a mixture of K Gaussian kernels: $p(a|s) = \sum_{k=1}^{K} \alpha_k(s) d_k(a|s)$, where $\alpha_k(s)$ is the mixing coefficient (that can be interpreted as a state conditioned prior probability), and $d_k(a|s)$ is the conditional density of the target a. Now assuming that the K kernels are centered on $\Psi_k(s)_{k=1:K}$ and have fixed covariance $\sigma^2 \mathbb{1}$ where σ is a hyperparameter, we

can write the log-likelihood of the demonstrations data $\mathcal{D}$ as:

$$\begin{aligned}\mathcal{LL}(\mathcal{D}) &= \sum_{s,a\in\mathcal{D}} \log(p(s)p(a|s)) = \sum_{s,a\in\mathcal{D}} \log p(s) + \log\big(\sum_{k=1}^{K} \alpha_k(s) d_k(a|s)\big) \\ &= \sum_{s,a\in\mathcal{D}} \log p(s) + \log\big(C \sum_{k=1}^{K} \alpha_k(s) \exp\big(-\frac{\|\Psi_k(s)-a|^2}{\sigma^2}\big)\big).\end{aligned}$$

Therefore minimizing the negative log likelihood reduces to minimizing:

$$\sum_{s,a\in\mathcal{D}} -\log\big(\sum_{k=1}^{K} \alpha_k(s) \exp\big(-\frac{\|\Psi_k(s)-a\|^2}{\sigma^2}\big)\big).$$

We propose to use a uniform prior $\alpha_k(s) = \frac{1}{K}$ when learning the locations of the centroids, which leads exactly to Equation (7.1) where the variance σ^2 is the temperature T. Note that we initially learned the state conditioned prior $\alpha_k(s)$, but we found no empirical evidence that it may be used to improve the performance of the downstream algorithms defined in Section 7.4.

7.5.2 Asymptotic behavior of the AQuaDem loss

For lighter notations, we write $x_k = ||\Psi_k(s) - a||^2$ and $x = (x_1, ..., x_K)$. For a single state-action pair (the empirical expectation being not relevant for studying the effect of the temperature), the AQuaDem loss can be rewritten:

$$J(\Psi) = -T \log \sum_{k=1}^{K} \exp(-\frac{x_k}{T}) = -T \log(\frac{1}{K} \sum_{k=1}^{K} \exp(-\frac{x_k}{T})) - \log K.$$

Let's define $f_T(x) = -T \log(\frac{1}{K} \sum_{k=1}^{K} \exp(-\frac{x_k}{T}))$. The function f_T is the same as the loss up to a constant term that does not change the solution of the optimization problem. Therefore, we can study this function for the behavior of the loss with respect to the temperature.

Now, denoting $x_m = \min_k x_k$, we'll first study the behavior for low temperature.

$$\begin{aligned}f_T(x) &= T \log K - T \log(\sum_{k=1}^{K} \exp(-\frac{x_k}{T})) \\ &= T \log K - T \log(\exp(\frac{-x_m}{T}) \sum_{k=1}^{K} \exp(-\frac{x_k - x_m}{T})) \\ &= T \log K + x_m - T \log(1 + \sum_{k=1, k\neq m}^{K} \exp(-\frac{x_k - x_m}{T})) \xrightarrow[T\to 0]{} x_m.\end{aligned}$$

Therefore, when the temperature goes to zero, f_T behaves as the minimum.

For large temperatures, we have, using Taylor expansions:

$$f_T(x) = -T\log(\sum_{k=1}^{K}\frac{1}{K}\exp(-\frac{x_k}{T})) = -T\log(\sum_{k=1}^{K}\frac{1}{K}(1-\frac{x_k}{T}+o(\frac{x_k}{T})))$$
$$= -T\log(1-\frac{1}{K}\sum_{k=1}^{K}\frac{x_k}{T}+o(\frac{1}{T})) = \frac{T}{K}\sum_{k=1}^{K}\frac{x_k}{T}+O(\frac{1}{T}) \xrightarrow[T\to\infty]{} \frac{1}{K}\sum_{k=1}^{K}x_k.$$

Hence, when the temperature goes to infinite, f_T behaves as the average.

7.6 Related work

Continuous action discretization. The discretization of continuous action spaces has been introduced in control problems by Bushaw [1953] with the "bang-bang" controller [Bellman et al., 1956]. This naive discretization is problematic in high-dimensional action spaces, as the number of actions grows exponentially with the action dimensionality. To mitigate this phenomenon, a possible strategy is to assume that action dimensions are independent [Tavakoli et al., 2018, Andrychowicz et al., 2020b, Vieillard et al., 2021, Tang and Agrawal, 2020], or to assume or learn a causal dependence between them [Metz et al., 2017, Tessler et al., 2019, Sakryukin et al., 2020, Tavakoli et al., 2021]. The AQuaDem framework circumvents the curse of dimensionality as the discretization is based on the demonstrations and hence is dependent on the multimodality of the actions picked by the demonstrator rather than the dimensionality of the action space. Recently, Seyde et al. [2021] replaced the Gaussian distribution on top of a number of continuous control methods, including SAC, by a Bernoulli distribution to sample extremal actions (thus still relying on a sampling-based optimisation procedure due to the high dimensionality of the action space), and show that performance remains similar on the DM control benchmark [Tassa et al., 2020]. The AQuaDem framework does not favor extremal actions (which could be suboptimal) but the actions selected by the demonstrator instead. Close to our setup is the case where the action space is both discrete and continuous [Neunert et al., 2020] or the action space is discrete and large [Dulac-Arnold et al., 2015]. Those setups are interesting directions for extending AQuaDem.

Q-learning in continuous action spaces. Policy-based methods consist in solving continuous or discrete MDPs based on maximizing the expected return over the parameters of a family of policies. If the return is estimated through Monte Carlo rollouts, this leads to algorithms that are typically sample-inefficient and difficult to train in high-dimensional action spaces [Williams, 1992, Schulman et al., 2015a, 2017]. As a result, a number of policy-based methods inspired from the policy gradient theorem [Sutton et al., 2000], aim at maximizing the return using an approximate version of the Q-value thus making them more sample-efficient. One common architecture is to parameterize a Q-value, which is estimated by enforcing Bellman consistency, and define a policy using an optimization procedure of the parametrized Q-value. Typical strategies to solve the Q-value maximization include enforcing the Q-value to be concave [Gu et al., 2016, Amos et al., 2017] making it easy to optimize through *e.g.* gradient ascent, to use a black box optimization method [Kalashnikov et al., 2018, Simmons-Edler et al., 2019, Lim et al., 2018], to solve a mixed integer programming problem [Ryu et al., 2020], or to follow a biased estimate of the policy gradient based on the approximate Q-value [Konda and Tsitsiklis, 2000, Lillicrap et al., 2016, Haarnoja et al., 2018a, Fujimoto et al., 2018]. Recently, Asadi et al. [2021] proposed to use a network that outputs actions together with their associated Q-values, tuned for each of the tasks at hand, on

low-dimensional action spaces. Note that maximizing the approximate Q-value is a key problem that does not appear in discrete action space environments, thus justifying the interest of the AQuaDem framework.

Hierarchical Imitation Learning. A number of approaches have explored the learning of *primitives* or *options* from demonstrations together with a high-level controller that is either learned from demonstrations [Kroemer et al., 2015, Krishnan et al., 2017, Le et al., 2018, Ding et al., 2019, Lynch et al., 2020], or learned from interactions with the environment [Manschitz et al., 2015, Kipf et al., 2019, Shankar et al., 2019], or hand specified [Pastor et al., 2009, Fox et al., 2019]. AQuaDem can be loosely interpreted as a two-level procedure as well, where the primitives (action discretization step) are learned fully offline, however there is no concept of goal nor temporally extended actions.

Modeling multimodal demonstrations. A number of works have modeled the demonstrator data using multimodal architectures. For example, Chernova and Veloso [2007], Calinon and Billard [2007] introduce Gaussian mixture models in their modeling of the demonstrator data. More recently, Rahmatizadeh et al. [2018] use Mixture density networks together with a recurrent neural network to model the temporal correlation of actions as well as their multimodality. [Yu et al., 2018] also uses Mixture density networks to meta-learn a policy from demonstrations for one-shot adaptation. Another recent line of works has considered the problem of modeling demonstrations using an energy-based model, which is well adapted for multimodalities [Jarrett et al., 2020, Florence et al., 2021]. Singh et al. [2020] also exploit the demonstrations prior for downstream tasks by learning a prior using a state-conditioned action generative model coupled with a continuous action algorithm. This is different from AQuaDem that exploits the demonstrations prior to learn a discrete action space in order to use discrete action RL algorithms.

7.7 Discussion

With the AQuaDem paradigm, we provide a simple yet powerful method that enables to use discrete-action deep RL methods on continuous control tasks using demonstrations, thus escaping the complexity or curse of dimensionality of existing discretization methods. We showed in three different setups that it provides substantial gains in sample efficiency and performance and that it leads to qualitatively better agents, as enlightened by the videos provided on the website.

There are a number of different research avenues opened by AQuaDem. Other discrete action specific methods could be leveraged in a similar way in the context of continuous control: count-based exploration [Tang et al., 2017] or planning [Browne et al., 2012]. Similarly a number of methods in Imitation Learning [Brantley et al., 2020, Wang et al., 2019] or in offline RL [Fujimoto and Gu, 2021, Wu et al., 2019] are evaluated on continuous control tasks and are based on Behavioral Cloning regularization which could be refined using the same type of multioutput architecture used in this work. Another possible direction for the AQuaDem framework is to be analyzed in the light of risk-MDPs as the constraint of the action space arguably reduces a notion of risk when acting in this environment. Finally, as the gain of sample efficiency is clear in different experimental settings, we believe that the AQuaDem framework could be an interesting avenue for learning controllers on physical systems.

Conclusion

Learning from demonstrations is a promising avenue for building more efficient algorithms, more closely aligned with human expectations and preferences.

In chapter 3, we designed an extensive empirical study on adversarial imitation learning. Some of our findings confirmed common practices, while others were more surprising or even contradicting prior work. We showed the importance of choosing an off-policy algorithm for sample efficiency. We showed that classical regularization techniques from supervised learning, like dropout, are as good for regularizing the discriminator as techniques that were specifically designed for this framework. We also highlighted the difference between imitating human and synthetic demonstrations. In chapter 4, we went beyond adversarial imitation learning to bypass its brittleness and its high number of hyperparameters. We designed an algorithm that has only two of them, by deriving an upper-bound of the Wasserstein-1 distance. We trained an agent to minimize this upper bound of the distance between the agent's state-action distribution and the distribution of demonstrations. We showed the performance of this agent in the *very low data regime*, where we trained Humanoid-v2 to high performance with a single sub-sampled trajectory as demonstration. We then studied in chapter 5 how to select hyperparameters in the context of imitation, without access to the underlying reward function. We showed that different approaches lead to successful behaviors, through selecting based on proxy-metrics like optimal-transport based ones, or through transfer.

We showed in chapter 6 how to extract intrinsic motivation from the demonstrations and transfer it to a learning agent. We demonstrated that this motivation, implemented as an intrinsic reward, was carrying knowledge on how to explore the environment. Finally, in chapter 7, we designed an algorithm that carries the demonstrator's incentives through a learned, state-dependent, discretization of the state space. This allowed to use the more sample efficient discrete reinforcement learning agents on continuous control problems and to carry human priors from the demonstrations to the agent. We showed that the resulting agents were extremely competitive on a wide variety of robotics tasks by leveraging human demonstrations that either solve the tasks or just play around with the environment, with access to a sparse reward or not. We also demonstrated that the resulting behaviors are "visually" much closer to human way of solving the tasks, demonstrating that human incentives were successfully transferred to the agent without implementing them in a closed-form in the reward function.

Along these works, we open-sourced 1) the generic adversarial imitation learning agent built for chapter 3, as well as the raw results of the study and a colab to study them, 2) the imitation learning agent designed in chapter 4, 3) the agent designed in chapter 7, 4) RLDS (chapter 2.6), the infrastructure that made chapters 3,4,5,7 possible thanks to human data collection and common data storage.

Great challenges remain to be met to properly transfer incentives from humans to artificial agents. But numerous works are proposing novel ideas to incorporate human evaluation in their learning process. In

addition to demonstrations, this feedback can take many forms. For example, human can rate behaviors, or express pairwise preferences explicitly. These ideas have already started to be used in several domains like robotics [Christiano et al., 2017] or language modelling [Ziegler et al., 2019, Hilton et al., 2016, Menick et al., 2022]. Expressing this feedback in natural language appears like a key step in making robot learning a day-to-day reality. Instruction-following reinforcement learning agents have begun to emerge [Lynch et al., 2020]. They showed that they could correct imperfect behavior at inference time using natural language instructions, however, actually learning from human language feedback remains an open challenge.

Learning from both human demonstrations and human preferences can unlock immense prospects. One could better understand how to learn complex behaviors defined both by task and *style* objectives. In the context of video games, this could lead to non-player characters that are extremely human-like, and to more immersive experiences. This could mean having a soccer video game where teams, in addition to scoring goals (to maximize their reward) actually reproduce the specific style of the real team, with lightning speed counter attacks or possession, construction based behaviors. Yet, to build such examples, a much deeper understanding of multi-agent reinforcement learning and how to cope with it in combination with learning from demonstrations would also be required.

AlphaGo [Silver et al., 2017], AlphaZero [Silver et al., 2018] or AlphaStar [Vinyals et al., 2019] have shown the immense capabilities of reinforcement learning agents in the context of games. Once trained, these agents have beaten several world champions or state-of-the-art algorithms, pushing the boundaries of how Chess, Go, Shogi or Competitive video games like StarCraft II are played. Yet learning from demonstrations and preferences could unlock new possibilities: agents that adapt themselves to the level of their human opponent, agents that are built to make their human opponent progress and not just beat them, agents that have specific styles or properties. Overall, it has the potential of building games that are always fun to play, always at the preferred level of difficulty and with more creative, less robotic and more human-like adversaries and non-player characters.

Three main obstacles remain to be tackled in order to successfully build scalable agents that learn from demonstrations and human preferences.

1) We need large, safe infrastructure for human-data collection. If the RLDS Creator [Ramos et al., 2021] we designed and open-sourced allowed for human data collection, it is not yet usable to give feedback to a learning agent in a real-time fashion.

2) We need research on how to make the most of several types of feedback. Agents are usually trained on a reward function, or from demonstrations, or from ratings, or from preferences but rarely combine several of these, and basically never combining all of them. Using natural language to provide this feedback seems like an extremely promising avenue for having complex, subtle feedback from humans, but remain under-explored.

3) We need research on how to keep these agents safe and fair as they start to learn from several forms of feedback. This area is still to be explored. While, along this thesis, we looked into means of making agents more "human-like", the human feedback may also bring about unexpected biases and potential flaws. Preventing this will be key to develop agents that are deployable in the real-world.

Acknowledgements

Many of the people I will mention deserve more than a *thank you* for they fully belong to the story of these past three years. For the sake of the brevity, I will not try to be exhaustive about the ingredients of this great journey that you all offered me.

First and foremost, thanks Olivier, thanks for believing I could actually be a researcher and for the tedious work you invested in trying not to be wrong. Needless to say, I learned throughout the past years that you are almost never wrong, you are just not yet right. Thanks for being this relentless Cassandra, that fun Hannibal that loves it when a plan comes together (and not the one that wishes we could chat longer but is having an old friend for dinner).

Thanks Philippe for making this PhD possible, and helping me throughout the journey.

A huge thanks Matthieu for always taking the time to repeat, explain, detail and repeat again. Thanks for making me feel like my work was interesting, your consideration was more than appreciated and your daily feedback more than valuable. I will never forget your mentorship.

I am honored and thankful Professors Pierre-Yves Oudeyer and Emmanuel Rachelson accepted to review this thesis. Thanks for taking the time to read it for the great feedback you gave me.

Thank you so much Olivier B. for your trust and guidance. Working with you is not easy, and you know that's a compliment. It's a constant lesson to see you excited for the new challenges ahead. Thanks to the other Olivier B. for strolling around the same Brazilian deserts as me.

A very special thanks to Robert, to whom I owe so much. Thanks for always sharing your thoughts and ideas, and always making room for mine. Sorry again for intoxicating you at my chinese bouiboui.

Big-up to my men Johan and Nino for being the Adecco PhD dream team. It's not possible to thank you enough without doubling the number of page of this thesis.

Thanks a lot Neil for paving the way and being such an example.

Thanks Mathieu S. for the long chats and the countless advice.

I'd really like to express my gratitude to Nikola for mentoring me with such kindness and showing me week after week how to transform intuitions into explanations.

A big thanks to Damien, Sabela, Raphaël, Sertan, Olivier T., Anton, Danila, Manu, Marcin, Matt, Bobak and Piotr for the patience which you have shown me over the years. Working with you is a great pleasure.

Merci à mes parents pour leur soutien indéfectible. Merci de m'avoir accueilli pendant le confinement et de m'avoir permis de travailler à cette thèse dans des conditions de luxe absolu. Mais surtout, merci d'avoir tout fait pour me rendre independant tout en acceptant quand je refuse de l'être et ainsi de me recevoir (moi et tous les potes) chaque fois avec autant de plaisir et de generosité.

Un grand merci, Joe, de toujours me ramenener aux choses qui comptent. En attendant d'autres partages de magret.

Mille mercis à mes deux docteurs préférés, Dominique Dreyfus et Francis Bordat pour les relectures, les longs échanges, les subtilités linguistiques des deux langues que vous avez grandement contribué à m'apprendre. L'une m'a bien servi dans ce manuscrit et l'autre m'a ouvert sur un monde nouveau! Merci pour cela et pour tout le reste.

Queria agradecer as minhas mães brasileiras, Crica e Licó, por me mostrarem que a pesquisa tem tudo a ver com o mundo real e sensível.

Paulo, Nati, Marília, Muito obrigado pela amizade inesgotável.

À Pipa, un immense merci, tu m'as ammené à mon premier jour de thèse il y a presque quatre ans et n'a pas economisé une once d'humour et de tendresse depuis. Merci pour tout.

À mes gars, Adri, Goga, Hippo, Jerem, José, Karlito, Martin, Matt, Masty, Rub, Thib, merci ne serait pas suffisant. J'ai peu de plaisirs dans la vie plus grands que de vous côtoyer.

À Matéo, Philémon et mes volleyeurs de ballek, un grand merci. Je vous promets de ne jamais oublier ni nos dîners, ni nos débats, ni nos voyages, ni d'annoncer 90 sans-atout.

À Straight Outta Vaison, merci d'exister.

À mon second frère, Tom, merci pour ton sens de l'amitié indéfectible. À Allan, Louis, Oriane, Valentine, Victoire, merci pour tous les moments mémorables passés et j'espère à venir.

À Obo, je ne te remercie pas d'etre si loin, mais je suis très heureux d'arriver à faire en sorte que ca ne change presque rien.

À Paul, avec qui j'apprends une nouvelle chose chaque fois, merci, c'est tout simplement inestimable.

À Marion, un grand merci pour les pauses cafés et tout le reste.

À Marguerite, avec qui chaque bout de vie est un bonheur, j'en espère des centaines d'autres, merci pour tout.

À Nico qui m'a montré qu'il n'est jamais trop tard, merci encore.

À ma cousine Elisa, qui part souvent mais n'est jamais très loin, merci pour ton amitié sans faille.

Et finalement, à ceux qui ont supporté mes questions incessantes et qui, par leur abnégation, ont grandement contribué à me transmettre le goût des sciences, M. Skrzypek, Mme. Champiot, M. Bonnand, M. Courtillas, Mme. Saïbi, merci pour votre énergie.

Bibliography

Pieter Abbeel. *Apprenticeship learning and reinforcement learning with application to robotic control*. Stanford University, 2008. 18

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, 2004. 19, 42, 71

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018. 17

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, 2017. 98

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International conference on learning representations*, 2020a. 13, 26, 29, 33, 39, 55, 66

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020b. 26, 42, 71, 86, 98

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. 26, 55

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 2017. 19, 42

Kavosh Asadi, Neev Parikh, Ronald E Parr, George D Konidaris, and Michael L Littman. Deep radial-basis value functions for continuous control. In *AAAI Conference on Artificial Intelligence*, 2021. 98

Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*, 2018. 18, 52

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 182

JA Bagnell, Joel Chestnutt, David M Bradley, and Nathan D Ratliff. Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems*, 2007. 18, 42

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018. 32, 47, 125, 182

Andrew Barto, Marco Mirolli, and Gianluca Baldassarre. Novelty or surprise? *Frontiers in psychology*, 2013. 91

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in neural information processing systems*, pages 1471–1479, 2016. 71, 83

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. 17, 20, 66, 92

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017. 17, 125, 182

R. Bellman. A markovian decision process. *Indiana University Mathematics Journal*, 1957. 16, 87

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. 9

Richard Bellman, Irving Glicksberg, and Oliver Gross. On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 1956. 98

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012. 66

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 17, 25, 55, 86

Kent C Berridge. The debate over dopamine's role in reward: the case for incentive salience. *Psychopharmacology*, 191(3):391–431, 2007. 9

Glen Berseth, Daniel Geng, Coline Devin, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. Smirl: Surprise minimizing rl in dynamic environments. *arXiv preprint arXiv:1912.05510*, 2019. 71

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012. 16

Dimitri P Bertsekas. *Dynamic programming and optimal control*. Athena scientific Belmont, 2000. 87

Christopher M Bishop. *Mixture density networks*. Aston University, 1994. 96

Lionel Blondé, Pablo Strasser, and Alexandros Kalousis. Lipschitzness is all you need to tame off-policy generative adversarial imitation learning. *arXiv preprint arXiv:2006.16785*, 2020. 26, 33, 39

Mathieu Blondel, Felipe Llinares-López, Robert Dadashi, Léonard Hussenot, and Matthieu Geist. Learning energy networks with generalized fenchel-young losses. *Advances in neural information processing systems*, 2022. 13

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 19

Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich. Displacement interpolation using lagrangian mass transport. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, 2011. 48

Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, 2011. 52

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`. 26, 28, 58, 126

Kianté Brantley, Wen Sun, and Mikael Henaff. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2020. 50, 51, 99

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016a. 21, 58, 66

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016b. 20, 27, 47

Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, 2019. 19, 42

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 2012. 99

Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *Twenty-fourth international joint conference on artificial intelligence*, 2015. 18

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018. 57, 71, 72, 79, 83

Donald W. Bushaw. Differential equations with a discontinuous forcing term. Technical report, STEVENS INST OF TECH HOBOKEN NJ EXPERIMENTAL TOWING TANK, 1953. 86, 95, 98

Richard W Byrne and Andrew Whiten. *Machiavellian intelligence: social expertise and the evolution of intellect in monkeys, apes, and humans*. Clarendon Press, 1989. 71

Richard W Byrne, Anne E Russon, et al. Learning by imitation: A hierarchical approach. *Behavioral and brain sciences*, 21(5):667–684, 1998. 10

Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. In *ACM/IEEE international conference on Human-robot interaction*, 2007. 99

Annie Chen, HyunJi Nam, Suraj Nair, and Chelsea Finn. Batch exploration with examples for scalable robotic reinforcement learning. *IEEE Robotics and Automation Letters*, 2021. 33

Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *International Conference on Autonomous Agents and Multiagent Systems*, 2007. 99

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017. 101

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations*, 2016. 182

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020. 66

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018. 17

Robert Dadashi, Léonard Hussenot, Matthieu Geist, and Olivier Pietquin. Primal wasserstein imitation learning. *International Conference on Learning Representations*, 2021a. 12, 58, 66, 93, 192

Robert Dadashi, Shideh Rezaeifar, Nino Vieillard, Léonard Hussenot, Olivier Pietquin, and Matthieu Geist. Offline reinforcement learning with pseudometric learning. In *International Conference on Machine Learning*, pages 2307–2318. PMLR, 2021b. 13

Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. *Proceedings of the 39th International Conference on Machine Learning (ICML 2022)*, 2022. 12

Edward L Deci and Richard M Ryan. Intrinsic motivation. *The corsini encyclopedia of psychology*, pages 1–2, 2010. 71

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022. 17

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 20

Yiming Ding, Carlos Florensa, Mariano Phielipp, and Pieter Abbeel. Goal-conditioned imitation learning. *Advances in Neural Information Processing Systems*, 2019. 99

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L Griffiths, and Alexei A Efros. Investigating human priors for playing video games. *arXiv preprint arXiv:1802.10217*, 2018. 72

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015. 98

Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. Temporal cycle-consistency learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 51, 188

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021. 26, 55

Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and Charles Isbell. Imitating latent policies from observation. In *International Conference on Machine Learning*, 2019. 49

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020. 66

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005. 66

Richard Evans and Jim Gao. Deepmind ai reduces google data centre cooling bill by 40https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40, 2021. 17

Robert Feldt. Generating diverse software versions with genetic programming: an experimental study. *IEE Proceedings-Software*, 145(6):228–236, 1998. 26, 55

Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *Conference on Uncertainty in Artificial Intelligence*, 2004. 43

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR, 2016. 19, 42, 66

R'emi Flamary and Nicolas Courty. Pot python optimal transport library, 2017. URL https://pythonot.github.io/. 48, 56

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 2021. 95

Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. *arXiv preprint arXiv:2109.00137*, 2021. 99

Roy Fox, Ron Berenstein, Ion Stoica, and Ken Goldberg. Multi-task hierarchical imitation learning for home automation. In *International Conference on Automation Science and Engineering (CASE)*. IEEE, 2019. 99

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2017. 19, 26, 28, 29, 33, 38, 42, 125, 126, 188

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020a. 28, 36

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020b. 21, 58, 66

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021. 99

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018. 32, 47, 87, 98, 125

Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018. 18

Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277. PMLR, 2020. 19, 26, 28, 29, 30, 38, 42, 66, 93, 126, 191

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 17

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014. 19, 26, 42

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, 2016. 98

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017. 17

Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. `https://github.com/tensorflow/agents`, 2018. URL `https://github.com/tensorflow/agents`. 21

Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv preprint arXiv:2006.13888*, 2020. 21, 66

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017. 26, 33, 127

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019. 95

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018a. 17, 27, 32, 51, 87, 94, 98, 124, 191

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018b. 58

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018c. 17, 124, 188, 191, 199

Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1:177–185, 1988. 26, 33, 128

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL http://github.com/google/flax. 28, 58

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, volume 32, 2018. 20, 26, 55, 66

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 17

Matteo Hessel, David Budden, Fabio Viola, Mihaela Rosca, Eren Sezener, and Tom Hennigan. Optax: composable gradient transformation and optimisation, in jax!, 2020. URL http://github.com/deepmind/optax. 128

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 18, 31, 50, 55, 87, 92, 202

Jacob Hilton, Suchir Balaji, Reiichiro Nakanom, and John Schulman. Webgpt: Improving the factual accuracy of language models through web browsing. https://openai.com/blog/webgpt/, 2016. 101

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016. 19, 26, 28, 29, 37, 38, 42, 58, 59, 66, 93, 94, 125

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 75

Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, Léonard Hussenot, Robert Dadashi, Gabriel Dulac-Arnold, Manu Orsini, Alexis Jacq, Johan Ferret, Nino Vieillard, Seyed Kamyar Seyed Ghasemipour, Sertan Girgin, Olivier Pietquin, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Abe Friesen, Ruba Haroun, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979. 12, 20, 21, 26, 28, 46, 47, 58, 182, 199, 202

Sarah L Holloway and Gill Valentine. *Children's geographies: Playing, living, learning*. Routledge, 2004. 71

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Variational information maximizing exploration. *Advances in Neural Information Processing Systems (NIPS)*, 2016. 83

JMcVLevine Hunt. Intrinsic motivation and its role in psychological development. In *Nebraska symposium on motivation*, volume 13, pages 189–282. University of Nebraska Press, 1965. 71

Léonard Hussenot, Matthieu Geist, and Olivier Pietquin. Copycat: Taking control of neural policies with constant attacks. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020. 13

Léonard Hussenot, Marcin Andrychowicz, Damien Vincent, Robert Dadashi, Anton Raichuk, Sabela Ramos, Nikola Momchev, Sertan Girgin, Raphael Marinier, Lukasz Stafiniak, et al. Hyperparameter selection for imitation learning. In *International Conference on Machine Learning*, pages 4511–4522. PMLR, 2021a. 12, 39

Léonard Hussenot, Marcin Andrychowicz, Damien Vincent, Robert Dadashi, Anton Raichuk, Sabela Ramos, Nikola Momchev, Sertan Girgin, Raphael Marinier, Lukasz Stafiniak, et al. Hyperparameter selection for imitation learning. In *International Conference on Machine Learning*, 2021b. 93

Léonard Hussenot, Robert Dadashi, Matthieu Geist, and Olivier Pietquin. Show me the way: Intrinsic motivation from demonstrations. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021c. 12

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017. 26

Alexis Jacq, Matthieu Geist, Ana Paiva, and Olivier Pietquin. Learning from a learner. In *International Conference on Machine Learning*, 2019. 19, 42

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017. 66

Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. Strictly batch imitation learning by energy-based distribution matching. *Advances in Neural Information Processing Systems*, 2020. 99

Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, and Stefan Schaal. Learning locomotion over rough terrain using terrain templates. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 167–172. IEEE, 2009. 19

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *Conference on Robot Learning*, 2018. 98

Harini Kannan, Danijar Hafner, Chelsea Finn, and Dumitru Erhan. Robodesk: A multi-task reinforcement learning benchmark. https://github.com/google-research/robodesk, 2021. 95, 201

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 39

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018. 17

Liyiming Ke, Matt Barnes, Wen Sun, Gilwoo Lee, Sanjiban Choudhury, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization. *arXiv preprint arXiv:1905.12888*, 2019. 42, 66

Celeste Kidd, Steven T Piantadosi, and Richard N Aslin. The goldilocks effect: Human infants allocate attention to visual sequences that are neither too simple nor too complex. *PloS one*, 7(5):e36399, 2012. 71

Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, pages 2859–2867, 2013. 55

Kee-Eung Kim and Hyun Soo Park. Imitation learning via kernel mean embedding. In *AAAI Conference on Artificial Intelligence*, 2018. 52

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015. 82, 124, 127, 128, 182, 191

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, 2019. 99

Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse reinforcement learning through structured classification. In *Advances in Neural Information Processing Systems*, pages 1007–1015, 2012. 52

Edouard Klein, Bilal Piot, Matthieu Geist, and Olivier Pietquin. A cascaded supervised learning approach to inverse reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 1–16. Springer, 2013. 19, 52, 72, 74

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 2000. 86, 98

Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *International Conference on Learning Representations*, 2019. 19, 26, 28, 30, 32, 33, 47, 58, 66, 94, 125, 126, 183, 191

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021. 199

Sanjay Krishnan, Roy Fox, Ion Stoica, and Ken Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning*, 2017. 99

Oliver Kroemer, Christian Daniel, Gerhard Neumann, Herke Van Hoof, and Jan Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *International Conference on Robotics and Automation*. IEEE, 2015. 99

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020. 199

V. Kumar and E. Todorov. Mujoco haptix: A virtual reality system for hand manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015. 50, 188

Vikash Kumar. *Manipulators and Manipulation in high dimensional spaces*. PhD thesis, University of Washington, Seattle, 2016. URL https://digital.lib.washington.edu/researchworks/handle/1773/38104. 58

Jonathan Lacotte, Mohammad Ghavamzadeh, Yinlam Chow, and Marco Pavone. Risk-sensitive generative adversarial imitation learning. In *International Conference on Artificial Intelligence and Statistics*, 2019. 19

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003. 66

Peter J Lang, Michael Davis, and Arne Öhman. Fear and anxiety: animal models and human cognitive psychophysiology. *Journal of affective disorders*, 61(3):137–159, 2000. 71

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012. 66

Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019. 39

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480, 2007. 66

Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. *Advances in Neural Information Processing Systems*, 31, 2018. 17

Hoang M Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*, 2018. 99

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 17

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012. 66

Kuang-Huei Lee, Ian Fischer, Anthony Liu, Yijie Guo, Honglak Lee, John Canny, and Sergio Guadarrama. Predictive information accelerates learning in rl. *arXiv preprint arXiv:2007.12401*, 2020. 56

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 17

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018. 17

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020. 66, 199

Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, 2017. 19

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016. 17, 47, 98

Sungsu Lim, Ajin Joseph, Lei Le, Yangchen Pan, and Martha White. Actor-expert: A framework for using q-learning in continuous action spaces. *arXiv preprint arXiv:1810.09103*, 2018. 98

Fangchen Liu, Zhan Ling, Tongzhou Mu, and Hao Su. State alignment-based imitation learning. *International Conference on Learning Representations*, 2020. 19

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018. 39

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. Technical report, University of Freiburg, 2018. 33, 128

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709, 2018. 39

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, 2020. 87, 91, 95, 99, 101

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. 83

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 2007. 43

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021. 91

Simon Manschitz, Jens Kober, Michael Gienger, and Jan Peters. Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems*, 2015. 99

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, et al. Teaching language models to support answers with verified quotes. *arXiv preprint arXiv:2203.11147*, 2022. 101

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017. 86, 98

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018. 33, 127

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 17, 42, 71, 87, 92, 202

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 17

Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 2125–2133, 2015. 71

P Read Montague, Peter Dayan, and Terrence J Sejnowski. A framework for mesencephalic dopamine systems based on predictive hebbian learning. *Journal of neuroscience*, 16(5):1936–1947, 1996. 9

Alistair Muldal, Yotam Doron, John Aslanides, Tim Harley, Tom Ward, and Siqi Liu. dm_env: A python interface for reinforcement learning environments, 2019. URL http://github.com/deepmind/dm_env. 21

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation*. IEEE, 2018. 86

Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and autonomous systems*, 47(2-3): 79–91, 2004. 19

Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. In *Conference on Robot Learning*, 2020. 98

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287, 1999. 18, 73, 86, 87

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, page 2, 2000. 19, 26, 42, 55, 87

Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. What matters for adversarial imitation learning? *Advances in Neural Information Processing Systems*, 34, 2021. 12, 94

Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org, 2017. 71, 72, 83

Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009. 71, 83

Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007. 83

Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, Krzysztof Choromanski, Anna Choromanska, and Michael Jordan. Learning to score behaviors for guided policy optimization. In *International Conference on Machine Learning*, 2020. 19

Tom Le Paine, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, et al. Making efficient use of demonstrations to solve hard exploration problems. *arXiv preprint arXiv:1909.01387*, 2019. 18, 31

Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020. 56, 66

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544*, 2022. 18

Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation*. IEEE, 2009. 99

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017. 71, 83

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 2018. 52

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008. 18

Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016. 17

Bilal Piot, Matthieu Geist, and Olivier Pietquin. Learning from demonstrations: Is it worth estimating a reward function? In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 17–32. Springer, 2013. 42, 55

Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted bellman residual minimization handling expert demonstrations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 549–564. Springer, 2014. 18, 51, 55

Bilal Piot, Matthieu Geist, and Olivier Pietquin. Bridging the gap between imitation learning and inverse reinforcement learning. *IEEE transactions on neural networks and learning systems*, 28(8):1814–1826, 2016. 52, 74

Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Večerík, et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018. 18

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988. 19

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991. 18, 26, 32, 42, 55, 56, 58, 87, 88, 94, 126

Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017. 26, 55

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. 14, 42, 43, 86, 87

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *arXiv preprint arXiv:2003.13678*, 2020. 39

Rouhollah Rahmatizadeh, Pooya Abolghasemi, Aman Behal, and Ladislau Bölöni. From virtual demonstration to real-world manipulation using lstm and mdn. In *AAAI Conference on Artificial Intelligence*, 2018. 99

Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, 2020. 71, 83

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017. 27, 92, 201

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018. 50, 51

Sabela Ramos, Sertan Girgin, Léonard Hussenot, Damien Vincent, Hanna Yakubovich, Daniel Toyama, Anita Gergely, Piotr Stanczyk, Raphael Marinier, Jeremiah Harmsen, et al. Rlds: an ecosystem to generate, share and use datasets in reinforcement learning. *arXiv preprint arXiv:2111.02767*, 2021. 12, 21, 101

Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: imitation learning via regularized behavioral cloning. *International Conference on Learning Representations*, 2020. 50, 51

Shideh Rezaeifar, Robert Dadashi, Nino Vieillard, Léonard Hussenot, Olivier Bachem, Olivier Pietquin, and Matthieu Geist. Offline reinforcement learning as anti-exploration. In *AAAI Conference on Artificial Intelligence*, 2022. 13

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005. 66

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, 2010. 18, 42, 73

Stuart Russell. Learning agents for uncertain environments. In *Conference on Computational learning theory*, 1998. 19, 26, 42, 55, 86

Moonkyung Ryu, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja, and Craig Boutilier. Caql: Continuous action q-learning. *International Conference on Learning Representations*, 2020. 98

Andrey Sakryukin, Chedy Raissi, and Mohan Kankanhalli. Inferring DQN structure for high-dimensional continuous control. In *International Conference on Machine Learning*, 2020. 86, 98

Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*, 2018. 18, 86

Stefan Schaal. Learning from demonstration. *Advances in Neural Information Processing Systems*, 1997. 18

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999. ISSN 1364-6613. doi: https://doi.org/10.1016/S1364-6613(99)01327-3. URL http://www.sciencedirect.com/science/article/pii/S1364661399013273. 18, 26, 55

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations*, 2016. 202

Jürgen Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991. 83

Yannick Schroecker and Charles L Isbell. State aware imitation learning. In *Advances in Neural Information Processing Systems*, 2017. 51

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015a. 17, 28, 48, 98

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b. 124

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 17, 28, 32, 98, 124

Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus. Is bang-bang control all you need? solving continuous control with bernoulli policies. In *Advances in Neural Information Processing Systems*, 2021. 98

Tanmay Shankar, Shubham Tulsiani, Lerrel Pinto, and Abhinav Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations*, 2019. 99

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. 25, 42, 66, 71, 86

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 17, 101

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 17, 101

Riley Simmons-Edler, Ben Eisner, Eric Mitchell, Sebastian Seung, and Daniel Lee. Q-learning for continuous actions with cross-entropy guided policies. *arXiv preprint arXiv:1903.10605*, 2019. 98

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, 1994. 26, 55

Özgür Şimşek and Andrew G Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd international conference on Machine learning*, pages 833–840, 2006. 71

Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *International Conference on Learning Representations*, 2020. 86, 99

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15 (1):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html. 26, 28, 33, 128, 191

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020. 56

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008. 71, 72, 79, 83

Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444, 2009. 72

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 43, 87

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998. 9, 14, 21

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 2000. 98

Adrien Ali Taıga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. On bonus-based exploration methods in the arcade learning environment. *International Conference on Learning Representations (ICLR)*, 2020. 72

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017. 83, 99

Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. In *AAAI Conference on Artificial Intelligence*, 2020. 86, 98

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020. 98

Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2018. 86, 98

Arash Tavakoli, Mehdi Fatemi, and Petar Kormushev. Learning to represent action values as a hypergraph on the action vertices. *Internation Conference in Learning Representations*, 2021. 86, 98

Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995. 17, 42, 66, 71

Chen Tessler, Guy Tennenholtz, and Shie Mannor. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems*, 2019. 98

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. 47

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *International Joint Conference on Artificial Intelligence*, 2018. 49

George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard E Turner, Zoubin Ghahramani, and Sergey Levine. The mirage of action-dependent baselines in reinforcement learning. *arXiv preprint arXiv:1802.10031*, 2018. 26

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. 17, 202

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017. 18, 87, 92, 202

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 2020. 17, 92, 202

Nino Vieillard, Marcin Andrychowicz, Anton Raichuk, Olivier Pietquin, and Matthieu Geist. Implicitly regularized rl with implicit q-values. *arXiv preprint arXiv:2108.07041*, 2021. 98

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009. 19, 43, 44, 56

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. 17, 19, 26, 55, 66, 86, 101

Ruohan Wang, Carlo Ciliberto, Pierluigi Vito Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. In *International Conference on Machine Learning*, pages 6536–6544. PMLR, 2019. 50, 51, 57, 99

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992. 16, 87

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. 17, 86, 98

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019. 99

Huang Xiao, Michael Herman, Joerg Wagner, Sebastian Ziesche, Jalal Etesami, and Thai Hong Linh. Wasserstein adversarial imitation learning. *arXiv preprint arXiv:1906.08113*, 2019. 19

Danfei Xu and Misha Denil. Positive-unlabeled reward learning. *arXiv preprint arXiv:1911.00459*, 2019. 26, 33, 127

Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *Robotics: Science and Systems*, 2018. 99

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 33, 127

Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020. 21

Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010. 19

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 19, 26, 42, 55, 87

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019. 101

Appendix

Appendix A

What Matters for Adversarial Imitation Learning? Details and additional figures.

A.1 List of Investigated Choices

In this section we list all algorithmic choices which we consider in our experiments. See section 2.5 for an introduction to adversarial imitation and the notation used in this section. For convenience, we mark each of the choices with a number (e.g., C8) and a fixed name (e.g. `RL Algorithm (C8)`) that can be easily used to find a description of the choice in this section.

A.1.1 Reinforcement Learning algorithms

In all experiments we use MLPs for the policy and critic/value networks and sample the following HPs controlling the networks architectures: `policy MLP depth (C1)` (the number of *hidden* layers), `policy MLP width (C2)`, `critic MLP depth (C3)`, `critic MLP width (C4)`, `RL activation (C5)`, as well as `discount` γ `(C6)` and `batch size (C7)`. All networks are optimized with the Adam Kingma and Ba [2015] optimizer.

We sample `RL Algorithm (C8)` from the following options:

Proximal Policy Optimization (PPO, Schulman et al. [2017]) For PPO, `batch size (C7)` denotes the number of experience fragments, each of consisting `PPO unroll length (C9)` transitions, collected in each policy update step. In each policy update step, we perform `PPO number of epochs (C10)` passes over the gathered data when in each pass the data is split into `PPO number of minibatches (C11)` minibatches. We use the PPO loss with the clipping threshold set by `PPO clipping` ϵ `(C12)` and add an entropy loss with the coefficient specified by `PPO entropy cost (C13)`. We also sample `PPO learning rate (C14)`, and the GAE Schulman et al. [2015b] returns mixing coefficient `GAE` λ `(C15)`.

Soft Actor Critic (SAC, Haarnoja et al. [2018a]) We use a version of SAC with a policy entropy constraint Haarnoja et al. [2018c]. In particular, we choose `SAC entropy per dimension (C16)` and that

set the entropy constraint so that the policy entropy is not lower than the number of action dimensions times this value. We also sweep `SAC learning rate (C17)` and the target network polyak averaging coefficient `SAC polyak` τ `(C18)` (the target network is updates after each minibatch).

Twin Delayed Deep Deterministic Policy Gradient (TD3, Fujimoto et al. [2018]) For TD3, we sweep `TD3 policy learning rate (C19)` and `TD3 critic learning rate (C20)` separately, as well as sample `behavioral policy noise (C21)`. Following the original publication, we update the actor only using every other minibatch while the critic networks uses all minibatches. The target network is updated after every minibatch with the polyak coefficient fixed to 0.005. Following DAC Kostrikov et al. [2019], we clip actor gradients with magnitudes bigger than `TD3 gradient clipping (C22)`.

Distributed Distributional Deterministic Policy Gradients (D4PG, Barth-Maron et al. [2018]) This algorithm is similar to TD3 but uses a distributional C51-style critic Bellemare et al. [2017] outputting distributions over `number of atoms (C23)` atoms spaced equally between `-VMax (C24)` and `VMax (C24)` as well as `N-step returns (C25)` returns. In contrast to the original D4PG Barth-Maron et al. [2018], we use a single actor and do not use prioritized replay. The target network is fully updated every 100 training batches. As usual, we also sweep `D4PG learning rate (C26)`.

Moreover, for off-policy algorithm (SAC, TD3 and D4PG) we sample `replay ratio (C27)` which denotes the average number of times each transition is replayed. This is achieved in the following way — if `replay ratio (C27)` $\geq$ `batch size (C7)` than we replay `replay ratio (C27)` / `batch size (C7)` batches (each with `batch size (C7)` transitions) after every environment step. If `batch size (C7)` $>$ `replay ratio (C27)`, we replay a single batch every `batch size (C7)` / `replay ratio (C27)` transitions. The transitions for replay are sampled uniformly from a FIFO replay buffer of size `RL replay buffer size (C28)` and we start training whenever we have at least 10k transition in the buffer.

For the RL algorithms which train stochastic policies (PPO and SAC) we use a Gaussian distribution followed by tanh to squash actions into the $[-1, 1]$ range.[1] More precisely, the policy network output is split into two parts — μ and ρ, and the action distribution used during training is $\tanh(\mathcal{N}(\mu, \text{softplus}(\rho)+0.001))$. For policy evaluation, we choose `evaluation behavior policy type (C29)` from the following options:

- *stochastic*: sample from the distribution (same as behavioral policy used during training),
- *mode*: use the mode of the Gaussian instead of sampling,
- *average*: sample five action from the distribution and take the average of them.

A.1.2 Imitation-specific changes to RL

Reward function Let D denote the probability that a state-action pair (s, a) is classified as *expert* by the discriminator while h is the discriminator logit, i.e. $D = \sigma(h)$ where σ denotes the sigmoid function. Depending on the value of `reward function (C30)` we use one of the following reward functions (for completeness we write the formulas as a function of D as well as h):

- $r(s, a) = -\ln(1 - D) = \text{softplus}(h)$ (used in the original GAIL paper[2] Ho and Ermon [2016]),
- $r(s, a) = \ln D - \ln(1 - D) = h$ (introduced in AIRL Fu et al. [2017]).

[1] The action coordinates are scaled to $[-1, 1]$ regardless of the RL algorithm used.
[2] The GAIL paper uses the inverse convention in which D denotes the probability as being classified as *non-expert*.

- $r(s,a) = \ln D = -\text{softplus}(-h)$,
- $r(s,a) = -he^h$ (introduced in FAIRL Ghasemipour et al. [2020]).

We also clip rewards with the absolute values higher than `max reward magnitude (C31)`.

Absorbing state We optionally (if `absorbing state (C32)=True`) apply the absorbing state technique from DAC Kostrikov et al. [2019]. This technique encourages the agent to generate episodes of similar length to the ones of the expert. In particular, the demonstration and agent episodes are processed in the following way: for each terminal transition, we replace it with a non-terminal transition to a special absorbing state[3] and also add a transition from the absorbing state to itself with a zero action.

Replaying demonstrations For off-policy RL algorithms, we optionally (if `policy-to-expert replay ratio (C33)`$\neq \infty$) sample batches for RL training not only from the replay buffer, but also from the demonstrations. In particular, the ratio of policy to expert data in each minibatch is equal to `policy-to-expert replay ratio (C33)`.

Initialization with behavior cloning We optionally (if `BC pretraining (C34)=True`) pre-train the policy network offline at the beginning of training using Behavior Cloning Pomerleau [1991]. In particular, we perform 100k gradient steps with Adam on the MSE loss, using learning rate 10^{-4} and batch size 256.

A.1.3 Discriminator parameterization

Depending on the value of `discriminator input (C35)`, the discriminator is fed single states, state-action pairs, state-state pairs or state-action-state tuples.

Our basic discriminator architecture is an MLP with `discriminator MLP depth (C36)` hidden layers, each of size `discriminator MLP width (C37)` with the activation function specified by `discriminator activation (C38)`. Its output is interpreted as the logit of the probability of being classified as expert, i.e. for a state-action-state tuple (s,a,s') we have $D(s,a,s') = \sigma(f(s,a,s'))$, where D is the probability of classifying the tuple (s,a,s') as expert, σ denotes the sigmoid function, and f is a learnable function represented as an MLP.

We also consider two modifications introduced in the AIRL Fu et al. [2017] paper. The first one (enabled if `reward shaping (C39)=True`) adds a reward shaping term where the f function is parameterized in the following way: $f(s,a,s') = g(s,a,s') + \gamma h(s') - h(s)$ where g and h are MLPs parameterized as described above, and γ is the RL discount factor.[4] The second modification (enabled if `subtract log-pi (C40)=True`) parameterizes the discriminator as $D(s,a,s') = \frac{\exp(f(s,a,s'))}{\exp(f(s,a,s'))+\pi(a|s)}$, where π is the current agent policy. It can be easy shown that it is equivalent to $D(s,a,s') = \sigma((f(s,a,s') - \log \pi(a|s))$ so this just shifts the logits by $\log \pi(a|s)$.

A.1.4 Discriminator training

All discriminator weight matrices use the `lecun_uniform` initializer from JAX Bradbury et al. [2018]. The last discriminator layer initialization is additionally multiplied by `discriminator last layer init scale (C41)`.

[3] In practice, this is done by adding a special bit to every observation which is set to zero for normal observations and one for the absorbing state. The remaining bits of the absorbing state are all zeros.

[4] The inputs fed to g are specified by `discriminator input (C35)`.

The discriminator is trained with the Adam Kingma and Ba [2015] optimizer, the learning rate specified by `discriminator learning rate (C42)` and the cross-entropy loss. Each data batch contains exactly `batch size (C7)` expert transitions and `batch size (C7)` policy transitions. The policy transitions are sampled uniformly from a FIFO replay buffer of size `discriminator replay buffer size (C43)`.

We perform `discriminator to RL updates ratio (C44)` discriminator gradient steps for each RL gradient step. More precisely, after each environment step, we compute the number of RL gradient steps as described in App. A.1.1, and perform `discriminator to RL updates ratio (C44)` that many discriminator gradient steps *before* performing the RL update.

A.1.5 Discriminator regularization

Depending on the value of `discriminator regularizer (C45)`, we optionally apply one of the following regularizers to the discriminator:

Gradient Penalty (GP, Gulrajani et al. [2017]) Gradient penalty is parameterized with `gradient penalty k (C46)` and `gradient penalty` λ `(C47)`. This regularizer adds an extra term in the discriminator loss that encourages the discriminator gradient to be close to k on a convex combination of positive (expert) and negative (policy) data. In particular, for an expert data $x \sim \mathcal{D}_{expert}$ and policy data $\widetilde{x} \sim \mathcal{D}_\pi$, the gradient penalty is defined as $\lambda(||\nabla_{\hat{x}} D(\hat{x})||_2 - k)^2$, where $\hat{x}$ is a convex combination of x and $\widetilde{x}$, i.e. $\hat{x} := \epsilon x + (1-\epsilon)\widetilde{x}$ and ϵ follows a uniform distribution: $\epsilon \sim U[0,1]$. In practice, k is usually chosen to be 0 (penalty for high gradients) or 1 (penalty for gradients with norms far from 1). Our gradient penalty implementation uses the gradient of the discriminator logit instead of the classification probability.

Spectral normalization Miyato et al. [2018] Spectral normalization guarantees that the discriminator is 1-Lipschitz: $|D(x_2) - D(x_1)| \leq ||x_2 - x_1||$. It does so by dividing each dense layer matrix by its highest eigenvalue which can be efficiently computed with the power iteration method. See Miyato et al. [2018] for details.

Mixup Zhang et al. [2017] Mixup is parameterized with `mixup` α `(C48)` and relies on training the discriminator on a convex combination of positive (expert) and negative (policy) data. With expert data $x \sim \mathcal{D}_{expert}$ and policy data $\widetilde{x} \sim \mathcal{D}_\pi$, let ϵ follow a Beta distribution: $\epsilon \sim Beta(\alpha, \alpha)$. Instead of training the discriminator on x and $\widetilde{x}$ separately, we only train it on the convex combination of them $\hat{x} := \epsilon x + (1-\epsilon)\widetilde{x}$ with the label being the convex combinations of the labels, i.e. expert with probability ϵ and non-expert with probability $1-\epsilon$, so that the loss is $-\epsilon \ln D(\hat{x}) - (1-\epsilon)\ln(1 - D(\hat{x}))$.

Positive Unlabeled GAIL (PUGAIL, Xu and Denil [2019]) Normally the discriminator is trained under the assumption that expert trajectories are positive examples and policy trajectories are negative examples. The PUGAIL loss assumes instead that policy trajectories are a mix of positive and negative examples.

With `PUGAIL` η `(C49)` denoting the assumed proportion of positive samples in the policy data and `PUGAIL` β `(C50)` being a clipping threshold, the discriminator is trained with the following loss:

$$\eta \hat{\mathbb{E}}_{x \sim \mathcal{D}_{expert}}[-\ln(D(x))] + \max\left(-\beta,\ \hat{\mathbb{E}}_{x \sim \mathcal{D}_\pi}[-\ln(1 - D(x))] - \eta \hat{\mathbb{E}}_{x \sim \mathcal{D}_{expert}}[-\ln(1 - D(x))]\right).$$

Dropout Srivastava et al. [2014] We apply dropout to the hidden layers (`dropout hidden rate (C51)`) as well as inputs (`dropout input rate (C52)`). See Srivastava et al. [2014] for the description of dropout.

Weight decay Hanson and Pratt [1988], Loshchilov and Hutter [2018] Weight decay is parameterized with a parameter controlling its strength `weight decay` λ `(C53)`. Normally, weight decay is applied by adding a sum of the squares of the network parameters to the loss. However, this may interact negatively with an adaptive gradient optimizer like Adam Kingma and Ba [2015] unless the optimizer is modified appropriately Loshchilov and Hutter [2018]. In our experiments, we use a version of Adam with weight decay called AdamW Loshchilov and Hutter [2018] from the Optax library Hessel et al. [2020]. See Loshchilov and Hutter [2018] for the details.

Entropy bonus Similarly to entropy bonus in RL, we also experiment with adding to the discriminator loss a term proportional to the entropy of the discriminator output treated as a Bernoulli distribution: $\lambda\,(D \ln D + (1 - D) \ln(1 - D))$ where `entropy` λ `(C54)` is a HP.

A.1.6 Observation normalization

We optionally apply input normalization (choice `observation normalization (C55)`) which transforms linearly the observations to all neural networks (in the RL algorithm as well as the discriminator) so that each coordinate has approximately mean equal zero and standard deviation equal one. This is done by subtracting from each observation μ and dividing by $\max(\rho, 0.001)$, where μ and ρ are the empirical mean and standard deviation of either all demonstrations (we call it *fixed* normalization because it does not change during training) or the empirical mean and standard deviation of all the observations encountered by the policy being trained so far (called *online* because it changes during training).

A.1.7 Combining multiple batches

We consider processing multiple batches at once for improved accelerator (GPU or TPU) utilization (choice `number of combined batches (C56)`). In particular, we sample an N-times larger batch from a replay buffer, split it back into N smaller/proper batches on an accelerator, and process them sequentially. In order to keep the replay ratio unaffected, we decrease the frequency of updates accordingly, e.g. instead of performing one gradient update for every environment step, we perform N gradients updates every N environment steps. We apply this technique to the discriminator as well as the RL agent training.

A.2 Best hyperparameter values

Table A.1 shows the best value found for each HP in the main experiment. See App. A.5 for the full experimental report. The sample complexity can be slightly improved by decreasing `number of combined batches (C56)` and increasing `discriminator to RL updates ratio (C44)`. We used the suboptimal values from Table A.1 because they give a good trade-off between sample complexity and runtime. `discriminator learning rate (C42)` equal 10^{-6} is better when PUGAIL, entropy or no discriminator regularizer is used, and $3 \cdot 10^{-5}$ is better otherwise. The performance of observation normalization schemes depends heavily on the environment *and* discriminator regularization used. For completeness, we present the best HPs for all discriminator regularizers.

Table A.1: Best hyperparameter configuration.

Choice	Name	Best value
C1	policy MLP depth	2
C2	policy MLP width	256
C3	critic MLP depth	2
C4	critic MLP width	256
C5	RL activation	ReLu
C6	discount γ	0.97
C7	batch size	256
C8	RL Algorithm	SAC
C16	SAC entropy per dimension	-0.5
C17	SAC learning rate	$3 \cdot 10^{-4}$
C18	SAC polyak τ	0.01
C27	replay ratio	256
C28	RL replay buffer size	$3 \cdot 10^{6}$
C29	evaluation behavior policy type	mode
C30	reward function	AIRL
C31	max reward magnitude	∞
C32	absorbing state	True
C33	policy-to-expert replay ratio	∞
C34	BC pretraining	True
C35	discriminator input	(s, a)
C36	discriminator MLP depth	1
C37	discriminator MLP width	64
C38	discriminator activation	ReLu
C39	reward shaping	False
C40	subtract log-pi	False
C41	discriminator last layer init scale	1
C42	discriminator learning rate	10^{-6} or $3 \cdot 10^{-5}$
C43	discriminator replay buffer size	$3 \cdot 10^{6}$
C44	discriminator to RL updates ratio	1
C45	discriminator regularizer	spectral normalization
C46	gradient penalty k	0
C47	gradient penalty λ	1
C48	mixup α	1
C49	PUGAIL η	0.7
C50	PUGAIL β	∞
C51	dropout hidden rate	75%
C52	dropout input rate	50%
C53	weight decay λ	10
C54	entropy λ	0.03
C55	observation normalization	depends on the environment
C56	number of combined batches	8

A.3 Expert and random policy scores

Table A.2: Expert and random policy scores used to normalize the performance for all tasks.

Task	Random policy score	Expert score
`HalfCheetah-v2`	-282	8770
`Hopper-v2`	18	2798
`Walker2d-v2`	1.6	4118
`Ant-v2`	-59	5637
`Humanoid-v2`	123	9115
`pen-expert-v0`	94	3078
`door-expert-v0`	-56	2882
`door-human-v0`	-56	796
`hammer-expert-v0`	-274	12794
`hammer-human-v0`	-274	3071

A.4 Experiment wide

A.4.1 Design

For each of the 10 tasks, we sampled 12083 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `RL Algorithm (C8)`: {d4pg, ppo, sac, td3}
  - For the case "`RL Algorithm (C8)` = sac", we further sampled the sub-choices:
    - `SAC learning rate (C17)`: {0.0001, 0.0003, 0.001}
    - `SAC entropy per dimension (C16)`: {-2.0, -1.0, -0.5, 0.0}
    - `SAC polyak` τ `(C18)`: {0.001, 0.003, 0.01, 0.03}
    - `subtract log-pi (C40)`: {False, True}
    - `batch size (C7)`: {256.0}
  - For the case "`RL Algorithm (C8)` = d4pg", we further sampled the sub-choices:
    - `D4PG learning rate (C26)`: {3e-05, 0.0001, 0.0003}
    - `behavioral policy noise (C21)`: {0.1, 0.2, 0.3, 0.5}
    - `VMax (C24)`: {150.0, 750.0, 1500.0}
    - `number of atoms (C23)`: {51.0, 101.0, 201.0, 401.0}
    - `N-step returns (C25)`: {1.0, 3.0, 5.0}
    - `batch size (C7)`: {256.0}
  - For the case "`RL Algorithm (C8)` = td3", we further sampled the sub-choices:
    - `TD3 policy learning rate (C19)`: {0.0001, 0.0003, 0.001}
    - `TD3 critic learning rate (C20)`: {0.0001, 0.0003, 0.001}
    - `TD3 gradient clipping (C22)`: {40.0, ∞}
    - `behavioral policy noise (C21)`: {0.1, 0.2, 0.3, 0.5}
    - `batch size (C7)`: {256.0}
  - For the case "`RL Algorithm (C8)` = ppo", we further sampled the sub-choices:
    - `PPO learning rate (C14)`: {3e-05, 0.0001, 0.0003}
    - `PPO number of epochs (C10)`: {2.0, 5.0, 10.0, 20.0}
    - `PPO entropy cost (C13)`: {0.0, 0.001, 0.003, 0.01, 0.03, 0.1}
    - `PPO number of minibatches (C11)`: {8.0, 16.0, 32.0, 64.0}
    - `PPO unroll length (C9)`: {4.0, 8.0, 16.0, 32.0}
    - `PPO clipping` ϵ `(C12)`: {0.1, 0.2, 0.3}
    - `GAE` λ `(C15)`: {0.8, 0.9, 0.95, 0.99}
    - `subtract log-pi (C40)`: {False, True}
    - `batch size (C7)`: {64.0, 128.0, 256.0}
- `RL replay buffer size (C28)`: {300000.0, 1000000.0, 3000000.0}
- `policy MLP depth (C1)`: {1, 2, 3}

- `policy MLP width (C2)`: {64, 128, 256, 512}
- `critic MLP depth (C3)`: {1, 2, 3}
- `critic MLP width (C4)`: {64, 128, 256, 512}
- `RL activation (C5)`: {relu, tanh}
- `discount` γ `(C6)`: {0.9, 0.97, 0.99, 0.997}
- `BC pretraining (C34)`: {False, True}
- `absorbing state (C32)`: {False, True}
- `discriminator replay buffer size (C43)`: {300000, 1000000, 3000000}
- `reward shaping (C39)`: {False, True}
- `discriminator input (C35)`: {s, sa, sas, ss}
- `discriminator MLP depth (C36)`: {1, 2, 3}
- `discriminator MLP width (C37)`: {16, 32, 64, 128, 256, 512}
- `discriminator activation (C38)`: {elu, leaky_relu, relu, sigmoid, swish, tanh}
- `discriminator last layer init scale (C41)`: {0.001, 1.0}
- `discriminator regularizer (C45)`: {GP, Mixup, No regularizer, PUGAIL, dropout, entropy, spectral norm, weight decay}
  - For the case "`discriminator regularizer (C45)` = GP", we further sampled the sub-choices:
    - `gradient penalty` λ `(C47)`: {0.1, 1.0, 10.0}
    - `gradient penalty k (C46)`: {0.0, 1.0}
  - For the case "`discriminator regularizer (C45)` = Mixup", we further sampled the sub-choices:
    - `mixup` α `(C48)`: {0.1, 0.4, 1.0}
  - For the case "`discriminator regularizer (C45)` = PUGAIL", we further sampled the sub-choices:
    - `PUGAIL` η `(C49)`: {0.25, 0.5, 0.7}
    - `PUGAIL` β `(C50)`: {0.0, 0.7, ∞}
  - For the case "`discriminator regularizer (C45)` = entropy", we further sampled the sub-choices:
    - `entropy` λ `(C54)`: {0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3}
  - For the case "`discriminator regularizer (C45)` = weight decay", we further sampled the sub-choices:
    - `weight decay` λ `(C53)`: {0.3, 1.0, 3.0, 10.0, 30.0}

  - For the case "`discriminator regularizer (C45)` = dropout", we further sampled the sub-choices:
    * `dropout input rate (C52)`: {0.0, 0.25, 0.5, 0.75}
    * `dropout hidden rate (C51)`: {0.25, 0.5, 0.75}

- `observation normalization (C55)`: {fixed, none}
- `evaluation behavior policy type (C29)`: {average, mode, stochastic}
- `discriminator learning rate (C42)`: {1e-06, 3e-06, 1e-05, 3e-05, 0.0001, 0.0003}
- `max reward magnitude (C31)`: {0.5, 1.0, 2.0, 5.0, 10.0, 50.0, ∞}
- `reward function (C30)`: {-ln(1-D), AIRL, FAIRL, ln(D)}
- `replay ratio (C27)`: {256}
- `discriminator to RL updates ratio (C44)`: {1}
- `number of combined batches (C56)`: {8}

A.4.2 Results

For each of the sampled choice configurations we compute the performance metric as described in Section 3.2. We report aggregate statistics of the experiment in Tables A.3–A.6 as well as training curves in Figure A.1. We further provide per-choice analyses in Figures A.2-A.15.

Table A.3: Quantiles of the *final* agent performance across HP configurations for OpenAI Gym tasks.

	Ant	HalfCheetah	Hopper	Humanoid	Walker2d
90%	0.18	0.80	0.99	0.06	0.56
95%	0.56	0.98	1.15	0.30	0.85
99%	0.92	1.10	1.20	0.79	0.99
Max	1.10	1.39	1.32	1.02	1.06

Table A.4: Quantiles of the *final* agent performance across HP configurations for Adroit tasks.

	Door expert	Door human	Hammer expert	Hammer human	Pen expert
90%	0.12	0.07	0.16	0.12	0.28
95%	0.42	0.28	0.67	0.47	0.46
99%	0.90	1.20	1.26	2.03	0.77
Max	1.11	2.82	1.42	5.39	1.12

Table A.5: Quantiles of the *average* agent performance during training across HP configurations for OpenAI Gym tasks.

	Ant	HalfCheetah	Hopper	Humanoid	Walker2d
90%	0.13	0.54	0.62	0.05	0.31
95%	0.31	0.66	0.80	0.20	0.49
99%	0.62	0.85	0.98	0.49	0.71
Max	0.94	0.99	1.08	0.84	0.92

Table A.6: Quantiles of the *average* agent performance during training across HP configurations for Adroit tasks.

	Door expert	Door human	Hammer expert	Hammer human	Pen expert
90%	0.11	0.11	0.11	0.15	0.21
95%	0.27	0.24	0.34	0.33	0.35
99%	0.55	0.61	0.78	0.85	0.59
Max	0.87	1.65	1.01	1.97	0.84

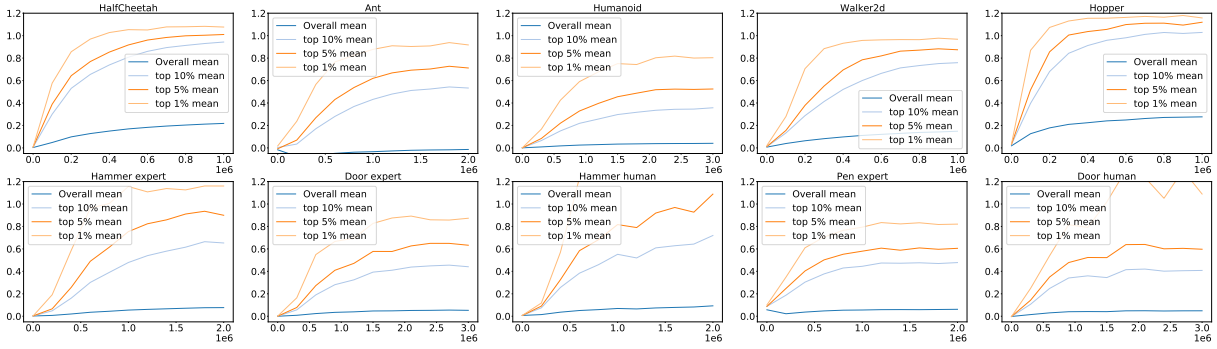


Figure A.1: Training curves.

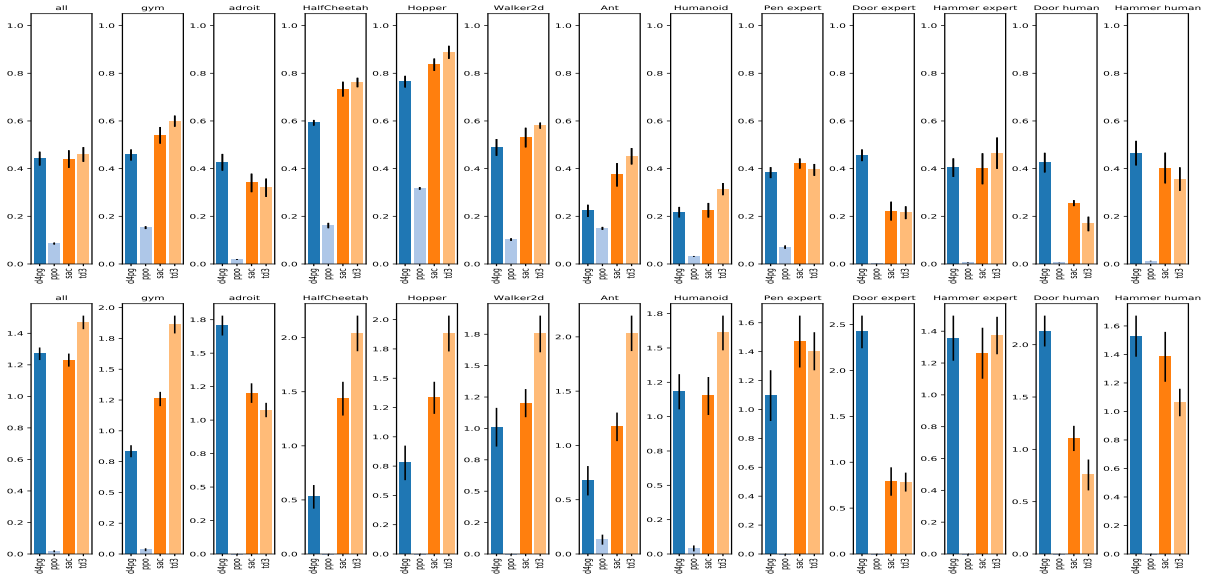


Figure A.2: Analysis of choice `RL Algorithm` (C8): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

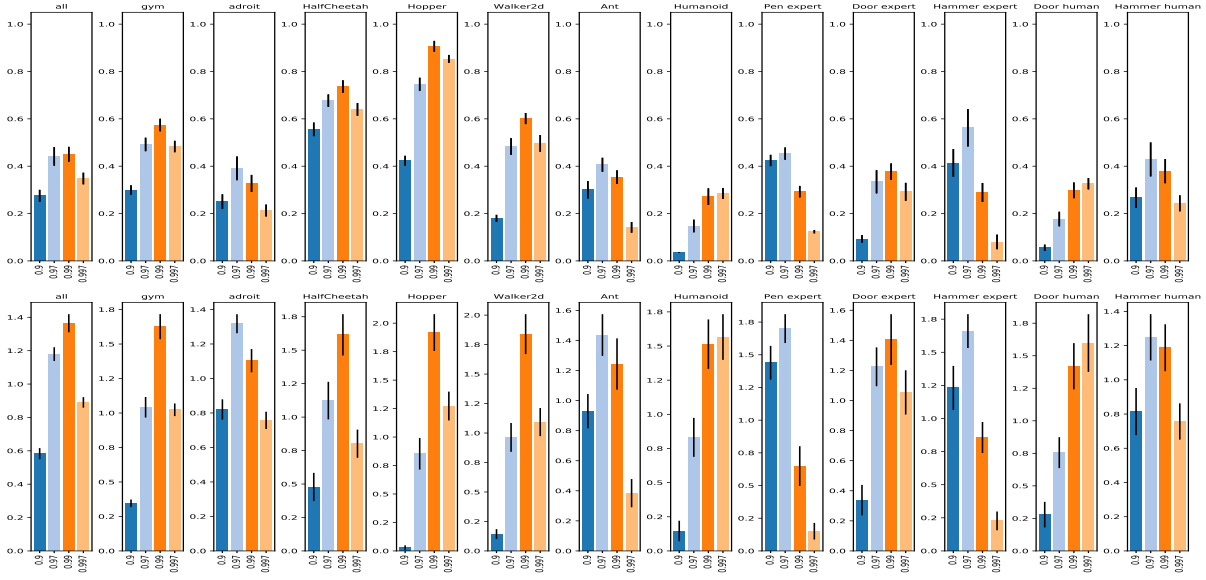


Figure A.3: Analysis of choice `discount` γ (C6): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

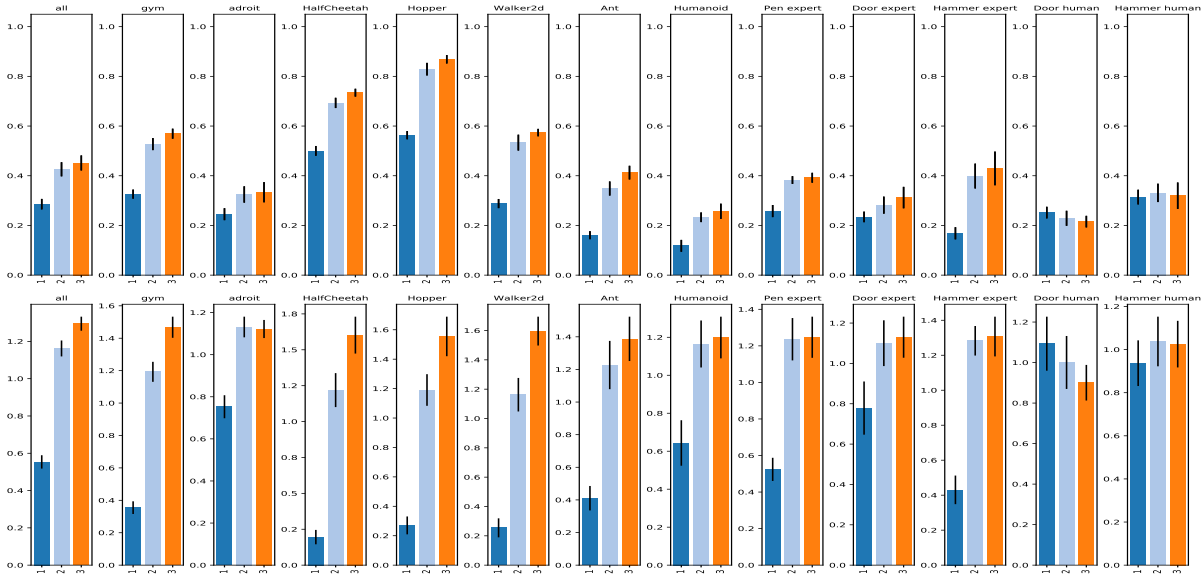


Figure A.4: Analysis of choice `critic MLP depth (C3)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

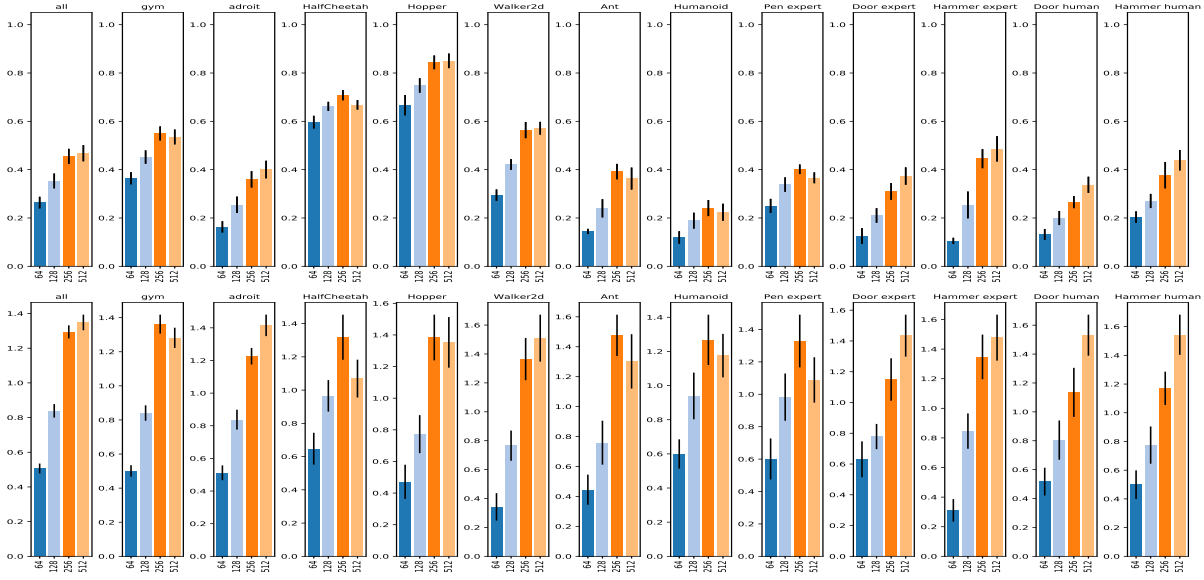


Figure A.5: Analysis of choice `critic MLP width (C4)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

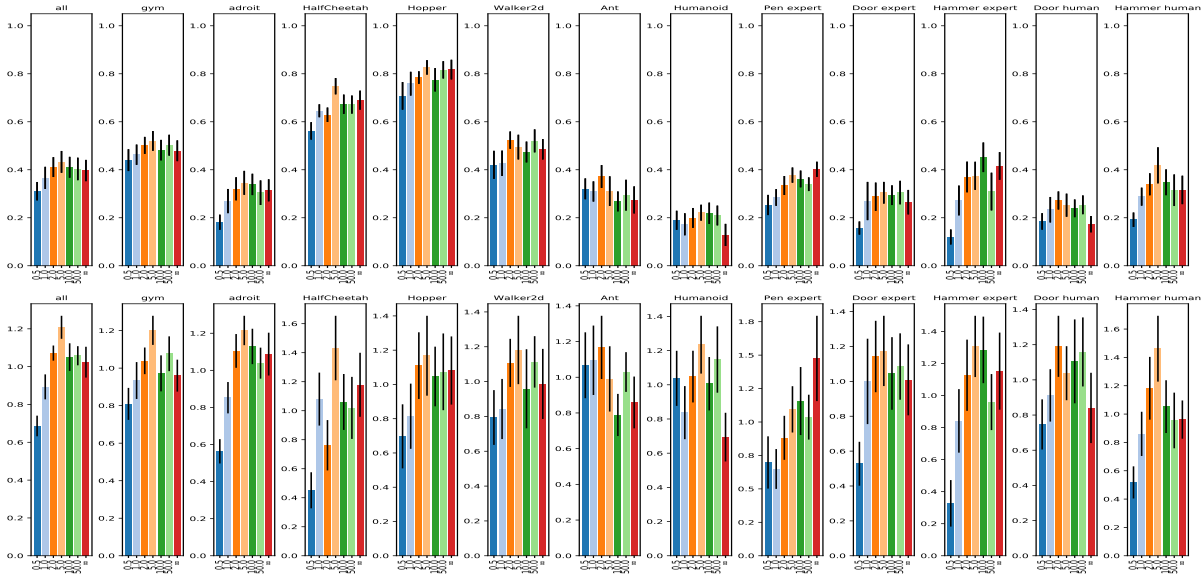


Figure A.6: Analysis of choice `max reward magnitude (C31)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

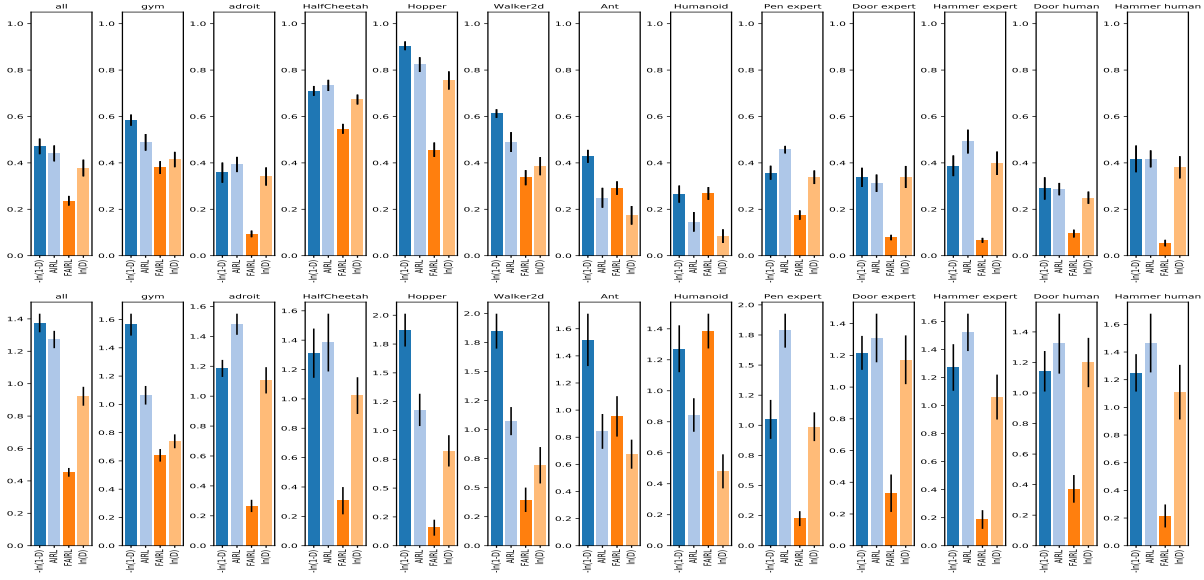


Figure A.7: Analysis of choice `reward function (C30)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

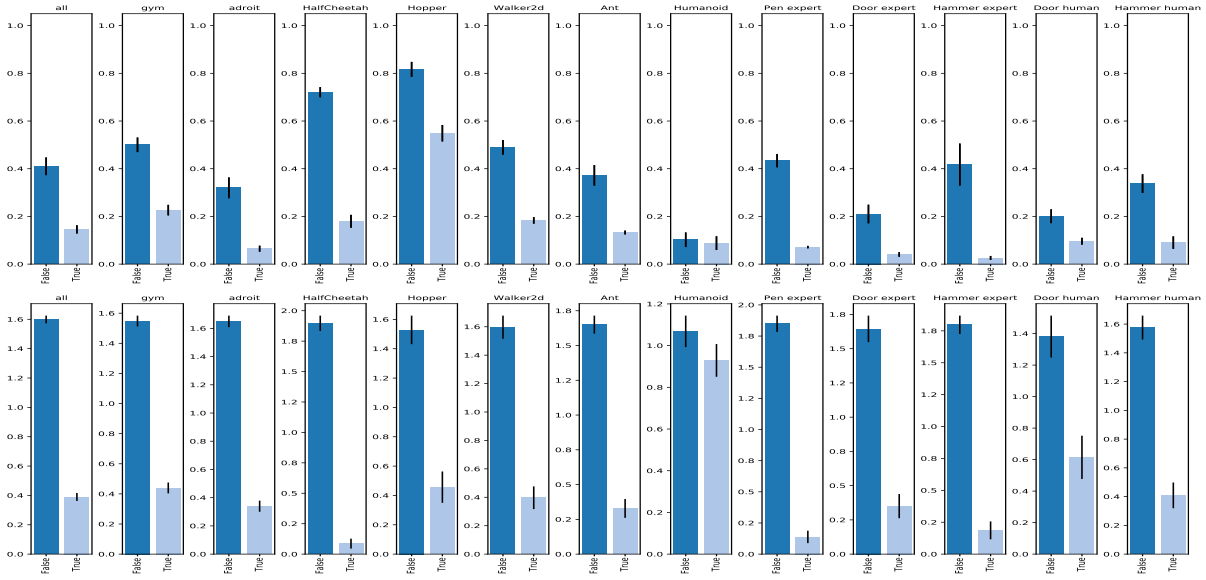


Figure A.8: Analysis of choice `subtract log-pi` (C40): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

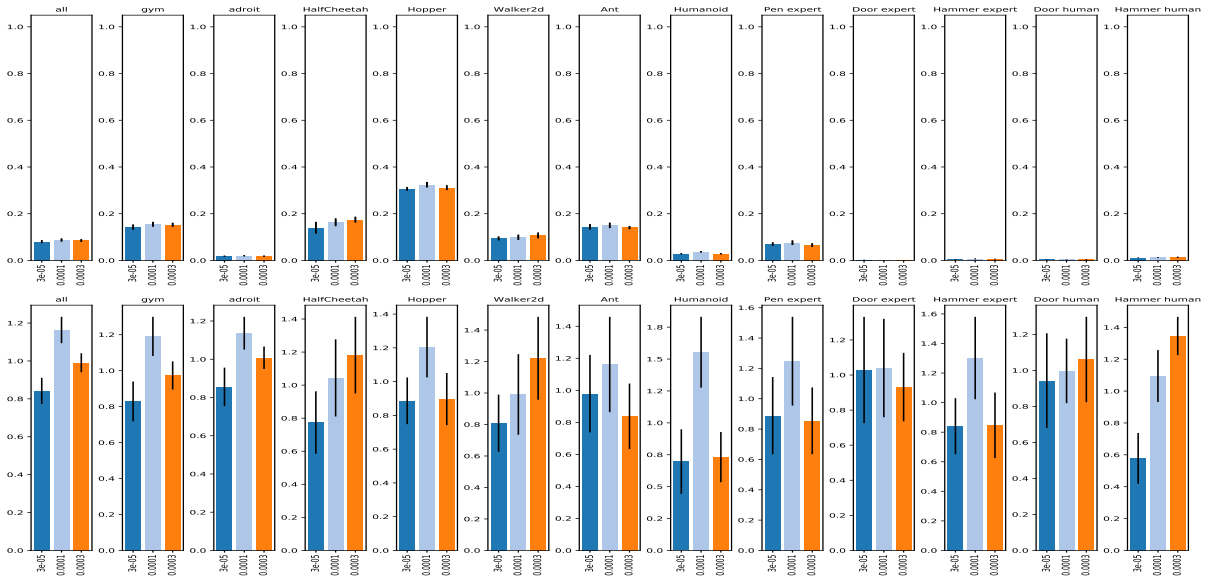


Figure A.9: Analysis of choice `PPO learning rate` (C14): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

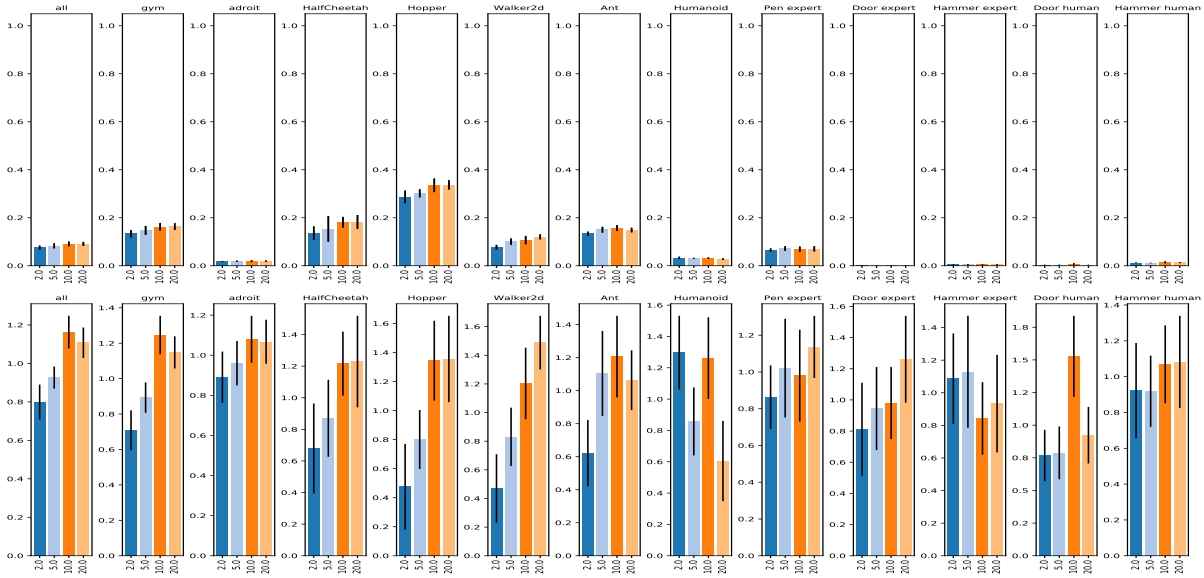


Figure A.10: Analysis of choice `PPO number of epochs` (C10): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

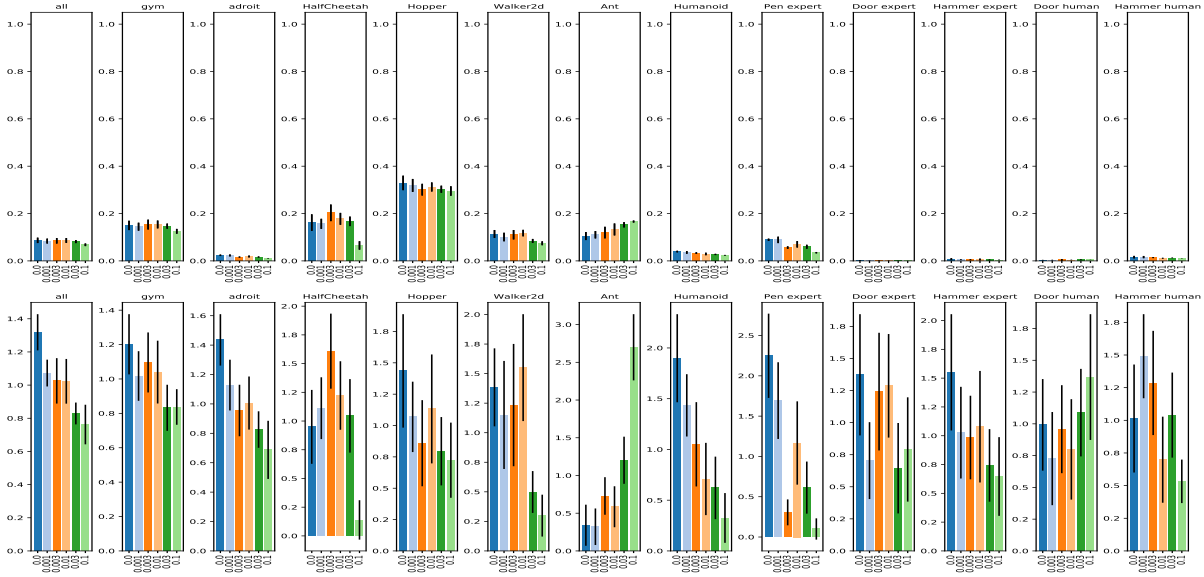


Figure A.11: Analysis of choice `PPO entropy cost` (C13): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

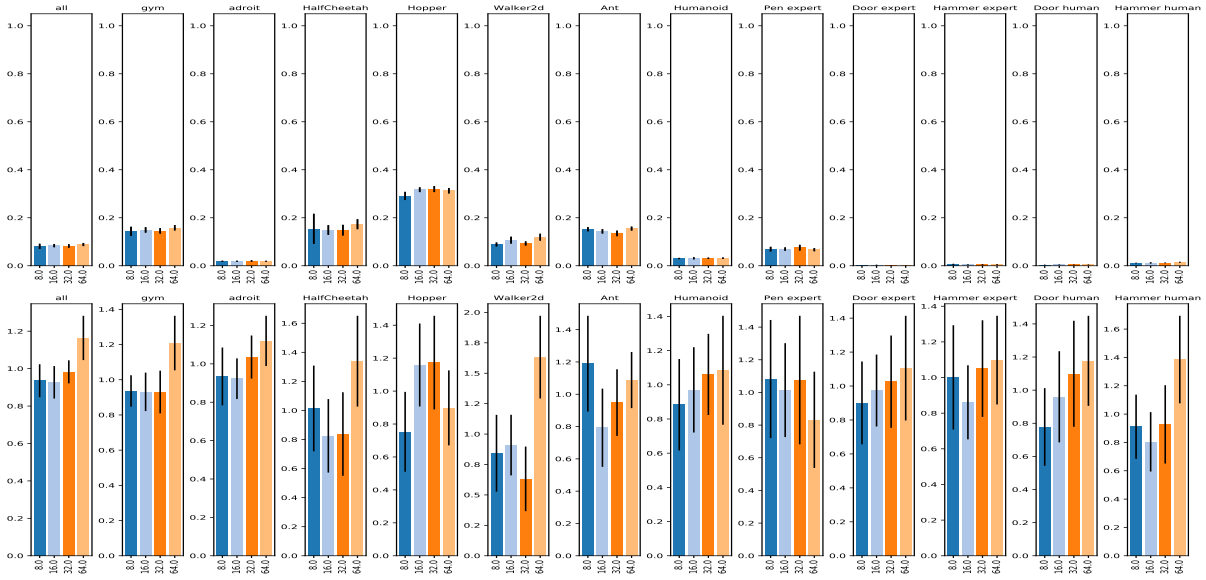


Figure A.12: Analysis of choice `PPO number of minibatches` (C11): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

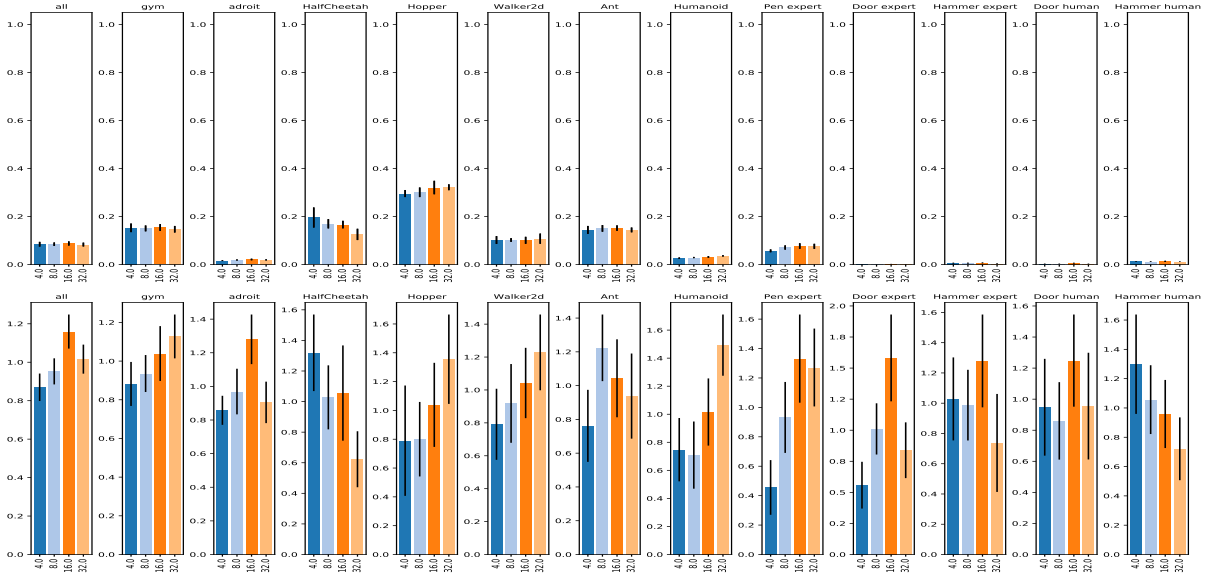


Figure A.13: Analysis of choice `PPO unroll length` (C9): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

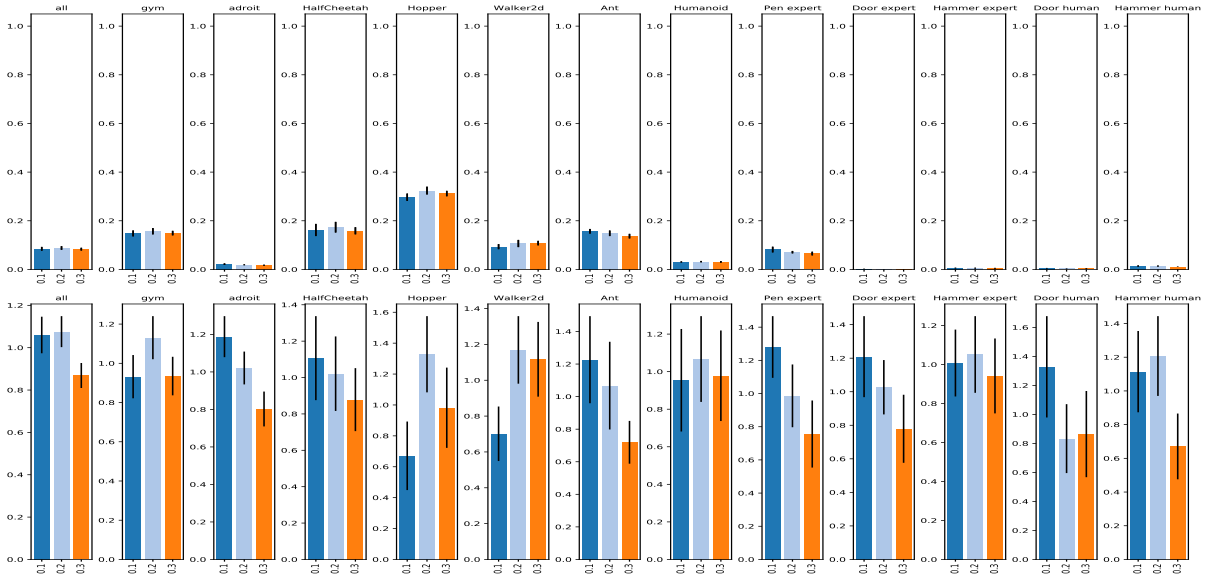


Figure A.14: Analysis of choice `PPO clipping` ϵ (C12): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

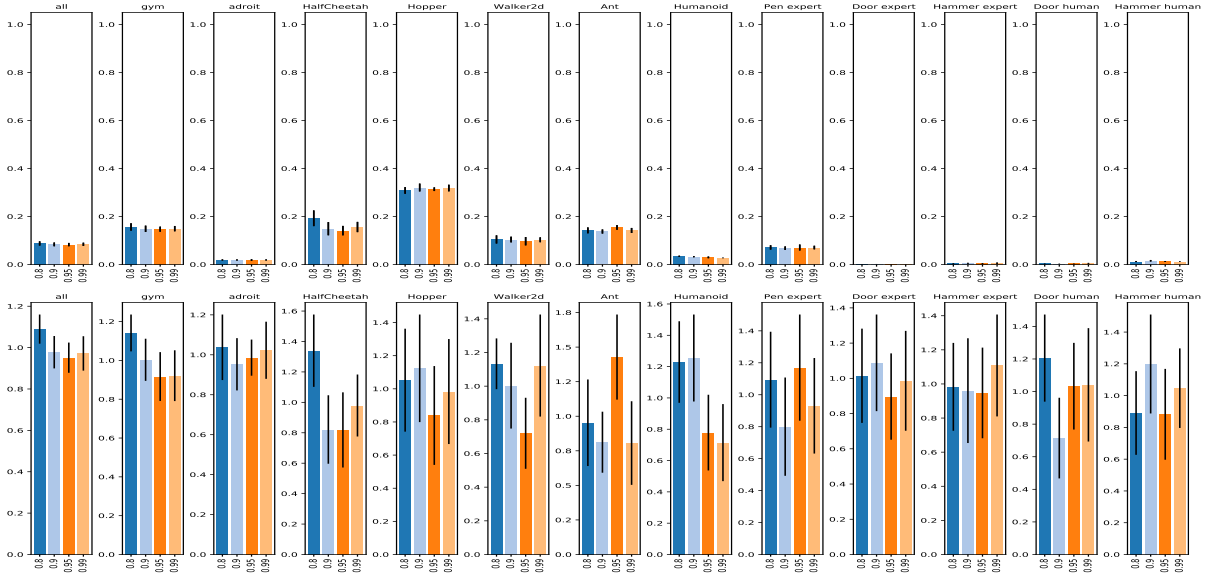


Figure A.15: Analysis of choice `GAE` λ (C15): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

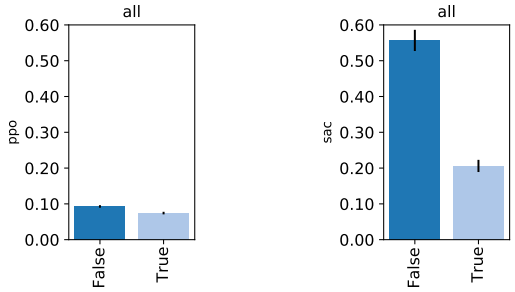


Figure A.16: 95th percentile of performance scores conditioned on `RL Algorithm (C8)`(subplots) and `subtract log-pi (C40)`(bars).

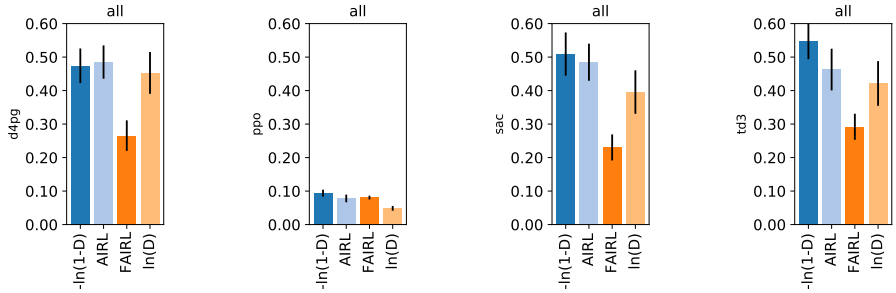


Figure A.17: 95th percentile of performance scores conditioned on `RL Algorithm (C8)`(subplots) and `reward function (C30)`(bars).

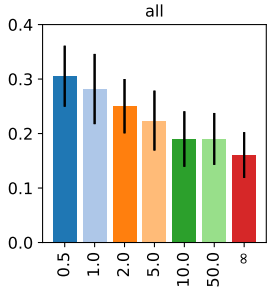


Figure A.18: 95th percentile of performance scores conditioned on `max reward magnitude (C31)` and `reward function (C30)=FAIRL`.

A.5 Experiment main

A.5.1 Design

For each of the 10 tasks, we sampled 25334 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `RL Algorithm (C8)`: {d4pg, sac, td3}
  - For the case "`RL Algorithm (C8)` = sac", we further sampled the sub-choices:
    - `SAC learning rate (C17)`: {0.0001, 0.0003, 0.001}
    - `SAC entropy per dimension (C16)`: {-2.0, -1.0, -0.5, 0.0}
    - `SAC polyak` τ `(C18)`: {0.001, 0.003, 0.01, 0.03}
  - For the case "`RL Algorithm (C8)` = d4pg", we further sampled the sub-choices:
    - `D4PG learning rate (C26)`: {3e-05, 0.0001, 0.0003}
    - `behavioral policy noise (C21)`: {0.1, 0.2, 0.3, 0.5}
    - `VMax (C24)`: {150.0, 750.0, 1500.0}
    - `number of atoms (C23)`: {51.0, 101.0, 201.0, 401.0}
    - `N-step returns (C25)`: {1.0, 3.0, 5.0}
  - For the case "`RL Algorithm (C8)` = td3", we further sampled the sub-choices:
    - `TD3 policy learning rate (C19)`: {0.0001, 0.0003, 0.001}
    - `TD3 critic learning rate (C20)`: {0.0001, 0.0003, 0.001}
    - `TD3 gradient clipping (C22)`: {40.0, ∞}
    - `behavioral policy noise (C21)`: {0.1, 0.2, 0.3, 0.5}
- `RL replay buffer size (C28)`: {300000, 1000000, 3000000}
- `policy MLP depth (C1)`: {1, 2, 3}
- `policy MLP width (C2)`: {64, 128, 256, 512}
- `critic MLP depth (C3)`: {2, 3}
- `critic MLP width (C4)`: {256, 512}
- `RL activation (C5)`: {relu, tanh}
- `discount` γ `(C6)`: {0.97, 0.99}
- `BC pretraining (C34)`: {False, True}
- `absorbing state (C32)`: {False, True}
- `discriminator replay buffer size (C43)`: {300000, 1000000, 3000000}
- `reward shaping (C39)`: {False, True}
- `discriminator input (C35)`: {s, sa, sas, ss}

- `discriminator MLP depth (C36)`: {1, 2, 3}
- `discriminator MLP width (C37)`: {16, 32, 64, 128, 256, 512}
- `discriminator activation (C38)`: {elu, leaky_relu, relu, sigmoid, swish, tanh}
- `discriminator last layer init scale (C41)`: {0.001, 1.0}
- `discriminator regularizer (C45)`: {GP, Mixup, No regularizer, PUGAIL, dropout, entropy, spectral norm, weight decay}
  - For the case "`discriminator regularizer (C45)` = GP", we further sampled the sub-choices:
    - `gradient penalty` λ `(C47)`: {0.1, 1.0, 10.0}
    - `gradient penalty k (C46)`: {0.0, 1.0}
  - For the case "`discriminator regularizer (C45)` = Mixup", we further sampled the sub-choices:
    - `mixup` α `(C48)`: {0.1, 0.4, 1.0}
  - For the case "`discriminator regularizer (C45)` = PUGAIL", we further sampled the sub-choices:
    - `PUGAIL` η `(C49)`: {0.25, 0.5, 0.7}
    - `PUGAIL` β `(C50)`: {0.0, 0.7, ∞}
  - For the case "`discriminator regularizer (C45)` = entropy", we further sampled the sub-choices:
    - `entropy` λ `(C54)`: {0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3}
  - For the case "`discriminator regularizer (C45)` = weight decay", we further sampled the sub-choices:
    - `weight decay` λ `(C53)`: {0.3, 1.0, 3.0, 10.0, 30.0}
  - For the case "`discriminator regularizer (C45)` = dropout", we further sampled the sub-choices:
    - `dropout input rate (C52)`: {0.0, 0.25, 0.5, 0.75}
    - `dropout hidden rate (C51)`: {0.25, 0.5, 0.75}
- `observation normalization (C55)`: {fixed, none, online}
- `evaluation behavior policy type (C29)`: {average, mode, stochastic}
- `discriminator learning rate (C42)`: {1e-06, 3e-06, 1e-05, 3e-05, 0.0001, 0.0003}
- `reward function (C30)`: {-ln(1-D), AIRL, ln(D)}
- `batch size (C7)`: {256}
- `replay ratio (C27)`: {256}
- `discriminator to RL updates ratio (C44)`: {1}
- `number of combined batches (C56)`: {8}

A.5.2 Results

For each of the sampled choice configurations we compute the performance metric as described in Section 3.2. We report aggregate statistics of the experiment in Tables A.7–A.10 as well as training curves in Figure A.19. We further provide per-choice analyses in Figures A.32-A.66.

Table A.7: Quantiles of the *final* agent performance across HP configurations for OpenAI Gym tasks.

	Ant	HalfCheetah	Hopper	Humanoid	Walker2d
90%	0.90	1.07	1.18	0.51	0.99
95%	0.99	1.11	1.20	0.87	1.01
99%	1.07	1.17	1.23	1.01	1.04
Max	1.18	1.37	1.34	1.06	1.21

Table A.8: Quantiles of the *final* agent performance across HP configurations for Adroit tasks.

	Door expert	Door human	Hammer expert	Hammer human	Pen expert
90%	0.72	0.25	1.08	0.46	0.74
95%	0.91	0.83	1.26	1.15	0.89
99%	1.04	2.29	1.37	3.04	1.11
Max	1.16	3.73	1.45	5.55	1.44

Table A.9: Quantiles of the *average* agent performance during training across HP configurations for OpenAI Gym tasks.

	Ant	HalfCheetah	Hopper	Humanoid	Walker2d
90%	0.61	0.82	0.93	0.29	0.70
95%	0.72	0.87	0.98	0.53	0.76
99%	0.85	0.94	1.06	0.79	0.84
Max	0.96	1.05	1.10	0.92	0.92

Table A.10: Quantiles of the *average* agent performance during training across HP configurations for Adroit tasks.

	Door expert	Door human	Hammer expert	Hammer human	Pen expert
90%	0.42	0.30	0.59	0.42	0.56
95%	0.57	0.56	0.77	0.70	0.66
99%	0.74	1.04	0.96	1.23	0.84
Max	0.92	2.08	1.18	3.42	1.09

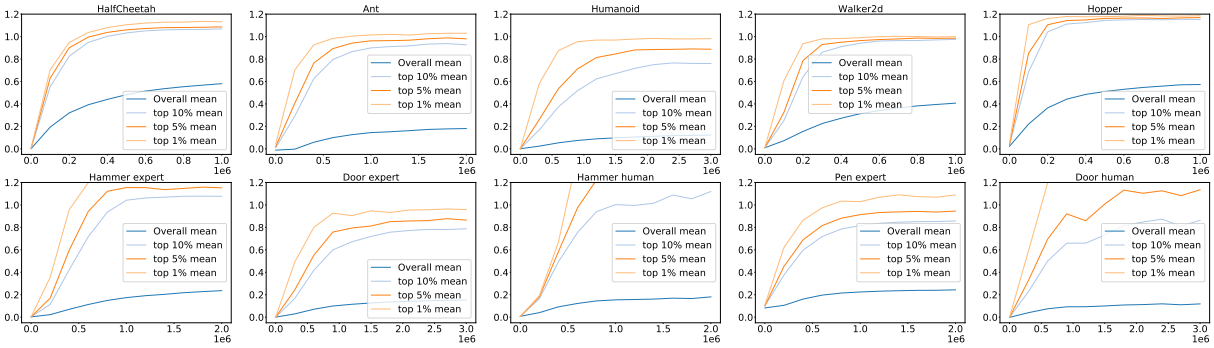


Figure A.19: Training curves.

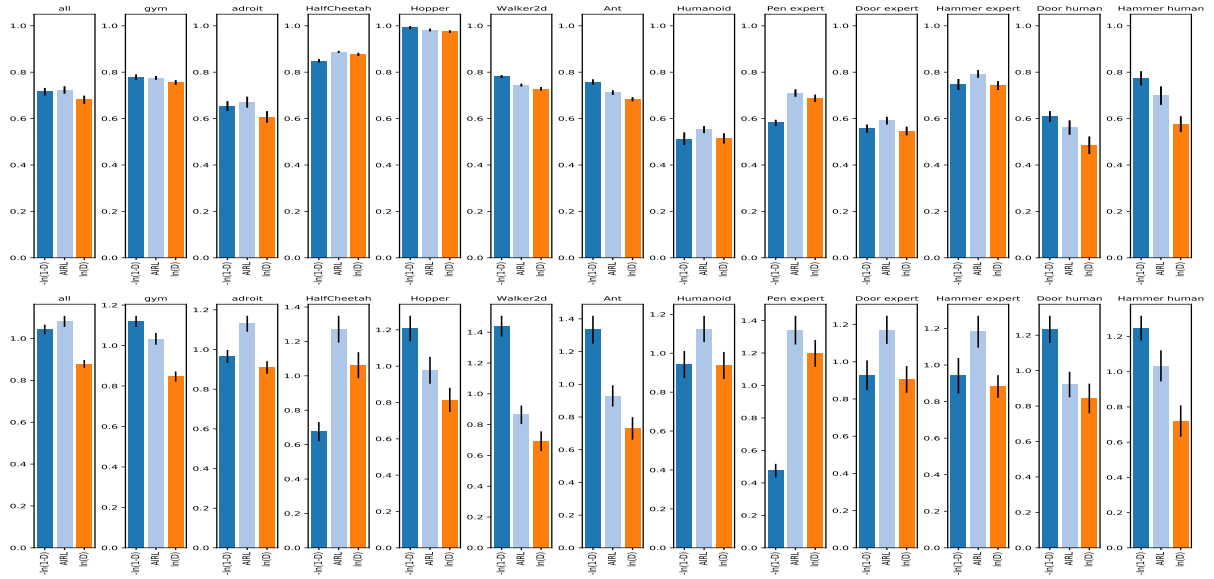


Figure A.20: Analysis of choice `reward function` (C30): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

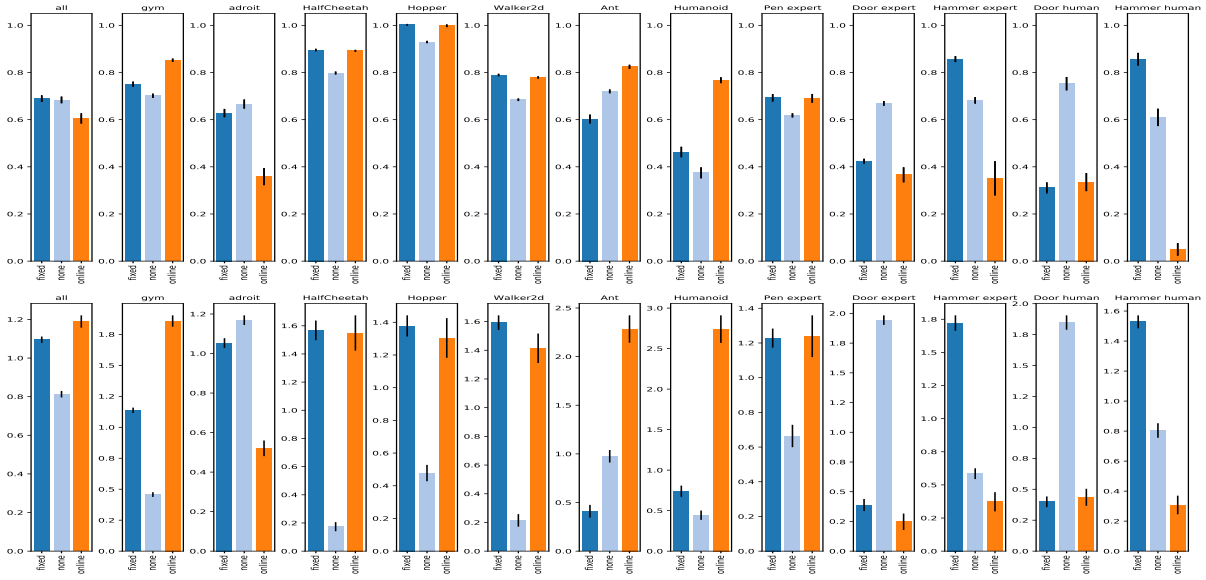


Figure A.21: Analysis of choice `observation normalization` (C55): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

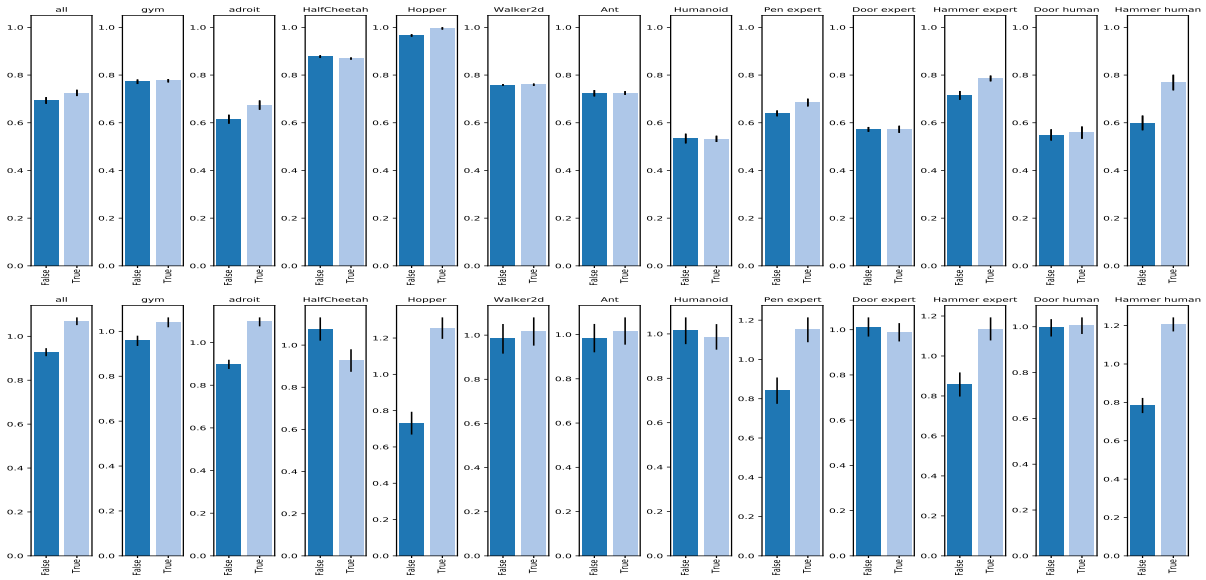


Figure A.22: Analysis of choice `BC pretraining` (C34): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

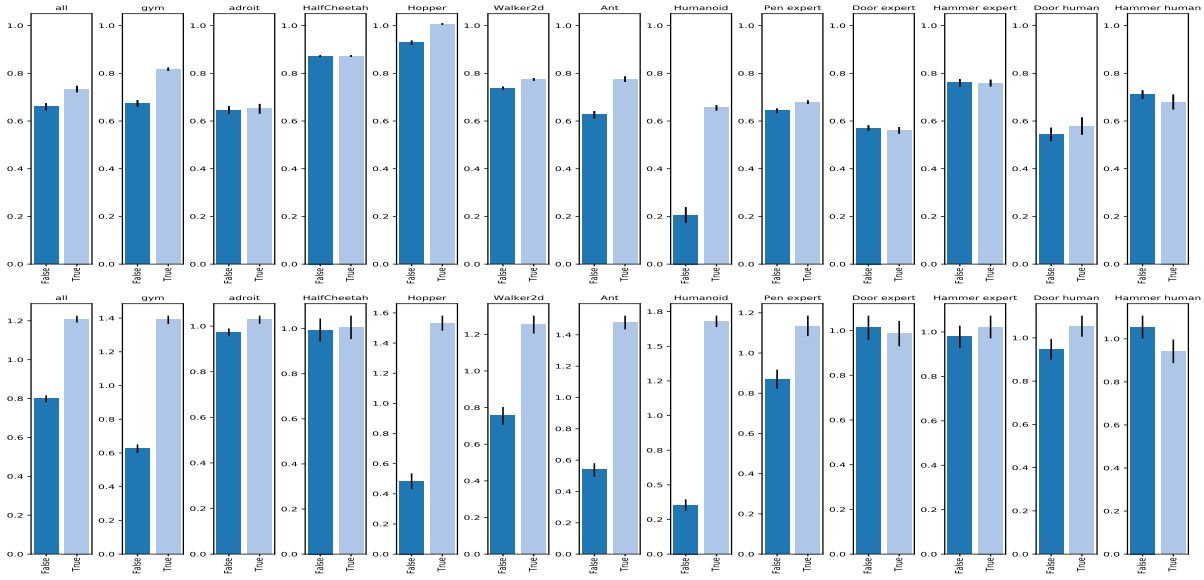


Figure A.23: Analysis of choice `absorbing state` (C32): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

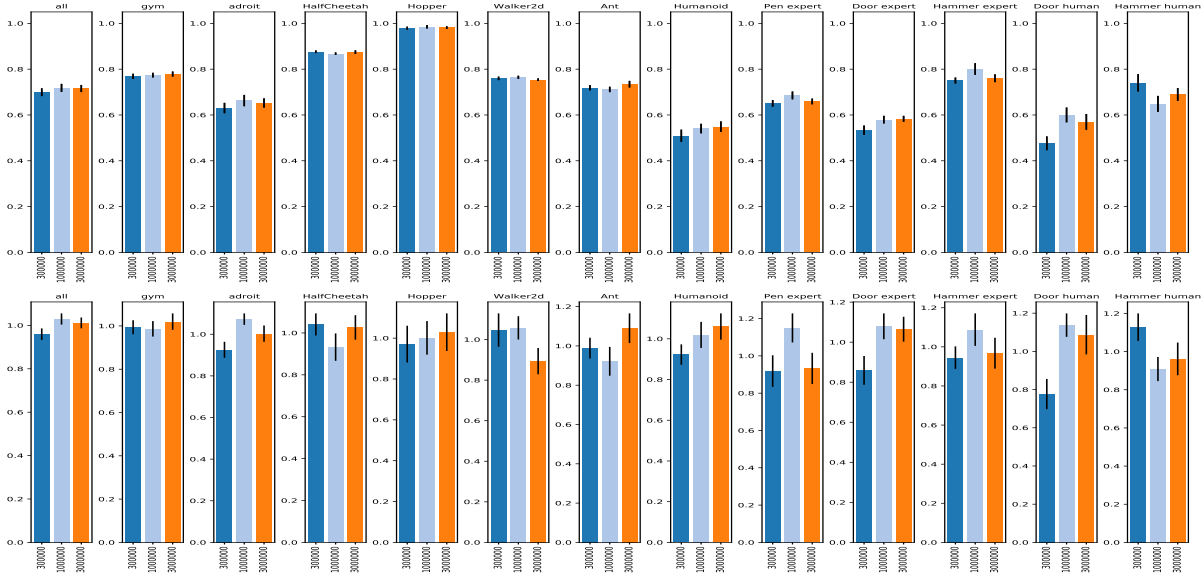


Figure A.24: Analysis of choice `RL replay buffer size` (C28): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

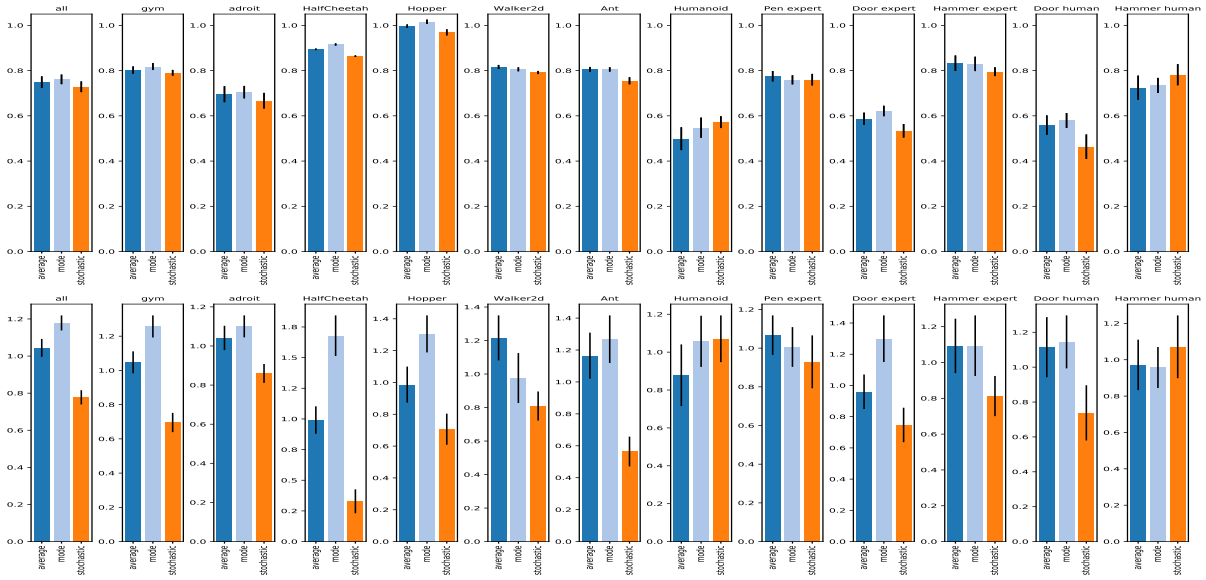


Figure A.25: Analysis of choice `evaluation behavior policy type (C29)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

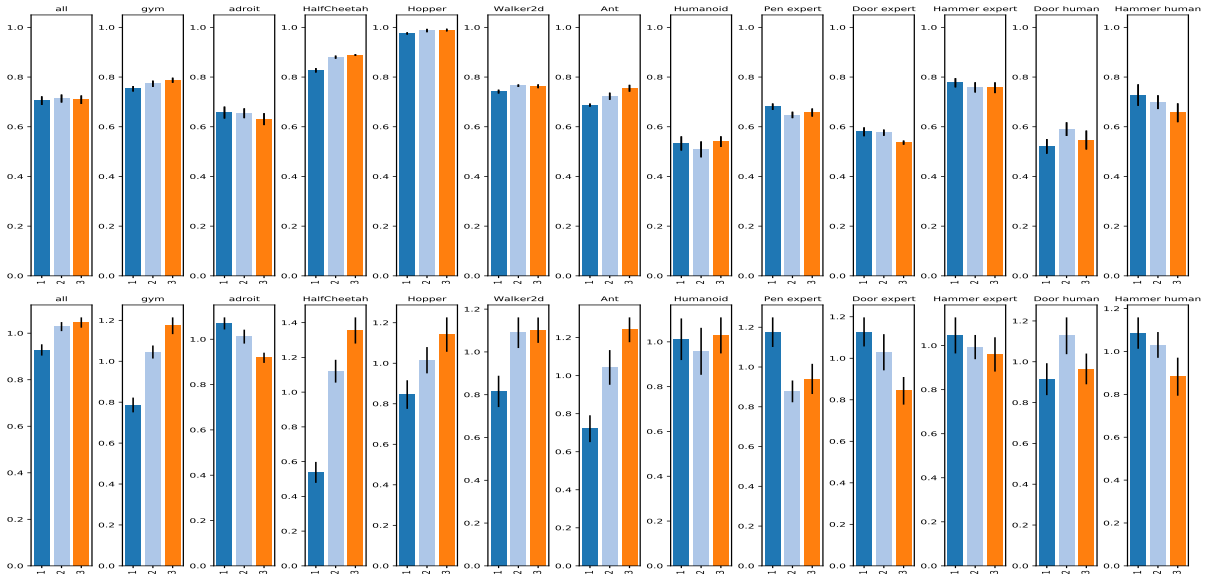


Figure A.26: Analysis of choice `policy MLP depth (C1)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

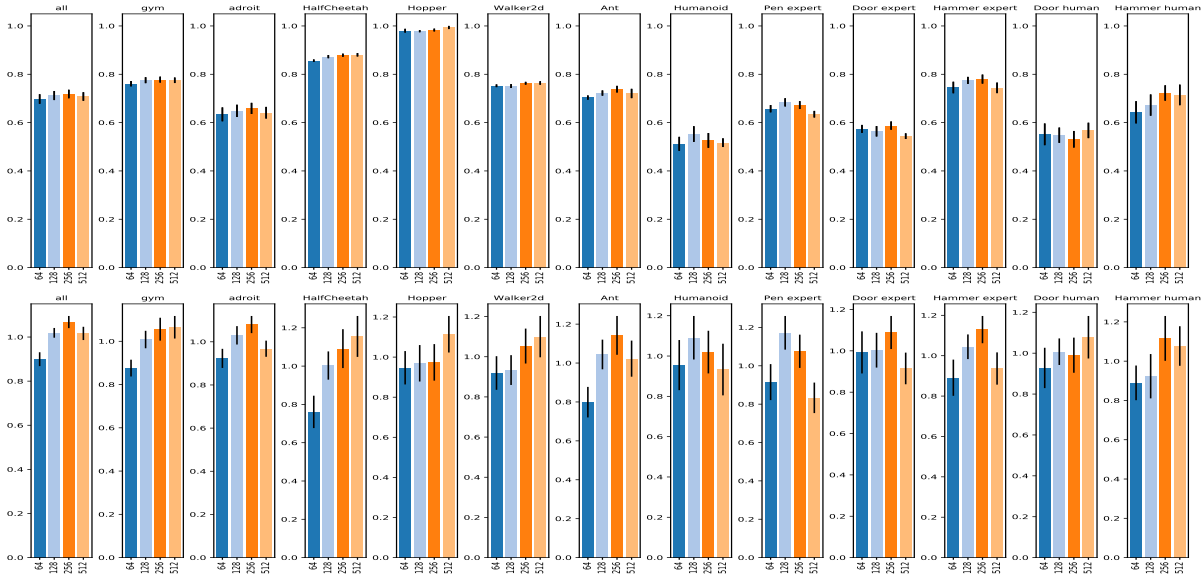


Figure A.27: Analysis of choice `policy MLP width` (C2): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

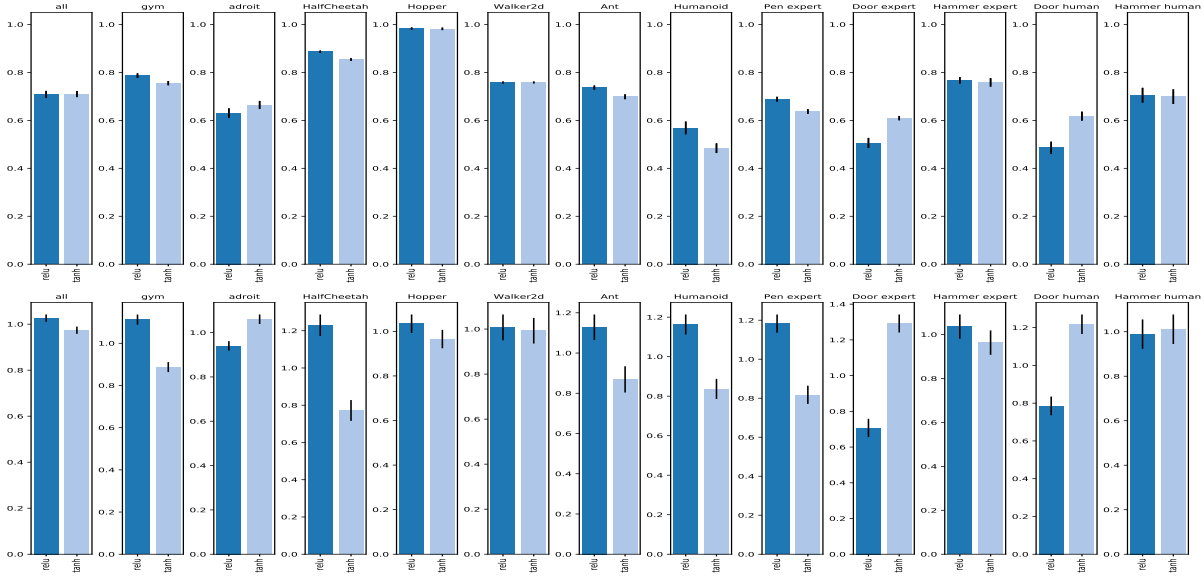


Figure A.28: Analysis of choice `RL activation` (C5): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

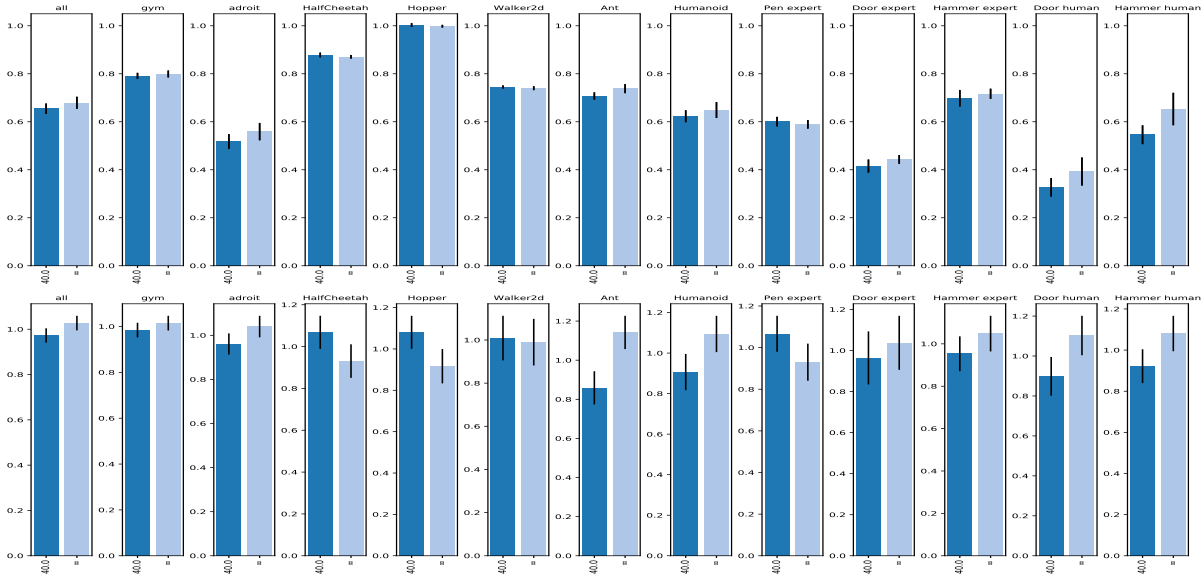


Figure A.29: Analysis of choice `TD3 gradient clipping` (C22): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

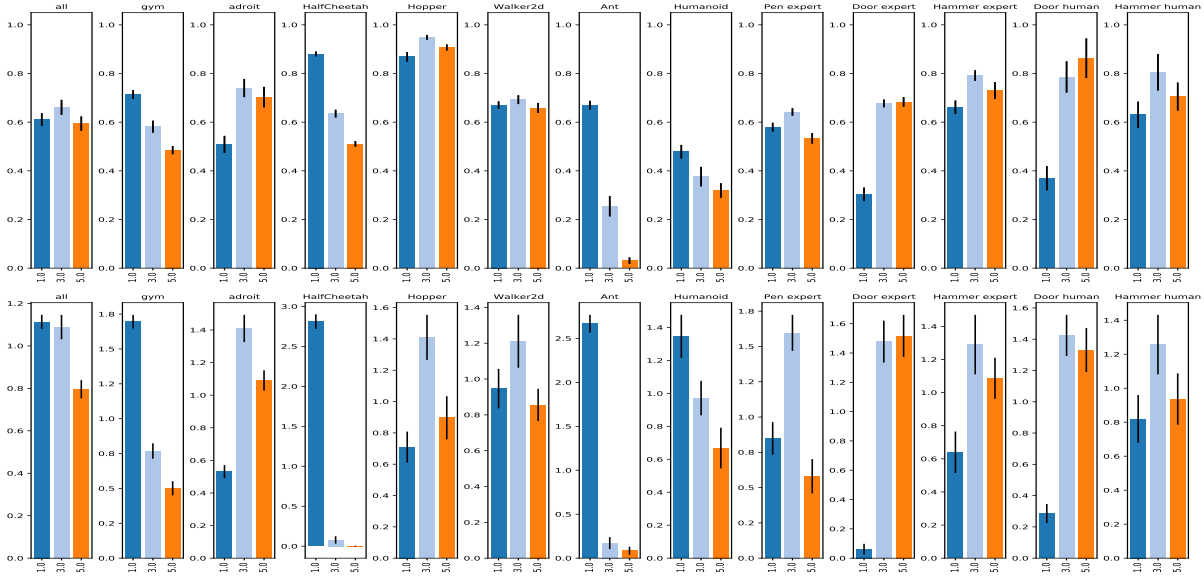


Figure A.30: Analysis of choice `N-step returns` (C25): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

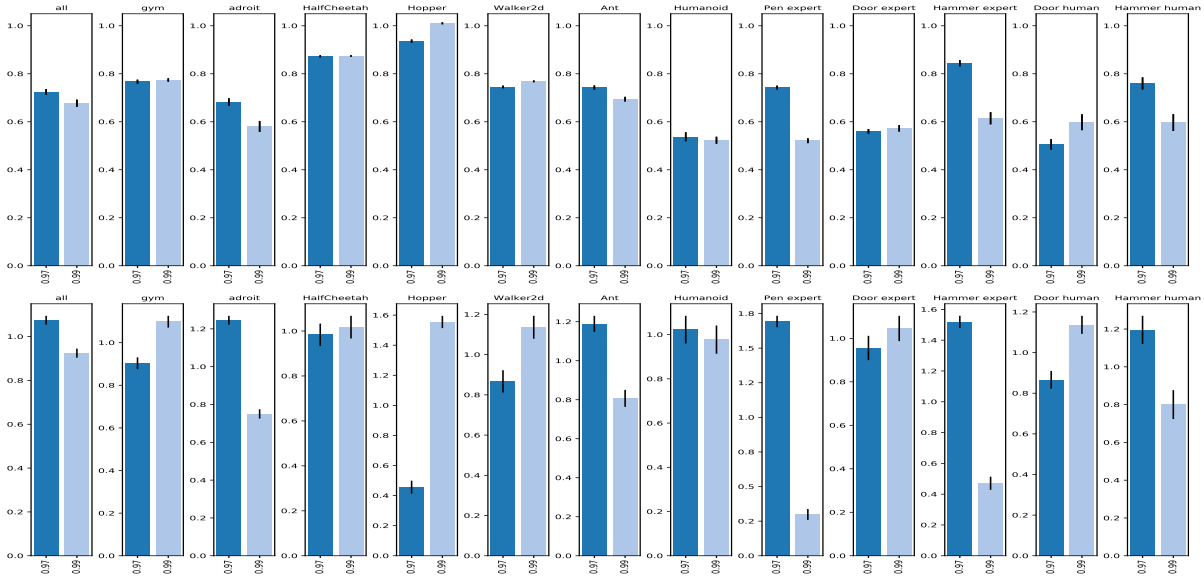


Figure A.31: Analysis of choice `discount` γ (C6): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

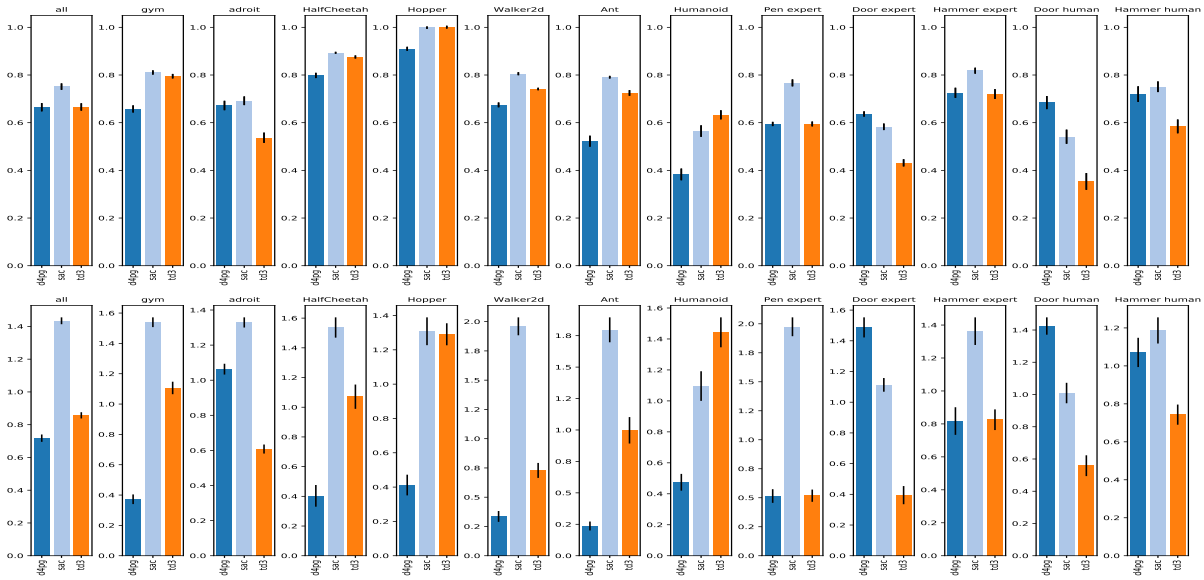


Figure A.32: Analysis of choice `RL Algorithm` (C8): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

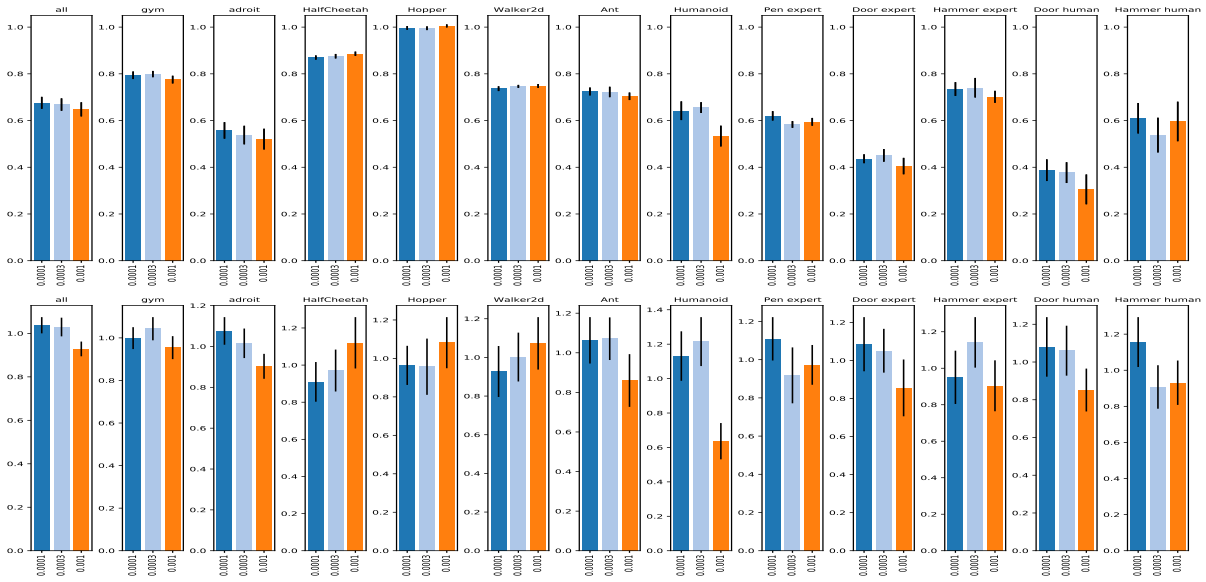


Figure A.33: Analysis of choice `TD3 policy learning rate` (C19): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

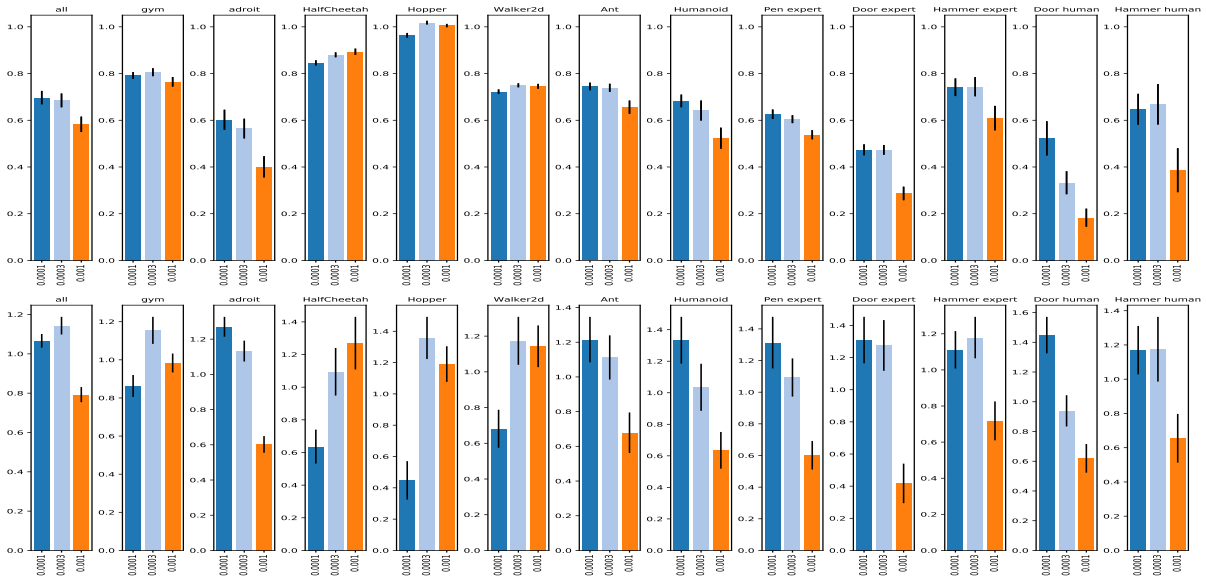


Figure A.34: Analysis of choice `TD3 critic learning rate` (C20): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

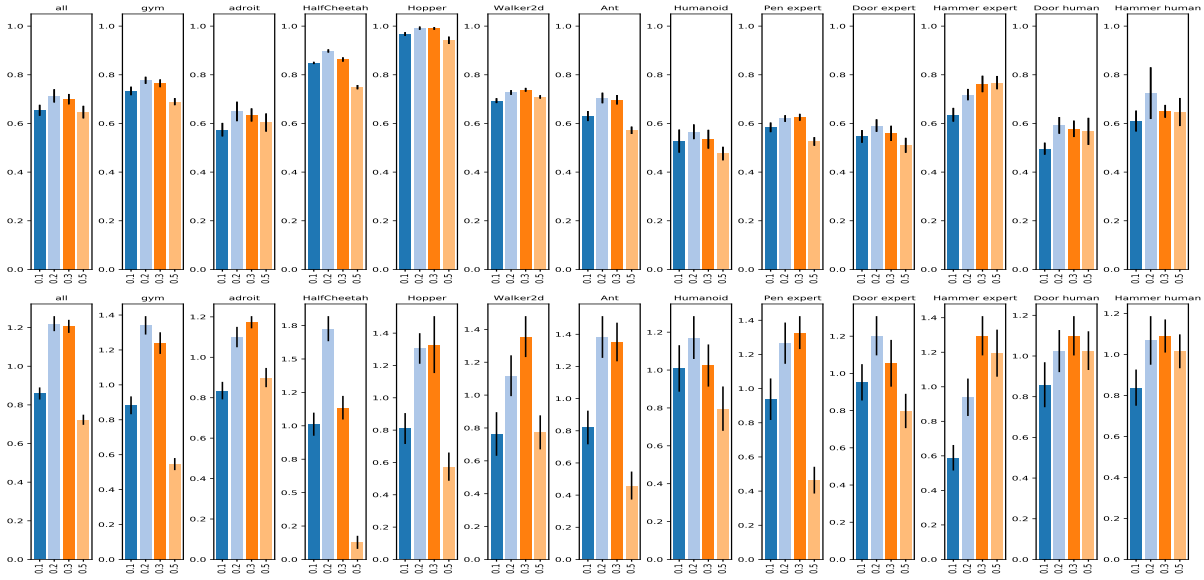


Figure A.35: Analysis of choice `behavioral policy noise` (C21): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

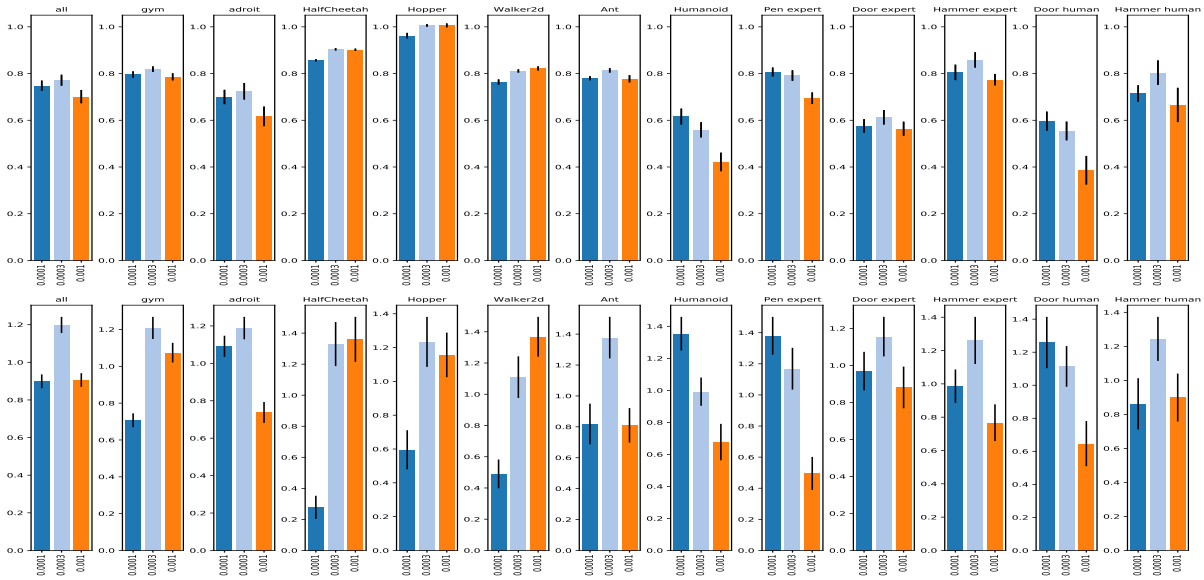


Figure A.36: Analysis of choice `SAC learning rate` (C17): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

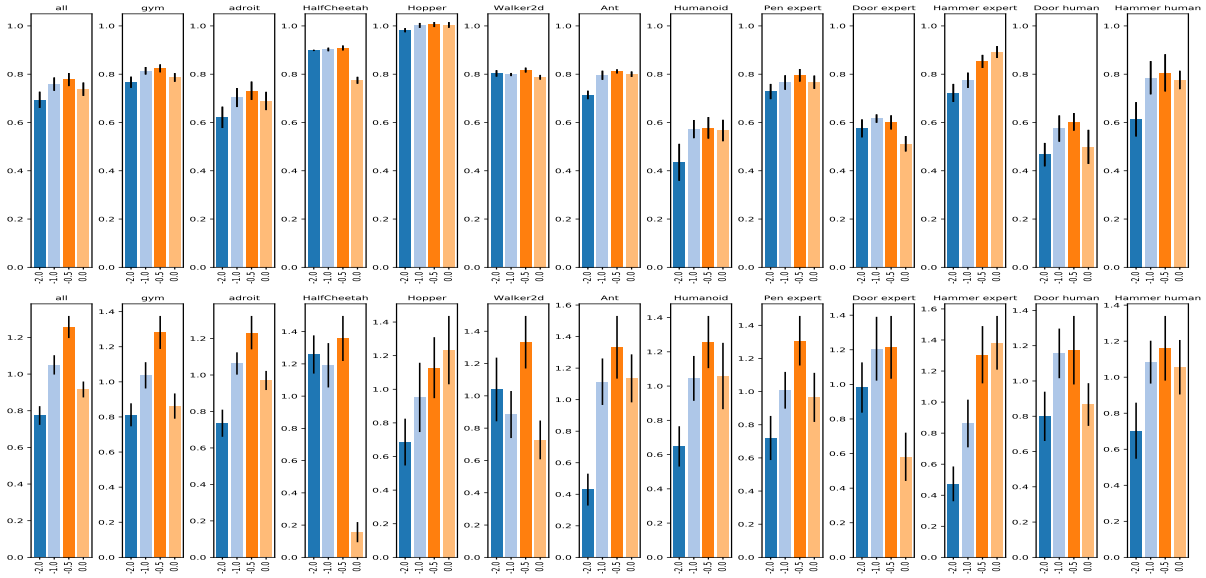


Figure A.37: Analysis of choice `SAC entropy per dimension` (C16): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

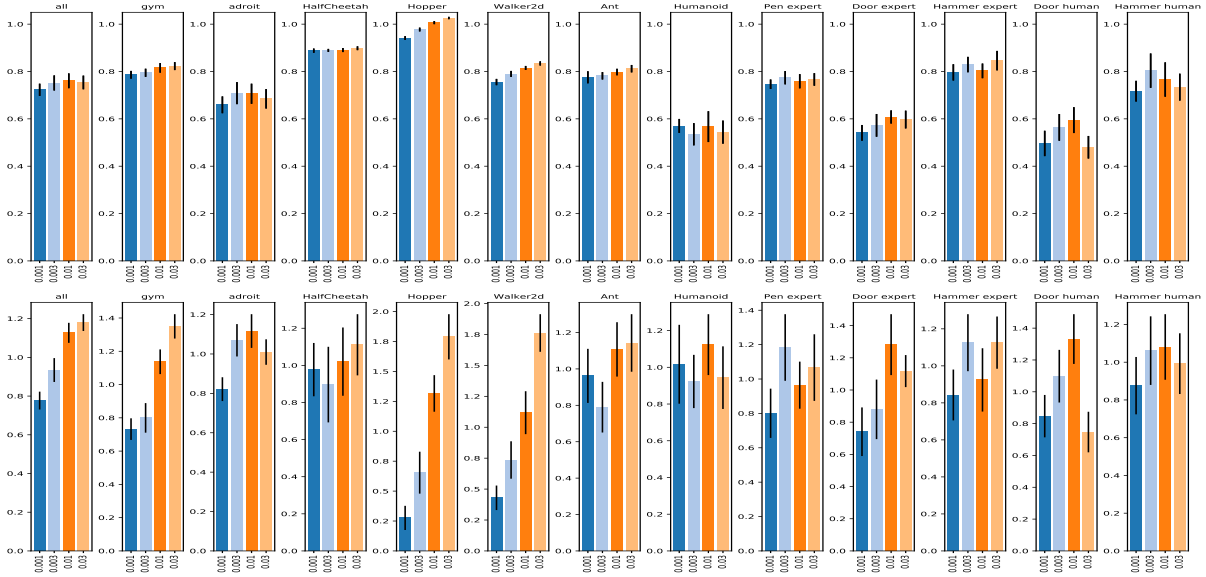


Figure A.38: Analysis of choice `SAC polyak` τ (C18): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

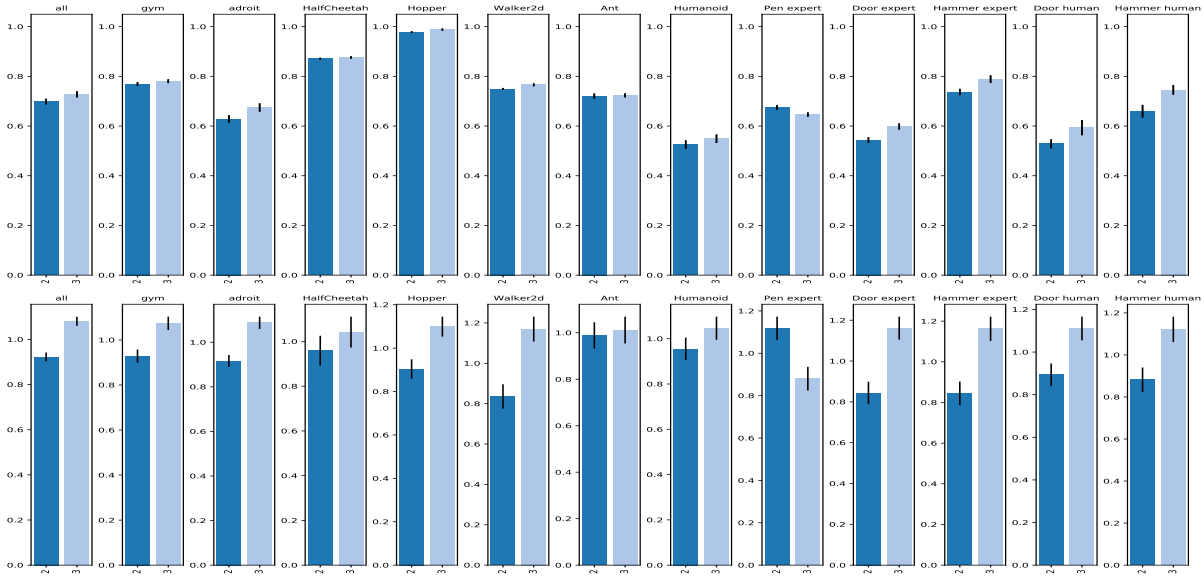


Figure A.39: Analysis of choice `critic MLP depth (C3)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

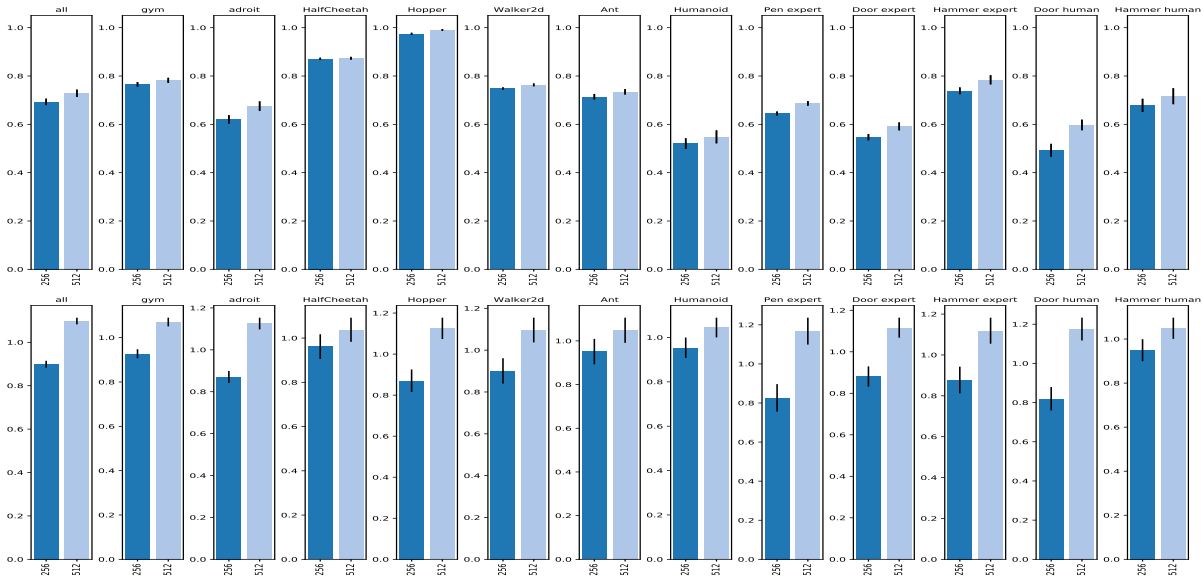


Figure A.40: Analysis of choice `critic MLP width (C4)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

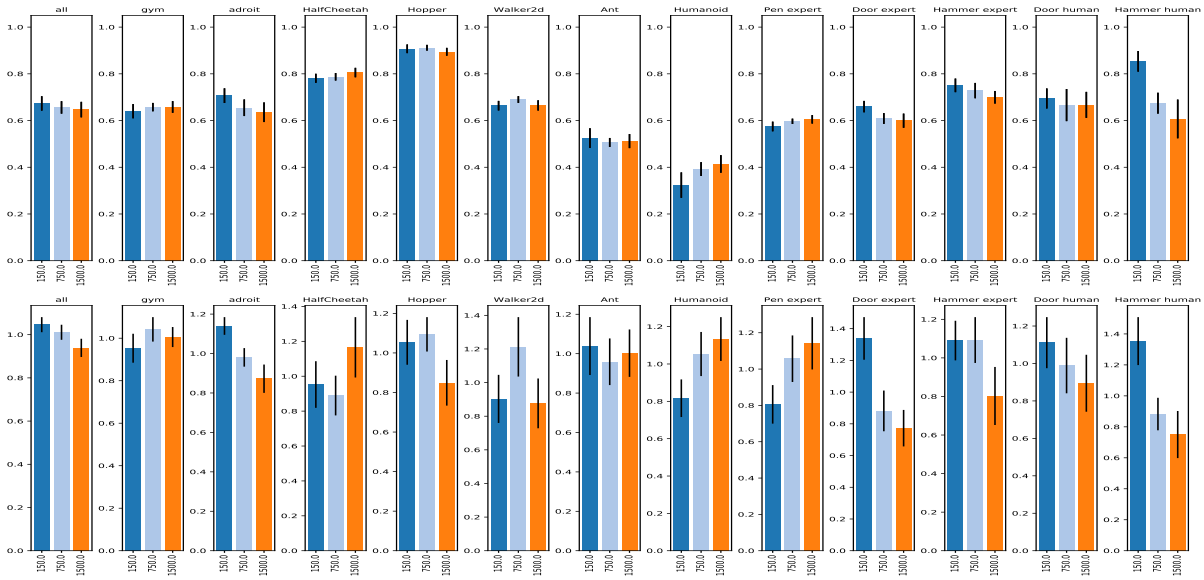


Figure A.41: Analysis of choice `VMax` (C24): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

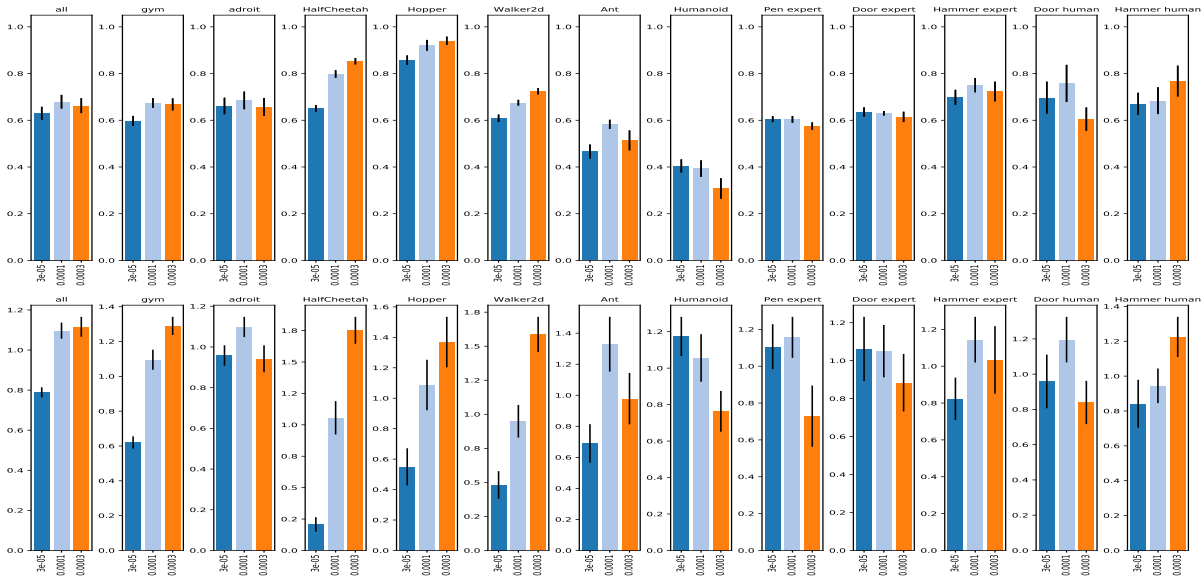


Figure A.42: Analysis of choice `D4PG learning rate` (C26): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

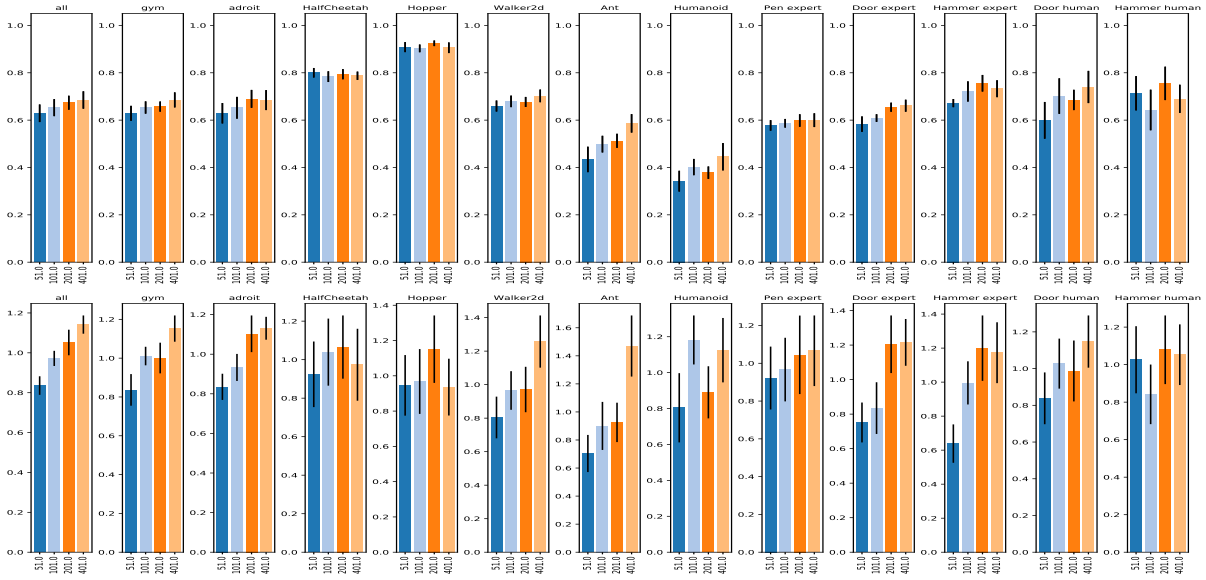


Figure A.43: Analysis of choice `number of atoms` (C23): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

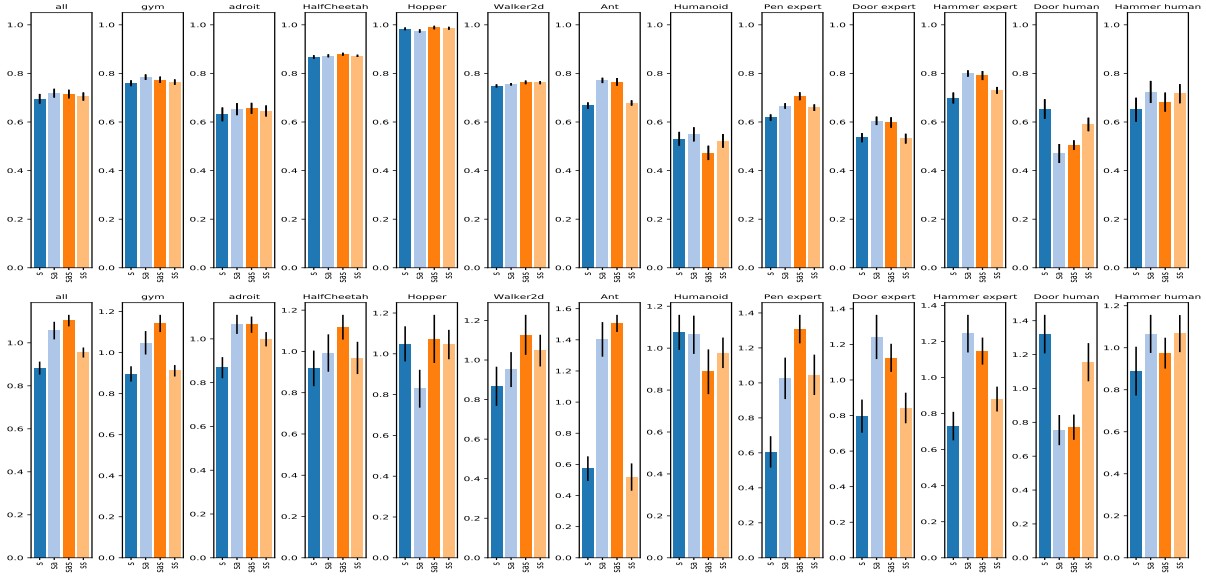


Figure A.44: Analysis of choice `discriminator input` (C35): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

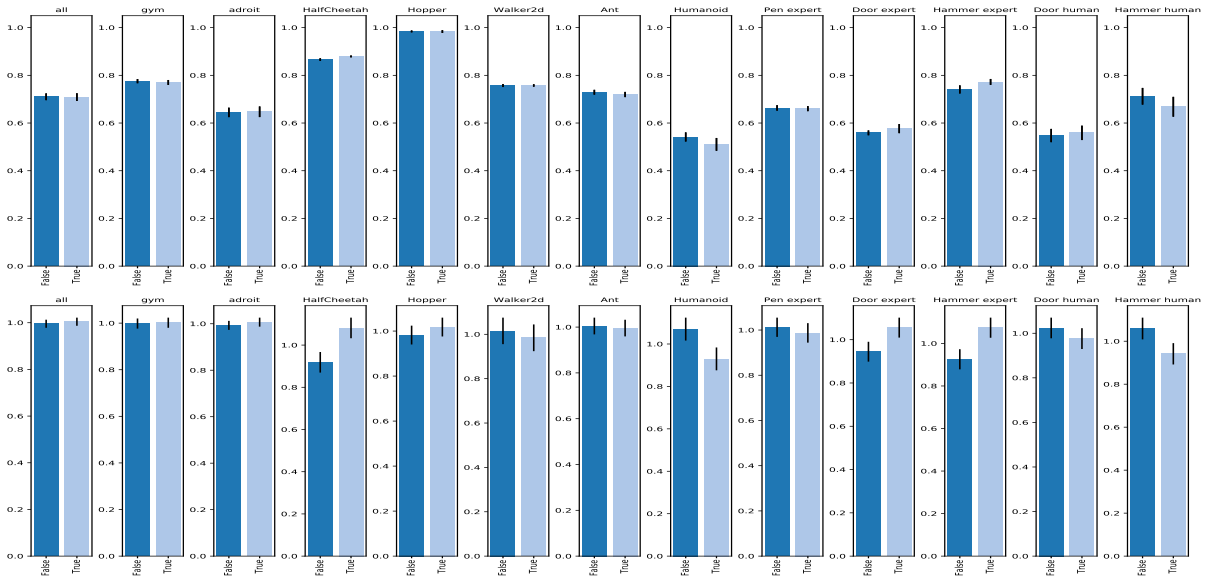


Figure A.45: Analysis of choice `reward shaping` (C39): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

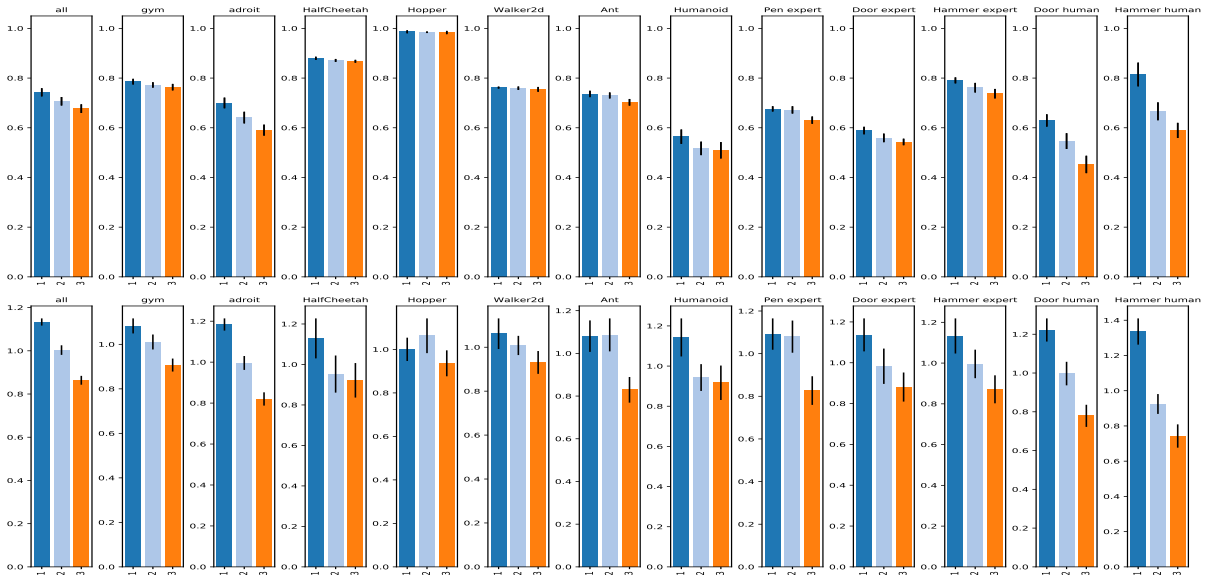


Figure A.46: Analysis of choice `discriminator MLP depth` (C36): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

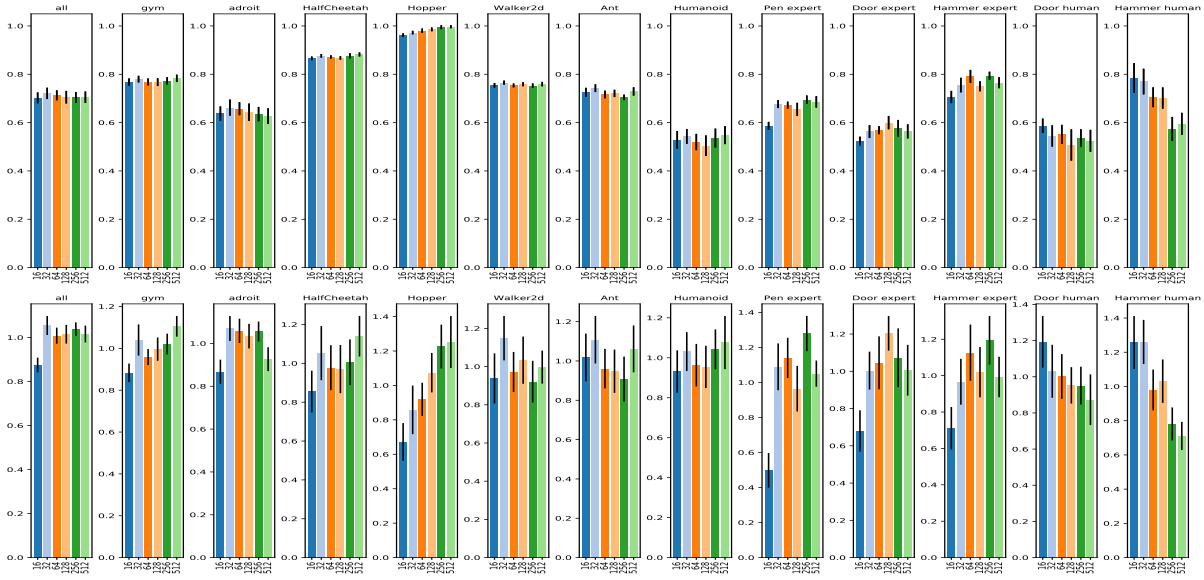


Figure A.47: Analysis of choice `discriminator MLP width (C37)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

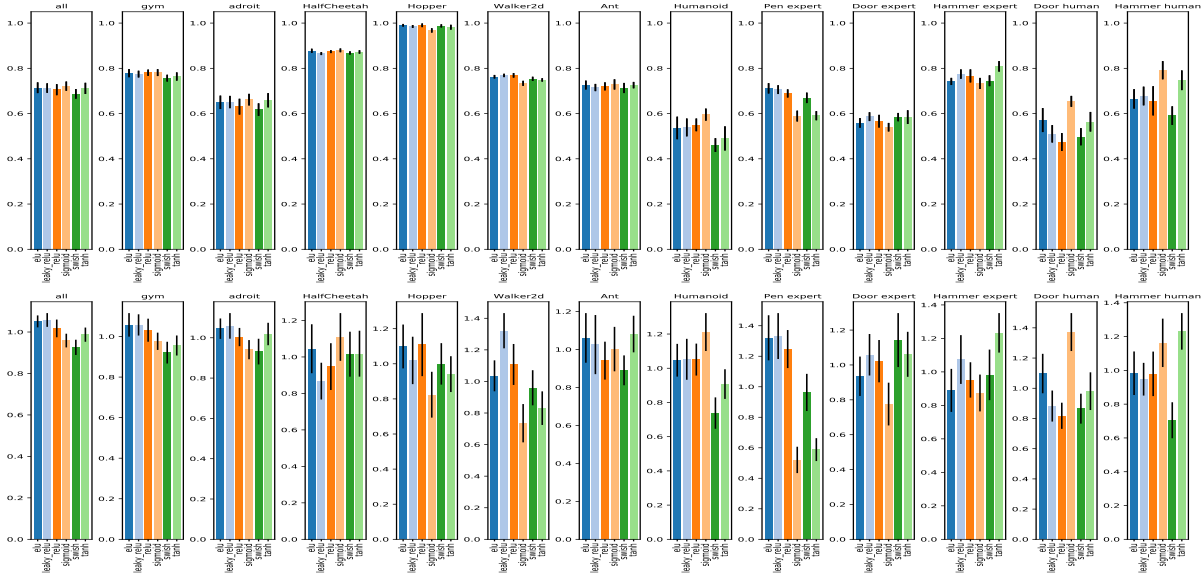


Figure A.48: Analysis of choice `discriminator activation (C38)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

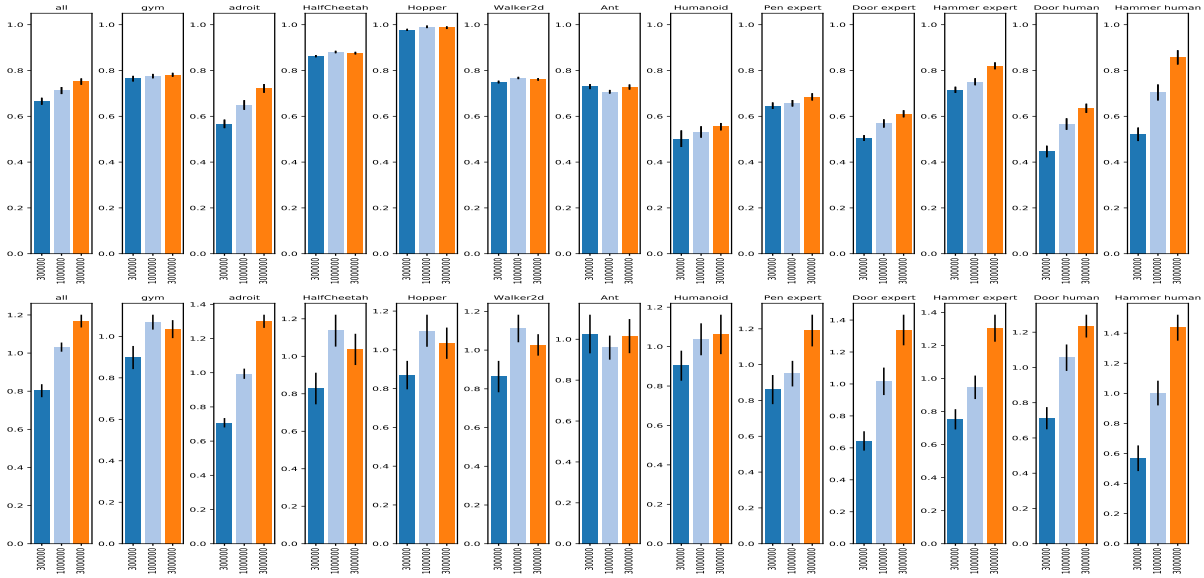


Figure A.49: Analysis of choice `discriminator replay buffer size` (C43): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

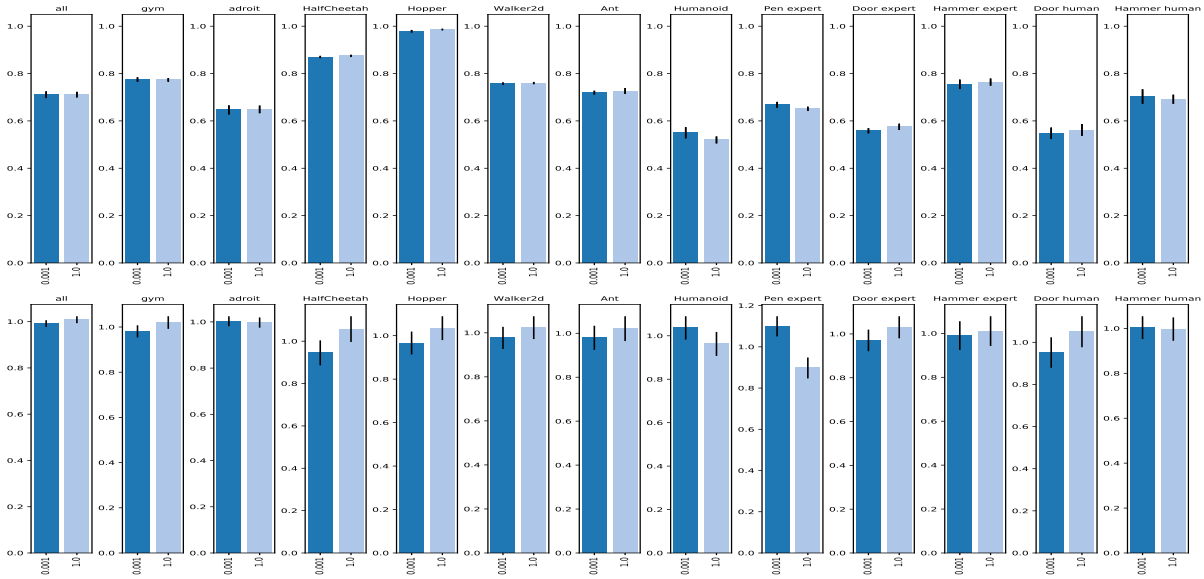


Figure A.50: Analysis of choice `discriminator last layer init scale` (C41): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

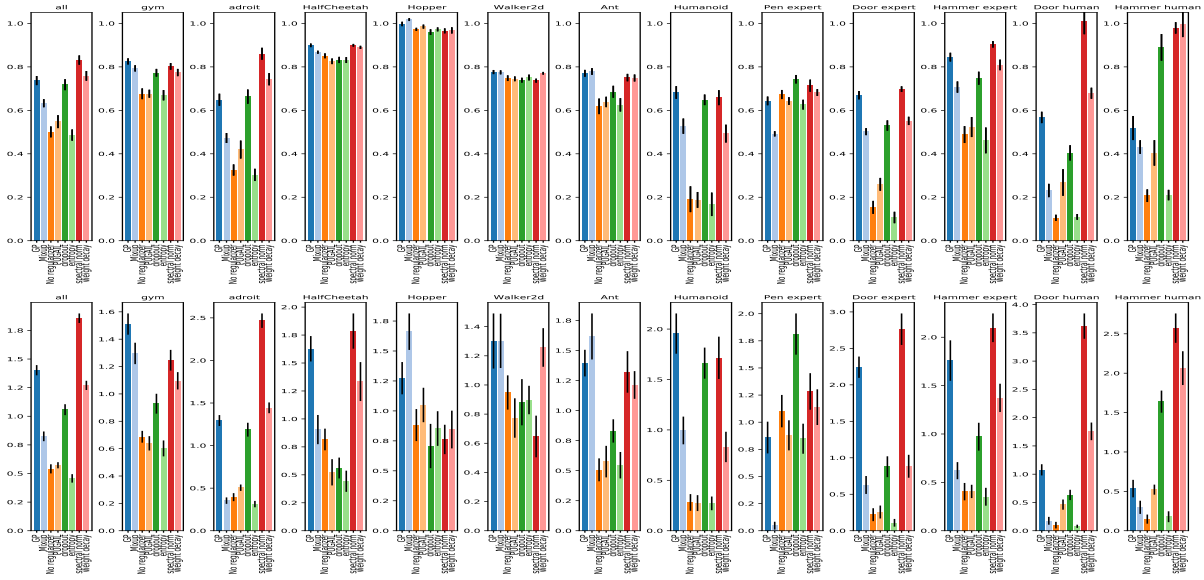


Figure A.51: Analysis of choice `discriminator regularizer` (C45): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

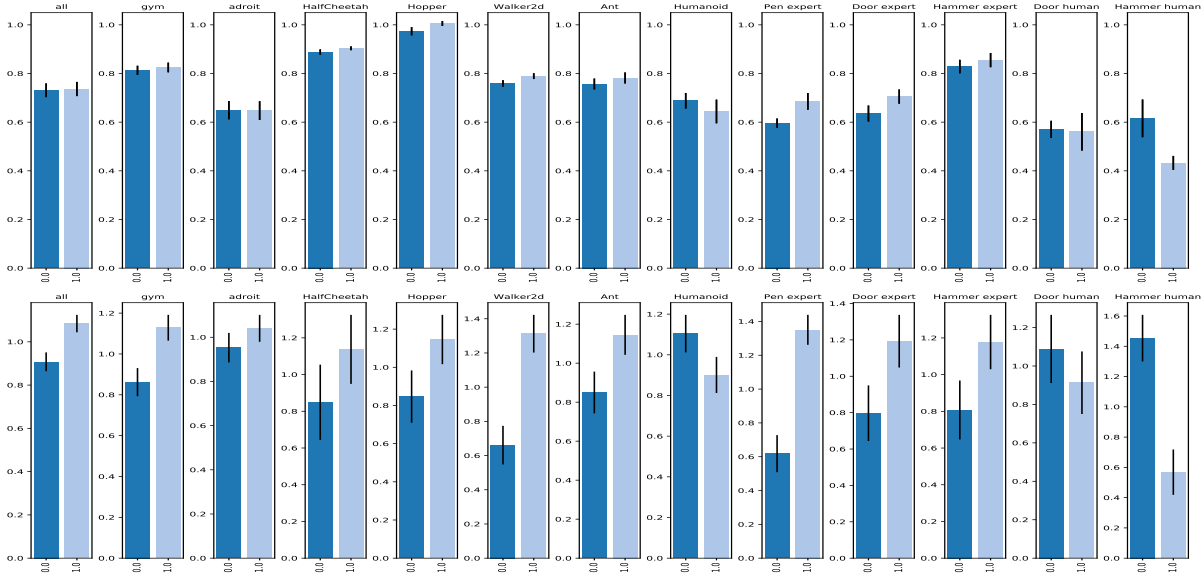


Figure A.52: Analysis of choice `gradient penalty k` (C46): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

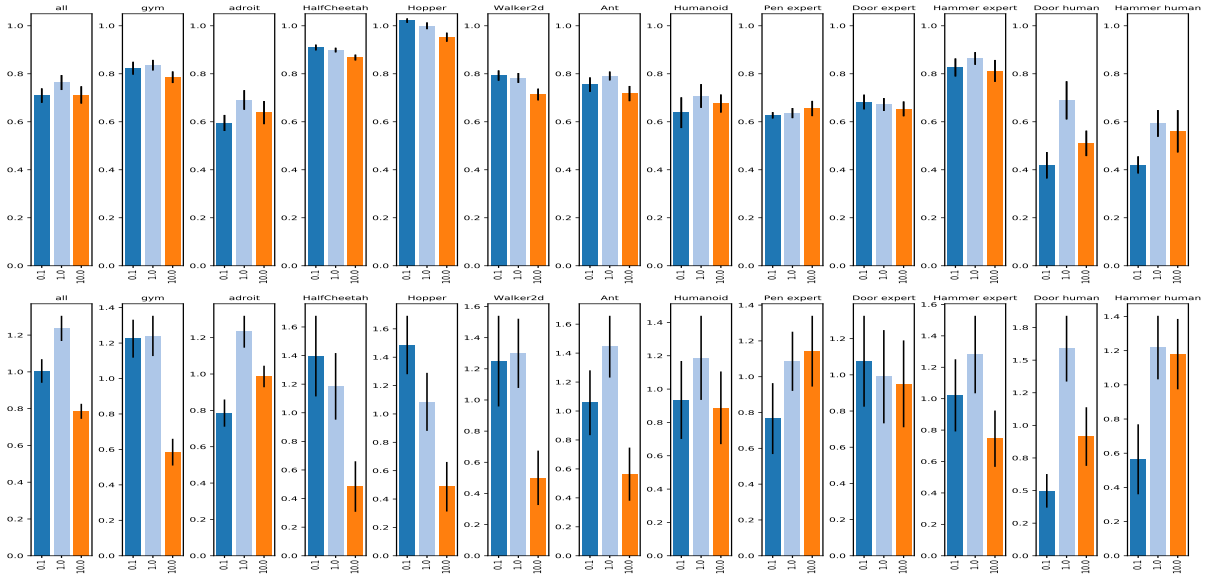


Figure A.53: Analysis of choice `gradient penalty` λ (C47): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

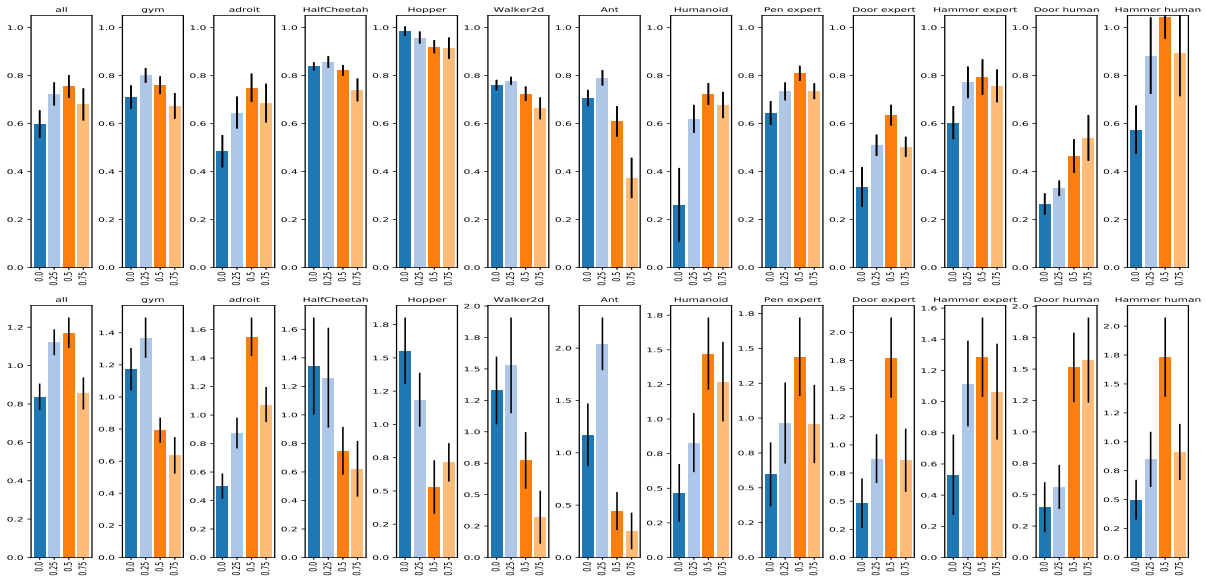


Figure A.54: Analysis of choice `dropout input rate` (C52): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

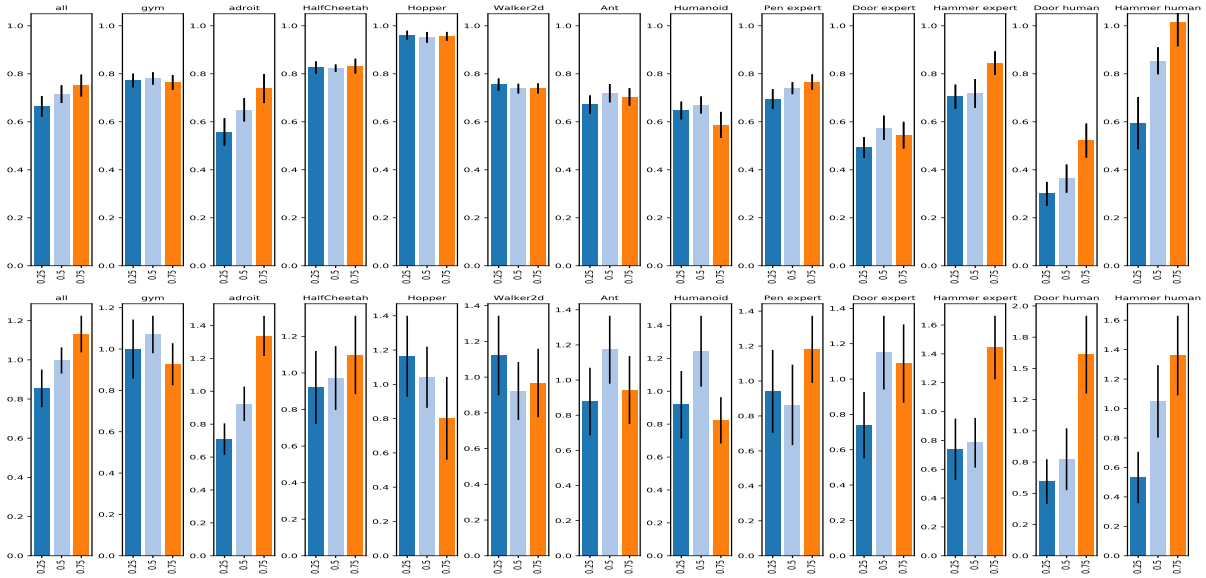


Figure A.55: Analysis of choice `dropout hidden rate` (C51): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

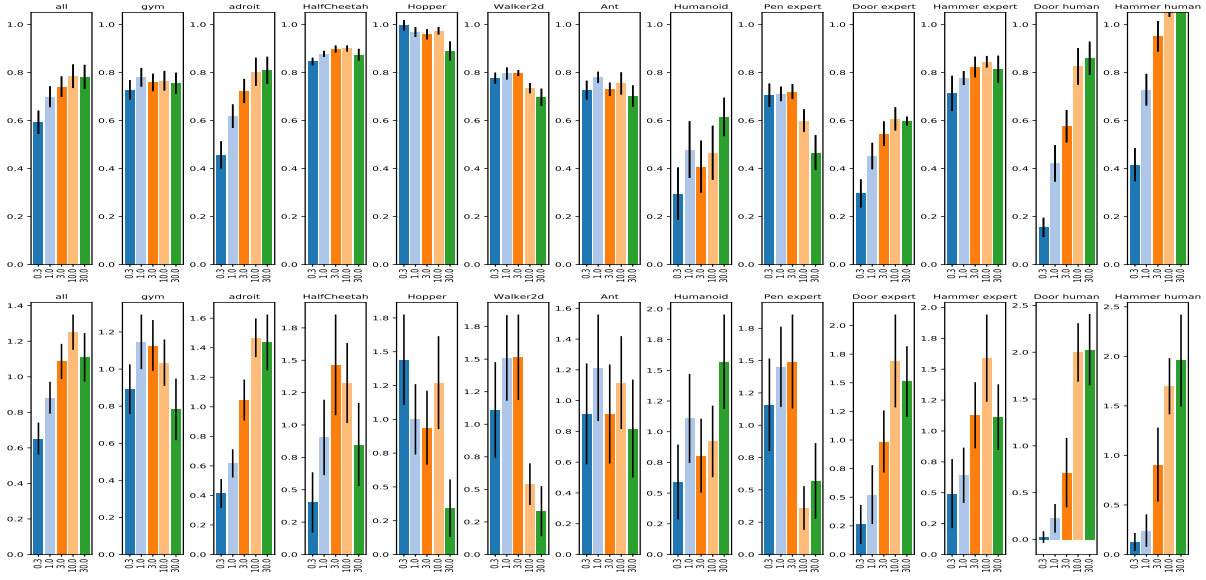


Figure A.56: Analysis of choice `weight decay` λ (C53): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

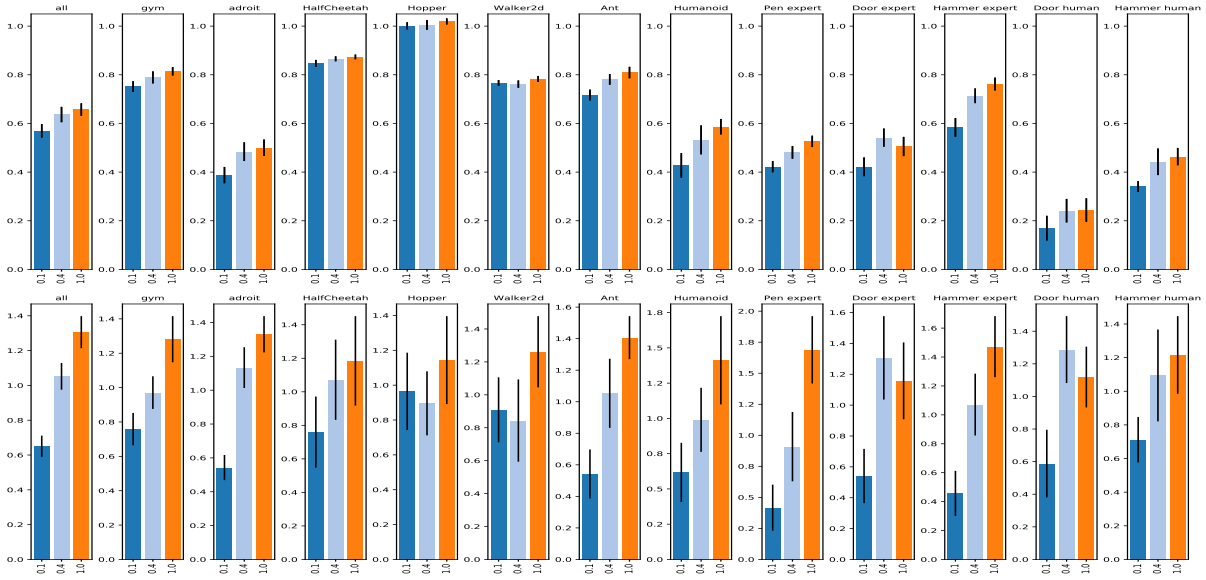


Figure A.57: Analysis of choice `mixup` α (C48): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

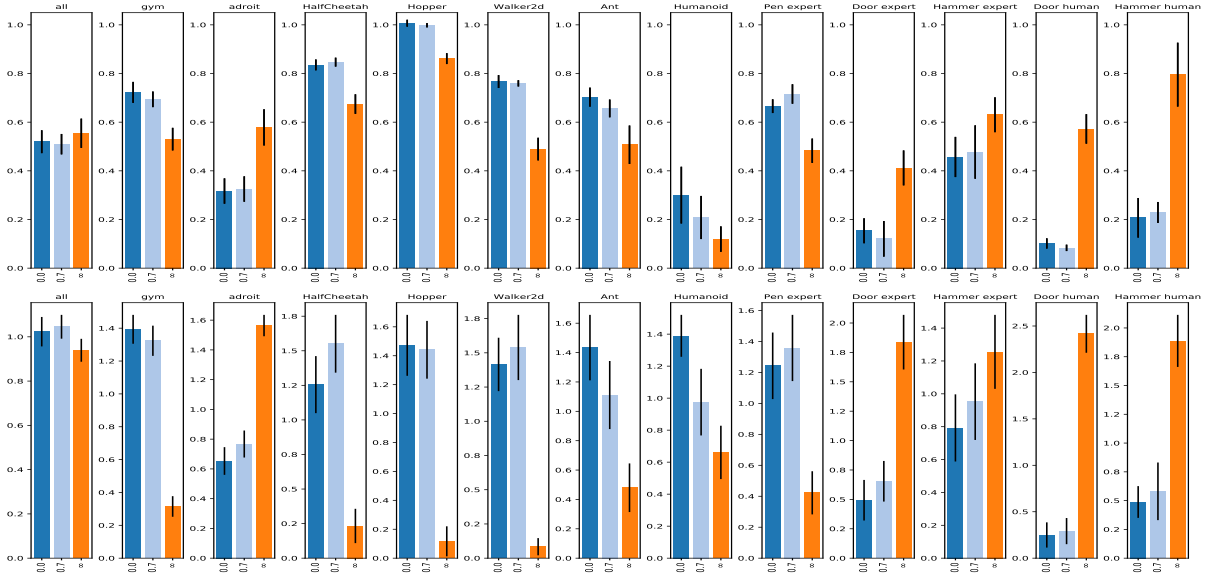


Figure A.58: Analysis of choice `PUGAIL` β (C50): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

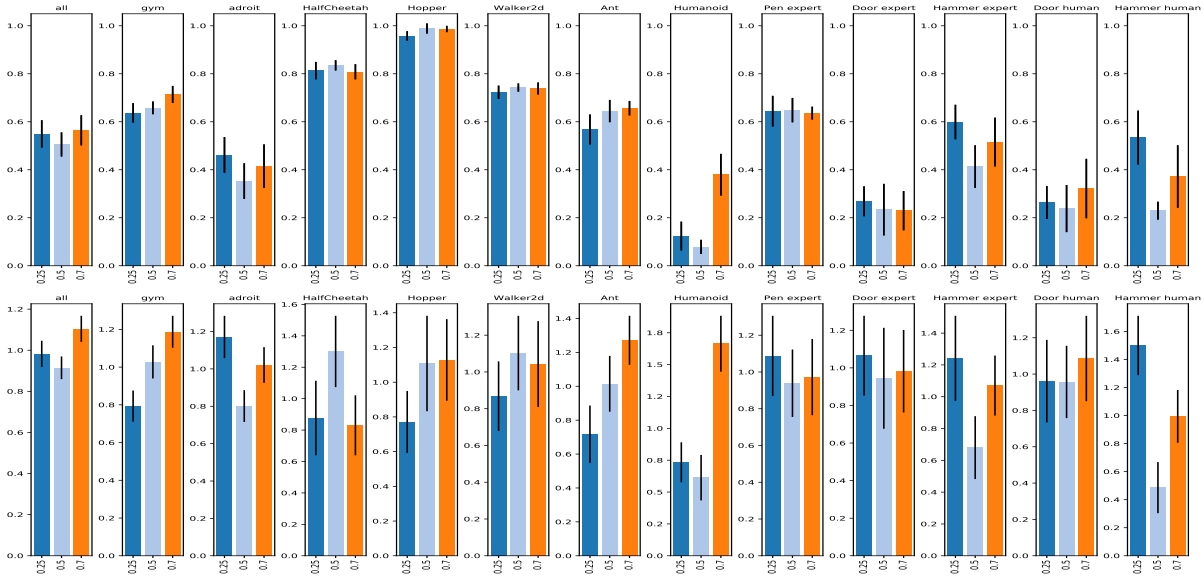


Figure A.59: Analysis of choice `PUGAIL` η (C49): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

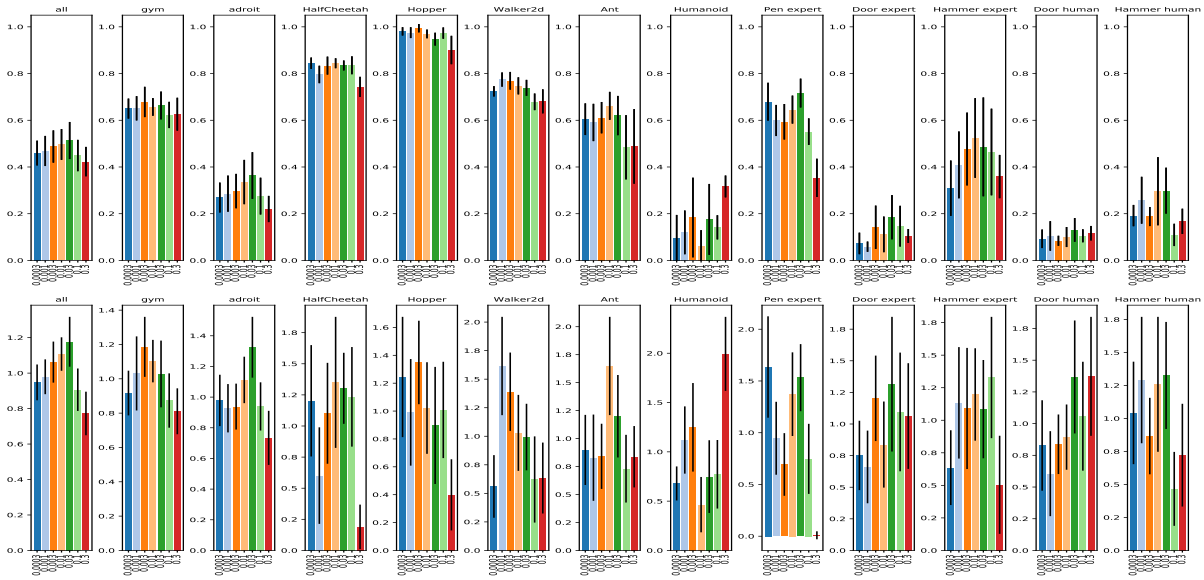


Figure A.60: Analysis of choice `entropy` λ (C54): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

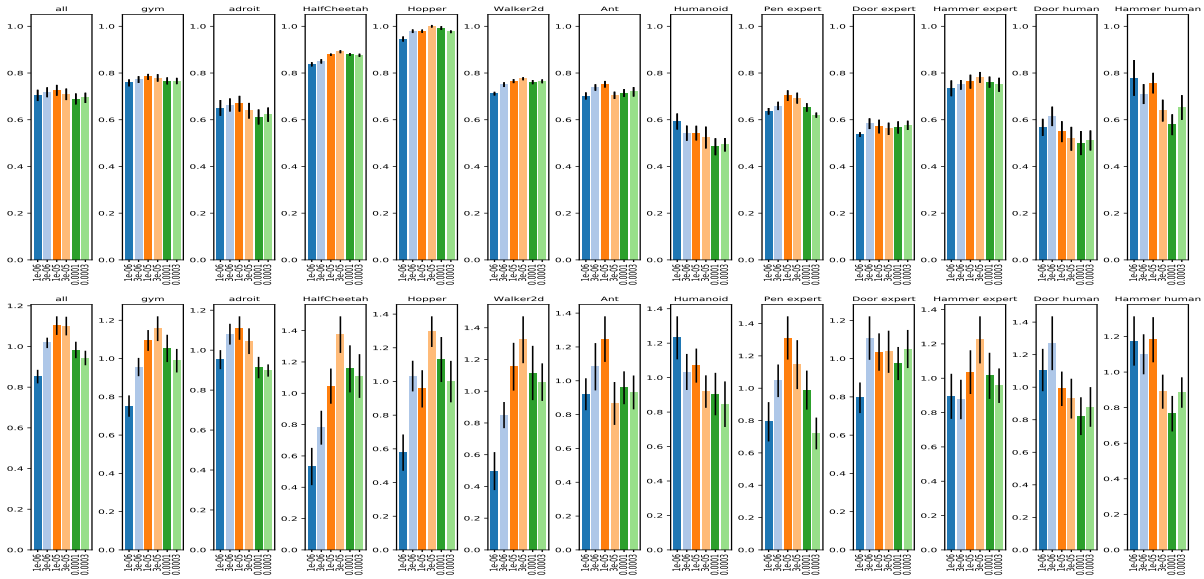


Figure A.61: Analysis of choice `discriminator learning rate` (C42): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

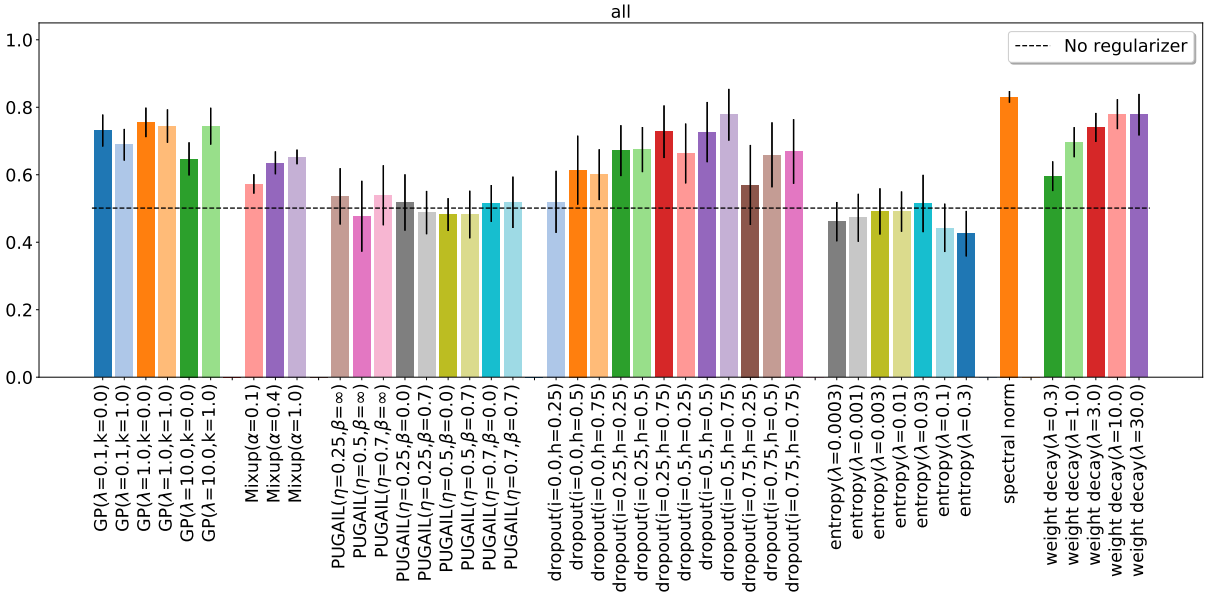


Figure A.62: 95th percentile of performance scores conditioned on `discriminator regularizer` (C45) and regularizers' HPs averaged across all environments.

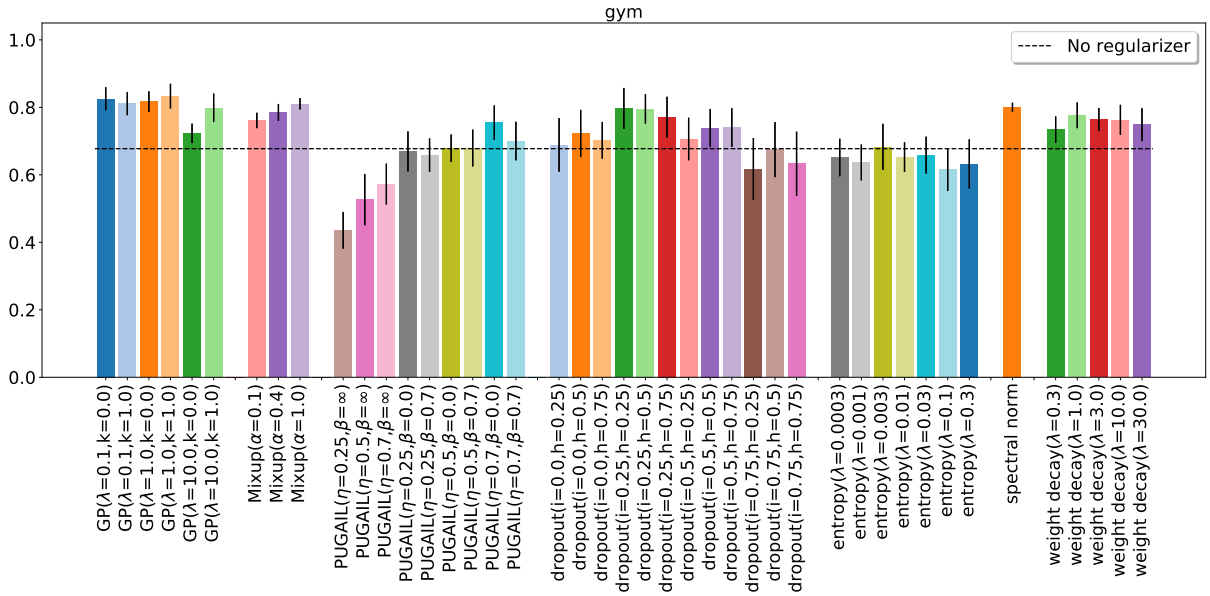


Figure A.63: 95th percentile of performance scores conditioned on `discriminator regularizer` (C45) and regularizers' HPs averaged across OpenAI Gym environments.

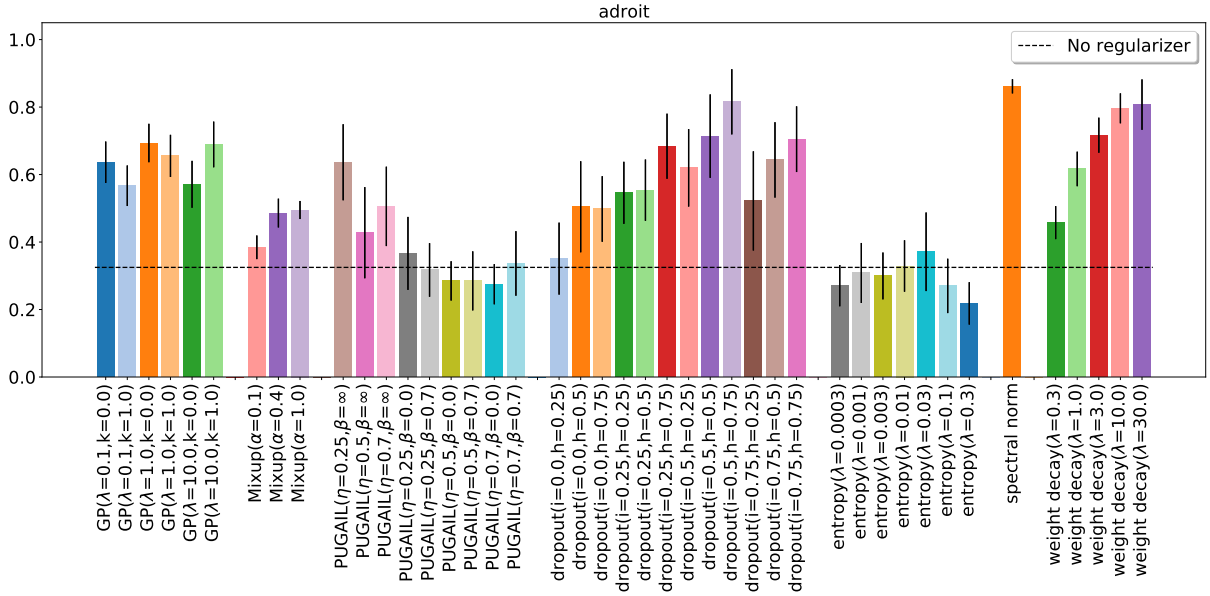


Figure A.64: 95th percentile of performance scores conditioned on `discriminator regularizer` (C45) and regularizers' HPs averaged across Adroit environments.

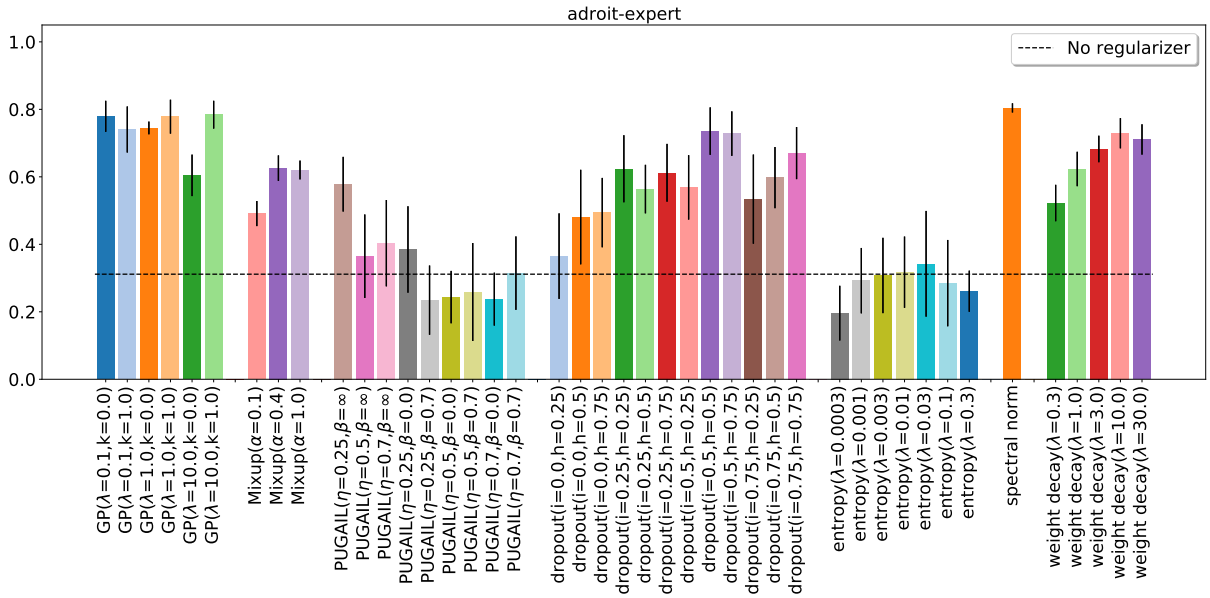


Figure A.65: 95th percentile of performance scores conditioned on `discriminator regularizer` (C45) and regularizers' HPs averaged across `door-expert` and `hammer-expert` tasks.

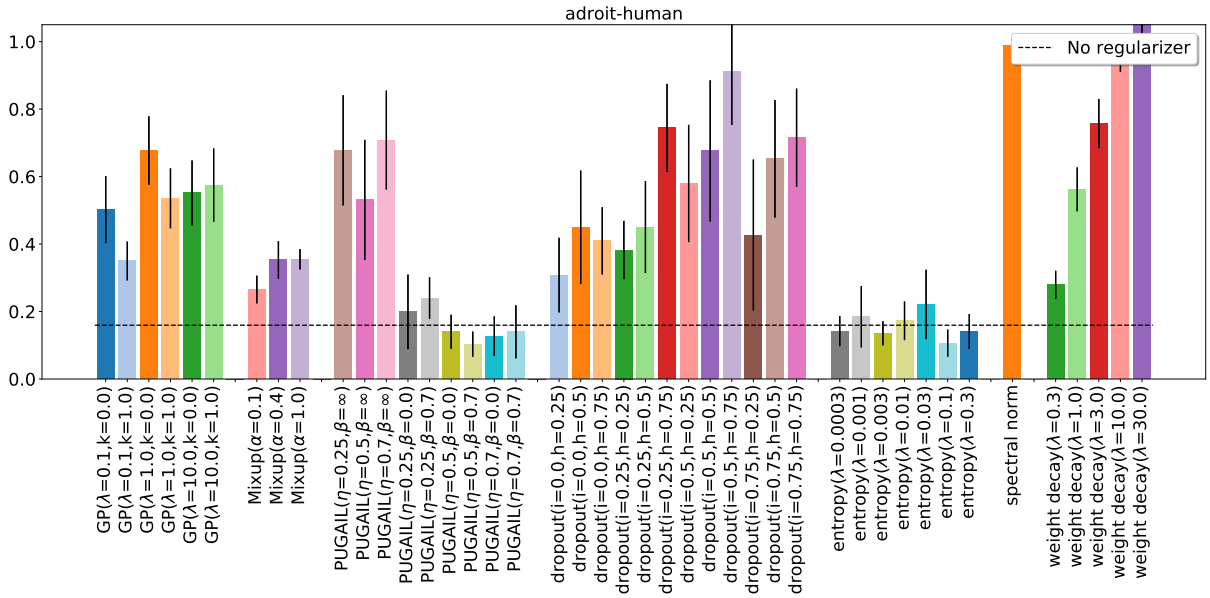


Figure A.66: 95th percentile of performance scores conditioned on `discriminator regularizer` (C45) and regularizers' HPs averaged across `door-human` and `hammer-human` tasks.

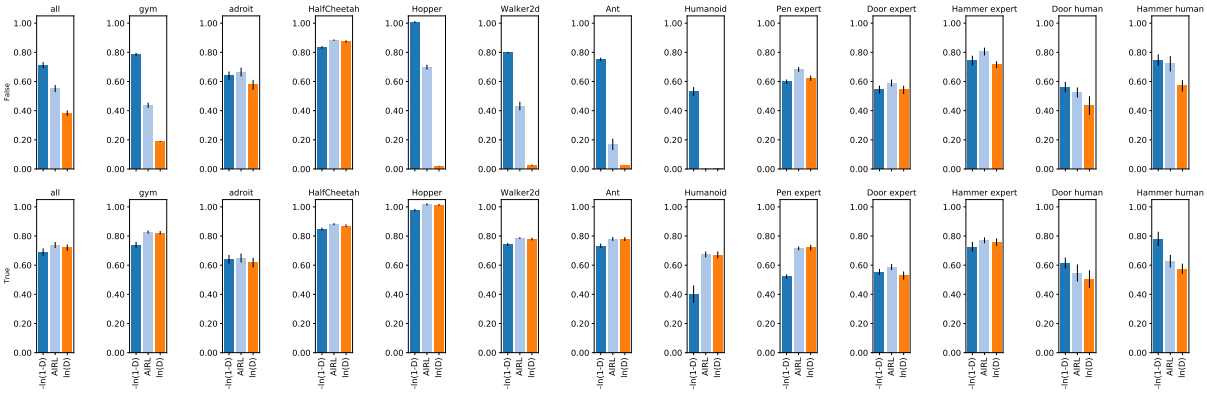


Figure A.67: 95th percentile of performance scores conditioned on `absorbing state (C32)`(rows) and `reward function (C30)`(bars).

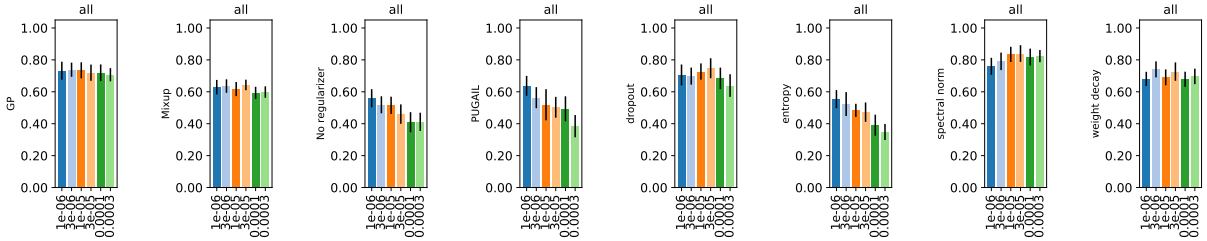


Figure A.68: 95th percentile of performance scores conditioned on `discriminator regularizer (C45)`(subplots) and `discriminator learning rate (C42)`(bars).

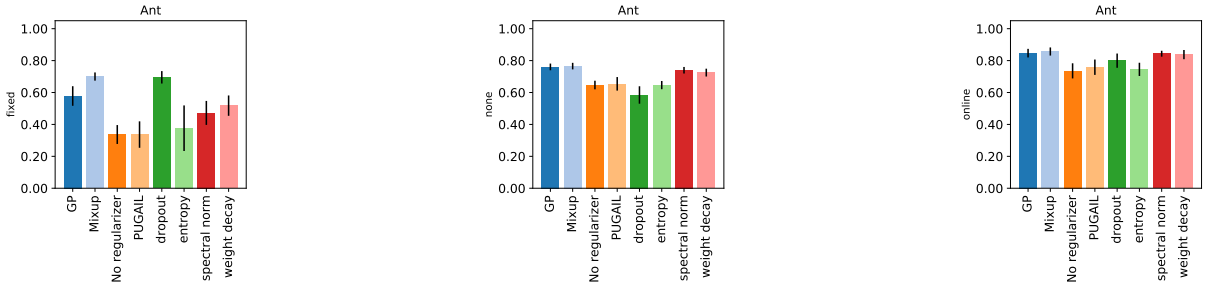


Figure A.69: 95th percentile of performance scores conditioned on `observation normalization` (C55)(subplots) and `discriminator regularizer` (C45)(bars) in the `Ant` environment.

A.6 Experiment trade-offs

A.6.1 Design

For each of the 10 tasks, we sampled 7991 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `RL Algorithm (C8)`: {d4pg, sac, td3}
  - For the case "`RL Algorithm (C8)` = sac", we further sampled the sub-choices:
    - `SAC learning rate (C17)`: {0.0001, 0.0003, 0.001}
    - `SAC entropy per dimension (C16)`: {-2.0, -1.0, -0.5, 0.0}
    - `SAC polyak` τ `(C18)`: {0.001, 0.003, 0.01, 0.03}
  - For the case "`RL Algorithm (C8)` = d4pg", we further sampled the sub-choices:
    - `D4PG learning rate (C26)`: {3e-05, 0.0001, 0.0003}
    - `behavioral policy noise (C21)`: {0.1, 0.2, 0.3, 0.5}
    - `VMax (C24)`: {150.0, 750.0, 1500.0}
    - `number of atoms (C23)`: {51.0, 101.0, 201.0, 401.0}
    - `N-step returns (C25)`: {1.0, 3.0, 5.0}
  - For the case "`RL Algorithm (C8)` = td3", we further sampled the sub-choices:
    - `TD3 policy learning rate (C19)`: {0.0001, 0.0003, 0.001}
    - `TD3 critic learning rate (C20)`: {0.0001, 0.0003, 0.001}
    - `TD3 gradient clipping (C22)`: {40.0, ∞}
    - `behavioral policy noise (C21)`: {0.1, 0.2, 0.3, 0.5}
- `RL replay buffer size (C28)`: {300000, 1000000, 3000000}
- `policy MLP depth (C1)`: {1, 2, 3}
- `policy MLP width (C2)`: {64, 128, 256, 512}
- `critic MLP depth (C3)`: {2, 3}
- `critic MLP width (C4)`: {256, 512}
- `RL activation (C5)`: {relu, tanh}
- `discount` γ `(C6)`: {0.97, 0.99}
- `BC pretraining (C34)`: {False, True}
- `absorbing state (C32)`: {False, True}
- `discriminator replay buffer size (C43)`: {300000, 1000000, 3000000}
- `reward shaping (C39)`: {False, True}
- `discriminator input (C35)`: {s, sa, sas, ss}

- discriminator MLP depth (C36): {1, 2, 3}
- discriminator MLP width (C37): {16, 32, 64, 128, 256, 512}
- discriminator activation (C38): {elu, leaky_relu, relu, sigmoid, swish, tanh}
- discriminator last layer init scale (C41): {0.001, 1.0}
- discriminator regularizer (C45): {GP, Mixup, No regularizer, PUGAIL, dropout, entropy, spectral norm, weight decay}
  - For the case "discriminator regularizer (C45) = GP", we further sampled the sub-choices:
    - gradient penalty λ (C47): {0.1, 1.0, 10.0}
    - gradient penalty k (C46): {0.0, 1.0}
  - For the case "discriminator regularizer (C45) = Mixup", we further sampled the sub-choices:
    - mixup α (C48): {0.1, 0.4, 1.0}
  - For the case "discriminator regularizer (C45) = PUGAIL", we further sampled the sub-choices:
    - PUGAIL η (C49): {0.25, 0.5, 0.7}
    - PUGAIL β (C50): {0.0, 0.7, ∞}
  - For the case "discriminator regularizer (C45) = entropy", we further sampled the sub-choices:
    - entropy λ (C54): {0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3}
  - For the case "discriminator regularizer (C45) = weight decay", we further sampled the sub-choices:
    - weight decay λ (C53): {0.3, 1.0, 3.0, 10.0, 30.0}
  - For the case "discriminator regularizer (C45) = dropout", we further sampled the sub-choices:
    - dropout input rate (C52): {0.0, 0.25, 0.5, 0.75}
    - dropout hidden rate (C51): {0.25, 0.5, 0.75}
- observation normalization (C55): {fixed, none}
- evaluation behavior policy type (C29): {average, mode, stochastic}
- discriminator learning rate (C42): {1e-06, 3e-06, 1e-05, 3e-05, 0.0001, 0.0003}
- replay ratio (C27): {64, 128, 256, 512, 1024}
- batch size (C7): {64, 128, 256, 512, 1024}
- discriminator to RL updates ratio (C44): {1, 2}
- number of combined batches (C56): {1, 2, 4, 8, 16, 32, 64}
- reward function (C30): {-ln(1-D), AIRL, ln(D)}

A.6.2 Results

For each of the sampled choice configurations we compute the performance metric as described in Section 3.2. We report aggregate statistics of the experiment in Tables A.11–A.14 as well as training curves in Figure A.70. We further provide per-choice analyses in Figures A.71-A.74.

Table A.11: Quantiles of the *final* agent performance across HP configurations for OpenAI Gym tasks.

	Ant	HalfCheetah	Hopper	Humanoid	Walker2d
90%	0.81	1.04	1.18	0.14	0.97
95%	0.94	1.08	1.19	0.62	1.00
99%	1.04	1.15	1.22	0.98	1.03
Max	1.15	1.41	1.31	1.05	1.16

Table A.12: Quantiles of the *final* agent performance across HP configurations for Adroit tasks.

	Door expert	Door human	Hammer expert	Hammer human	Pen expert
90%	0.71	0.25	1.03	0.45	0.70
95%	0.89	0.71	1.25	1.19	0.86
99%	1.04	2.12	1.36	2.95	1.07
Max	1.15	3.79	1.44	5.27	1.34

Table A.13: Quantiles of the *average* agent performance during training across HP configurations for OpenAI Gym tasks.

	Ant	HalfCheetah	Hopper	Humanoid	Walker2d
90%	0.48	0.75	0.90	0.12	0.63
95%	0.63	0.83	0.98	0.32	0.72
99%	0.77	0.92	1.06	0.62	0.83
Max	0.89	1.00	1.10	0.85	0.92

Table A.14: Quantiles of the *average* agent performance during training across HP configurations for Adroit tasks.

	Door expert	Door human	Hammer expert	Hammer human	Pen expert
90%	0.38	0.26	0.54	0.39	0.50
95%	0.53	0.49	0.71	0.65	0.63
99%	0.74	1.02	0.91	1.21	0.82
Max	0.94	2.05	1.17	2.13	1.01

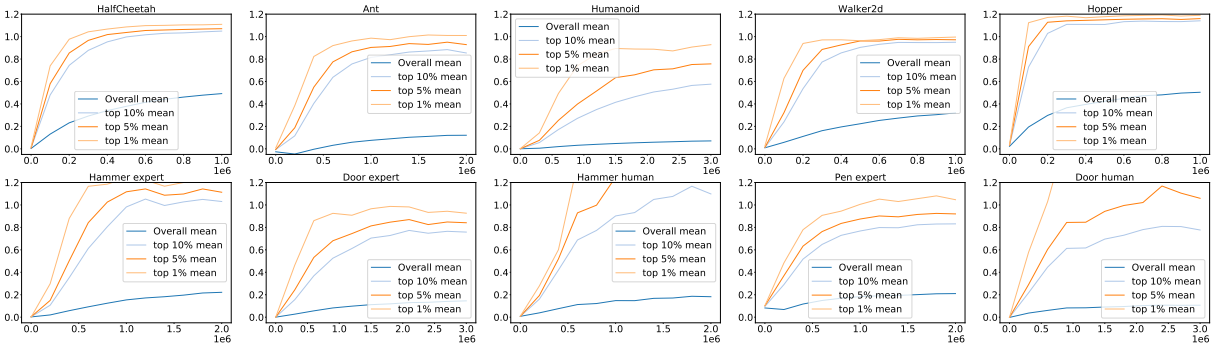


Figure A.70: Training curves.

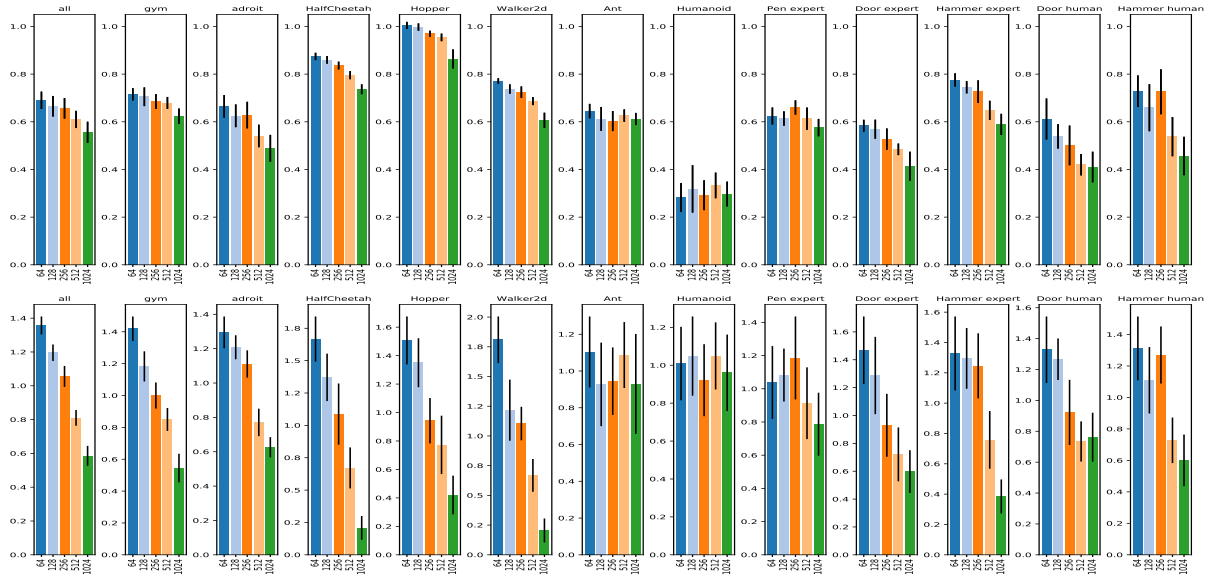


Figure A.71: Analysis of choice `batch size` (C7): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

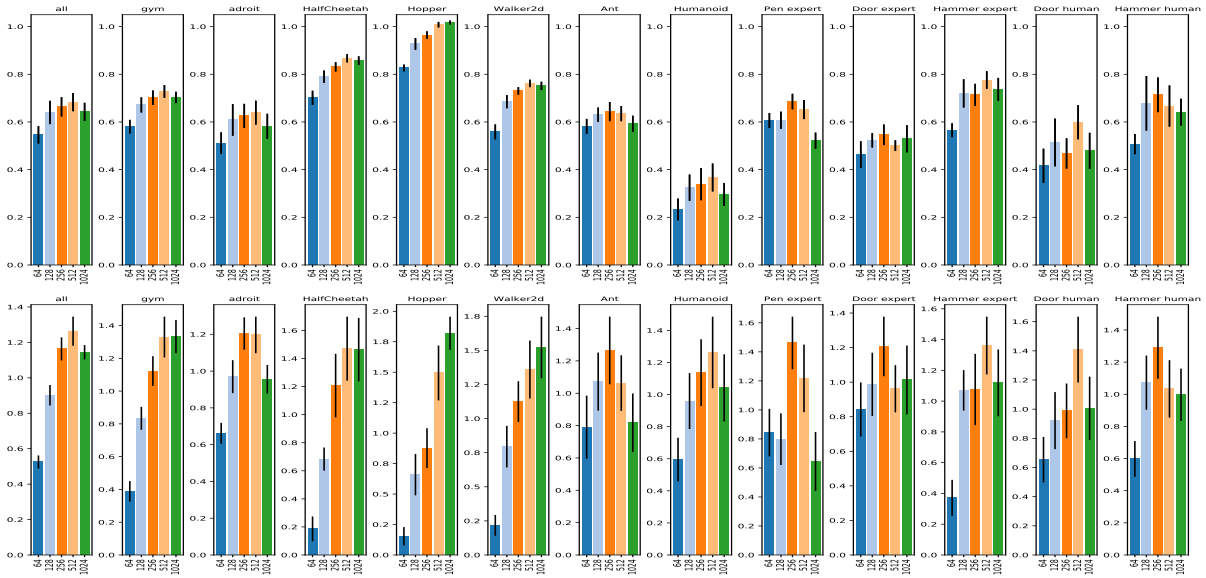


Figure A.72: Analysis of choice `replay ratio (C27)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

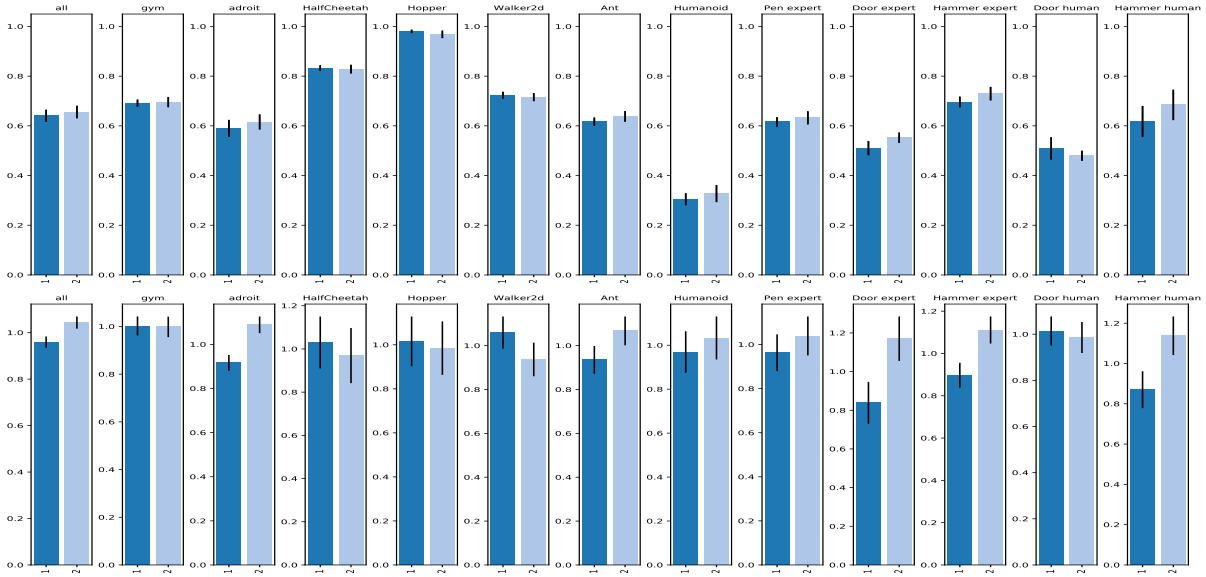


Figure A.73: Analysis of choice `discriminator to RL updates ratio (C44)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

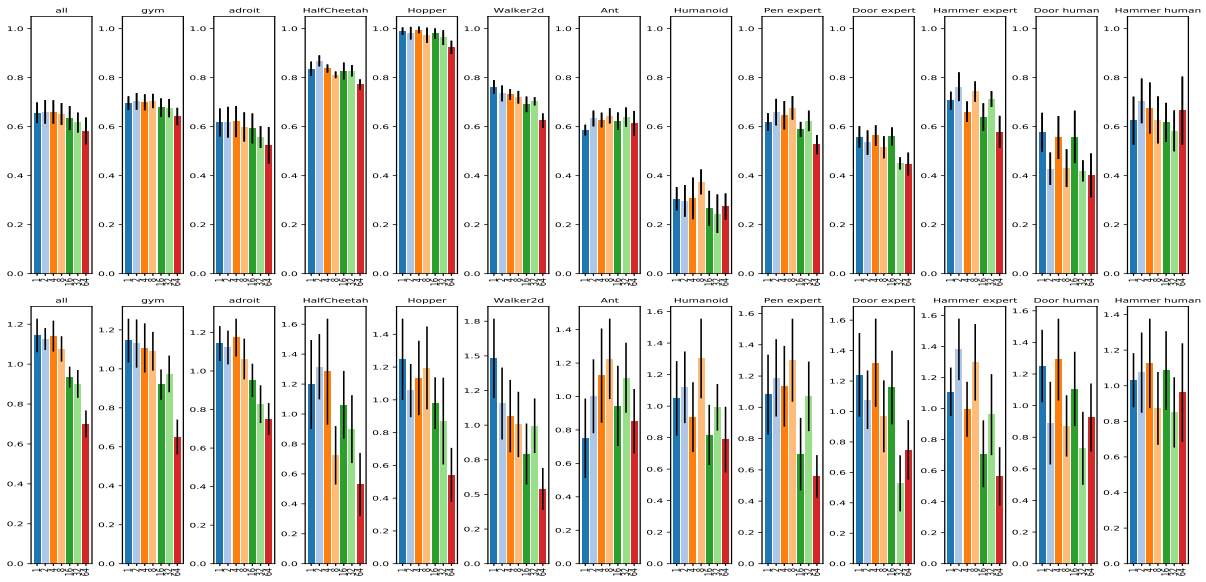


Figure A.74: Analysis of choice `number of combined batches` (C56): 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom).

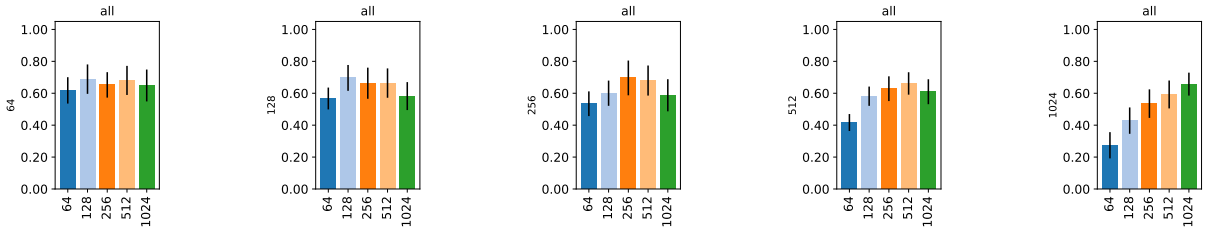


Figure A.75: 95th percentile of performance scores conditioned on `batch size (C7)`(subplots) and `replay ratio (C27)`(bars).

A.7 Additional experiments

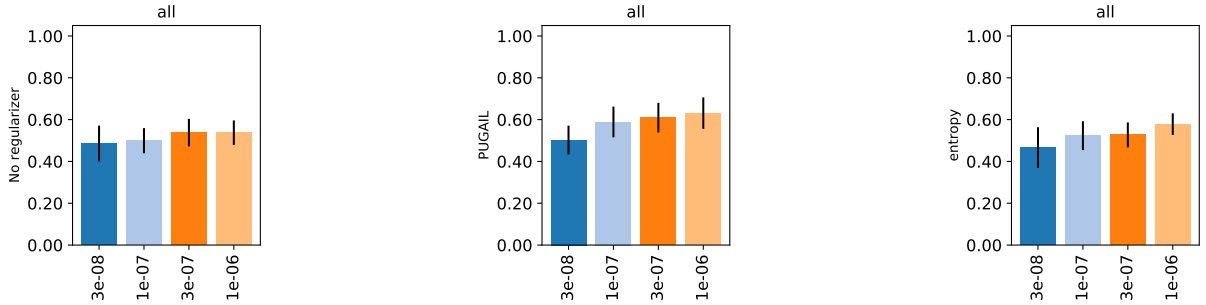


Figure A.76: 95th percentile of performance scores conditioned on `discriminator regularizer (C45)`(rows) and `discriminator learning rate (C42)`(bars). The data comes from an experiment similar to the main one but with smaller values of `discriminator learning rate (C42)`.

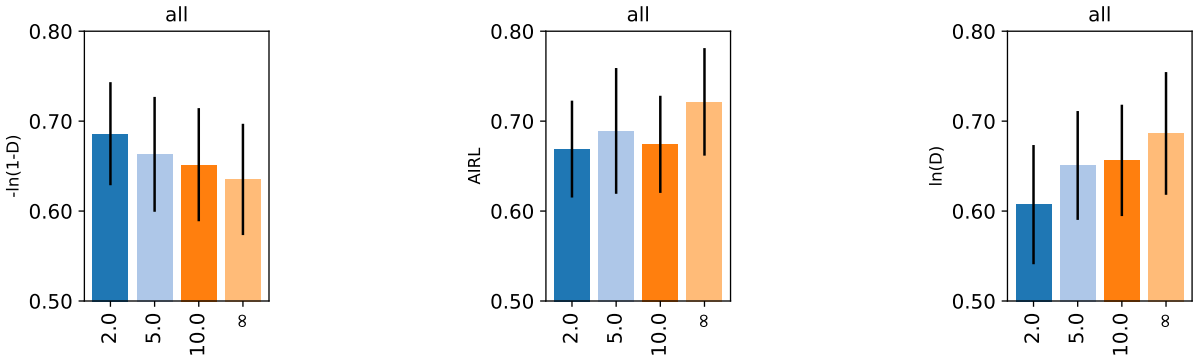


Figure A.77: 95th percentile of performance scores conditioned on `reward function (C30)`(subplots) and `max reward magnitude (C31)`(bars). The data comes from an experiment similar to the main one but with `max reward magnitude (C31)` swept.

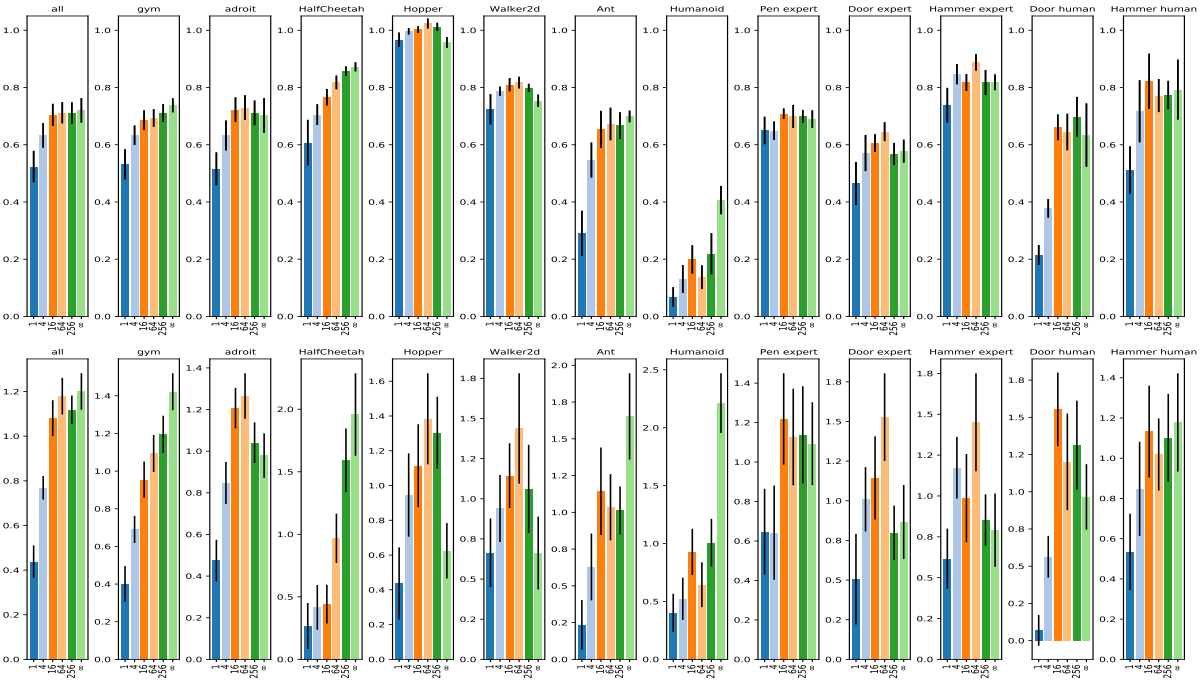


Figure A.78: Analysis of choice `policy-to-expert replay ratio (C33)`: 95th percentile of performance scores conditioned on choice (top) and distribution of choices in top 5% of configurations (bottom). The data comes from an experiment similar to the main one but in which we also sweep `policy-to-expert replay ratio (C33)`. All other experiments do not replay expert data.

Appendix B

Primal Wasserstein Imitation Learning: Details and additional figures.

B.1 Locomotion Experiments

B.1.1 PWIL Implementation

The agent we use is D4PG [Barth-Maron et al., 2018]. We use the default architecture from the ACME framework [Hoffman et al., 2020].

The actor architecture is a 4-layer neural network: the first layer has size 256 with tanh activation and layer normalization [Ba et al., 2016], the second layer and third layer have size 256 with elu activation [Clevert et al., 2016], the last layer is of size the dimension of the action space, with a tanh activation scaled to the action range of the environment. To enable sufficient exploration with use a Gaussian noise layer on top of the last layer with standard deviation $\sigma = 0.2$, that we clip to the action range of the environment. We evaluate the agent without exploration noise.

For the critic network we use a 4-layer neural network: the first layer has size 512 with tanh activation and layer normalization, the second layer is of size 512 with elu activation, the third layer is of size 256 with elu activation, the last layer is of dimension 201 with a softmax activation. The 201 neurons stand for the weights of the distribution supported within equal distance in the range $[-150, 150]$, as a categorical distribution [Bellemare et al., 2017].

We use the Adam optimizer [Kingma and Ba, 2015] with $\lambda_a = 5 \times 10^{-5}$ for the actor and $\lambda_c = 7 \times 10^{-5}$ for the critic. We use a batch size of 256. We clip both gradients from the critic and the actor to limit their L2 norm to 40.

We use a replay buffer of size 10^6, a discount factor $\theta = 0.99$ and n step returns with $n = 5$. We prefill the replay buffer with 50000 state-action pairs from the set of demonstrations (which means that we put multiple times the same expert transitions in the buffer). We perform updates on the actor and the critic every $k = 4$ interactions with the environment. The hyperparameters search is detailed in Table B.1.

For the reward function, we did run a hyperparameter search on α and β with the following values for α and β in $\{1, 5, 10\}$.

Parameters	Values
λ_a	$10^{-5}, 5 \times 10^{-5}, 7 \times 10^{-5}, 10^{-4}$
λ_c	$10^{-5}, 5 \times 10^{-5}, 7 \times 10^{-5}, 10^{-4}$
σ	$0.1, 0.2, 0.3$
k	$2, 4, 8, 16$

Table B.1: DDPG hyperparameters search.

B.1.2 DAC Implementation

We used the open-source implementation of DAC provided by Kostrikov et al. [2019]. For Humanoid (which was not reported in the paper), we did run a large hyperparameter search described in Table B.2. Out of the 729 experiments, 7 led to an average performance above 1000 at the 2.5M training environment steps mark.

Actor learning rate (lr)	Critic lr	Discriminator lr	WGAN regularizer	Exploration noise	Decay lr
0.001, 0.0005, 0.0001	0.001, **0.0005**, 0.0001	**0.001**, 0.0005, 0.0001	5, **10**, 20	**0.1**, 0.2, 0.3	0.3, 0.5, **0.8**

Table B.2: Hyperparameter search on Humanoid for 11 demonstrations. (tested/**best**)

B.1.3 BC Implementation

We used a 3-layer neural network, the first layer has size 128 with relu activation, the second layer has size 64 and the last layer has size of the action space with a tanh activation scaled to the action range of the environment. We found out that normalizing the observations with the average and standard deviation of the expert demonstrations' observations helped performance. We trained the network using the mean squared error as a loss, with an Adam optimizer. We ran an hyperpameter search on the learning rate in $\{10^{-5}, 10^{-4}, 10^{-3}\}$ and on the batch size $\{128, 256\}$.

B.1.4 PWIL Learning Curves

We present the learning curves of PWIL, in terms of the reward defined in Equation 4.6 as well as the original reward of the task.

B.2 Ablation study

We present the learning curves of PWIL in the presence of ablations.

B.3 Influence of the Direct RL Algorithm

In this section, we study the influence of the direct RL method on the performance on PWIL on locomotion tasks. We compare the performance of PWIL with D4PG with the performance of PWIL with TD3 and SAC. We use a custom implementation of both SAC and TD3 which is a reproduction of the authors' implementation. For SAC we used the same default hyperparameters; for TD3 we removed the delayed

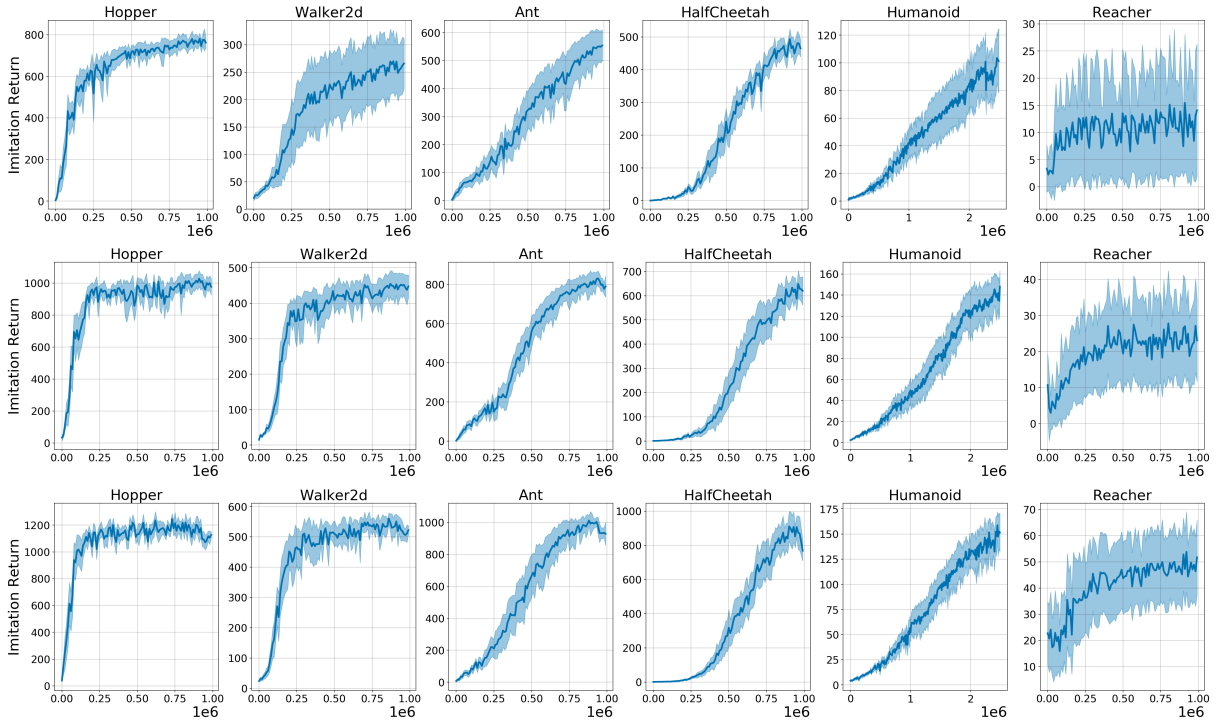


Figure B.1: Mean and standard deviation of the imitation return of the agent. We perform 10 rollouts over 10 seeds every 10k environment steps over 1M steps. The return here is in term of the reward defined with Equation (4.6). Top row: 1 demonstration, middle row: 4 demonstrations, bottom row 11 demonstrations.

policy update which led to better performance. We evaluate both methods on the same reward function as D4PG. We remove the pre-warming of the replay buffer for SAC. For TD3, we found out that we need different pre-warming for each environment to get good performance (contrary to D4PG): we used 1000 state-action pairs for Reacher, Hopper and Walker2d and no pre-warming for Humanoid, Ant and HalfCheetah.

We show in Figure B.5 that both PWIL-SAC and PWIL-TD3 recover expert-like performance on Hopper, Ant and HalfCheetah and Reacher. For Walker2d, there is a large standard deviation in performance, since some seeds lead to near-optimal expert performance, and some seeds lead to poor performance. Remark that we led minimal hyperparameter tuning on the default versions of SAC and TD3, which might explain the differences in performance.

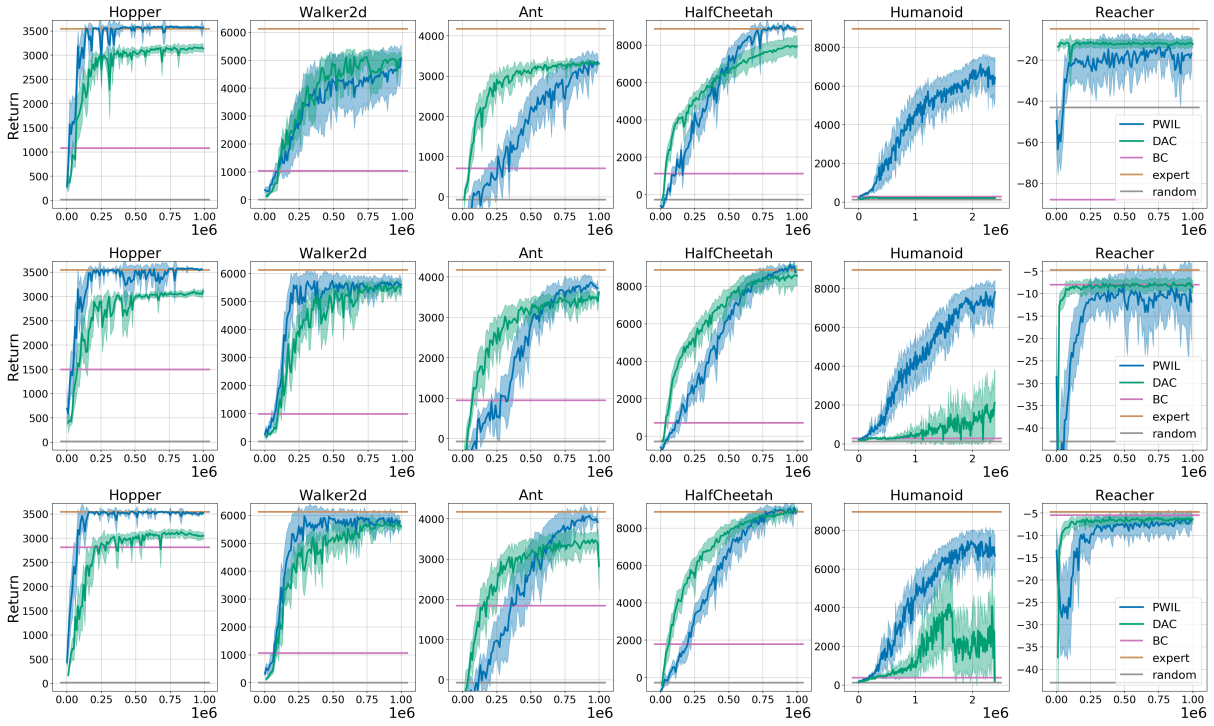


Figure B.2: Mean and standard deviation of the original environment returns of the evaluation policy over 10 rollouts and 10 seeds, reported every 10k environment steps over 1M steps. Top row: 1 demonstation, middle row: 4 demonstrations, bottom row: 11 demonstrations.

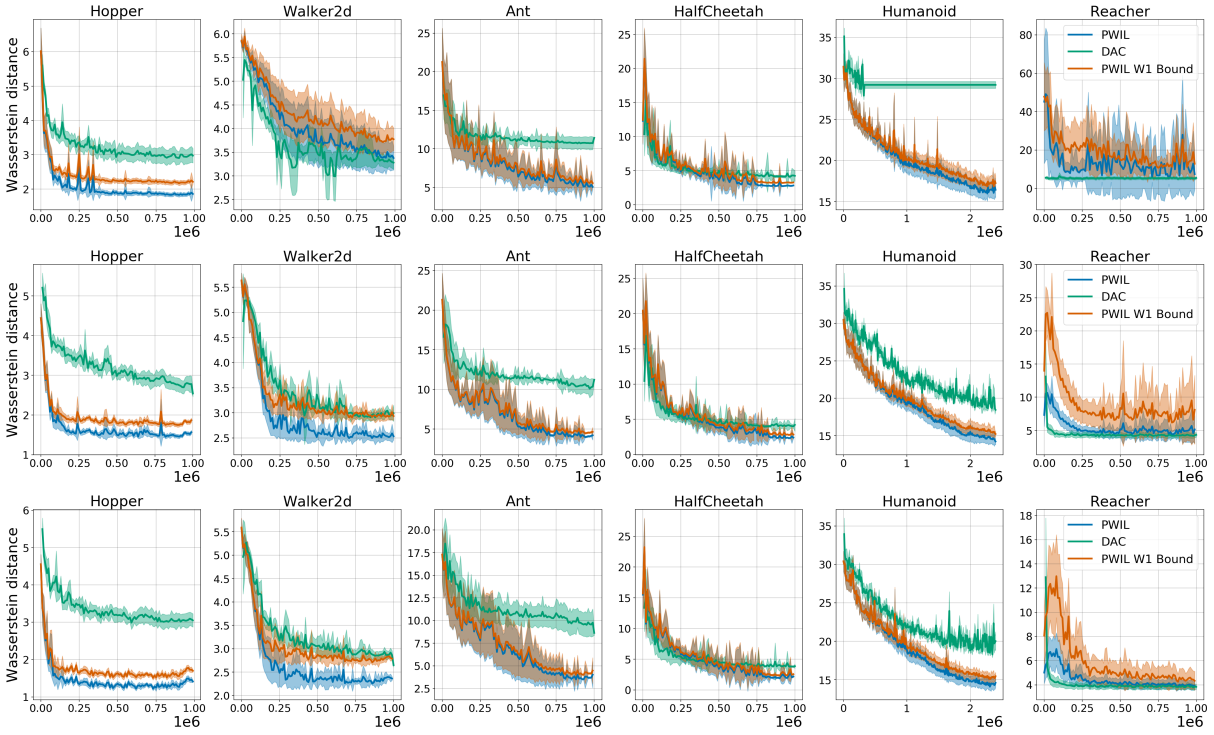


Figure B.3: Mean and standard deviation of the Wasserstein distance between the state-action distribution of the evaluation policy and the state-action distribution of the expert over 10 rollouts and 10 seeds, reported every 10k environment steps. We include the upper bound on the Wasserstein distance based on the greedy coupling defined in Equation (4.4). Top row: 1 demos, middle row: 4 demos, bottom row: 11 demos.

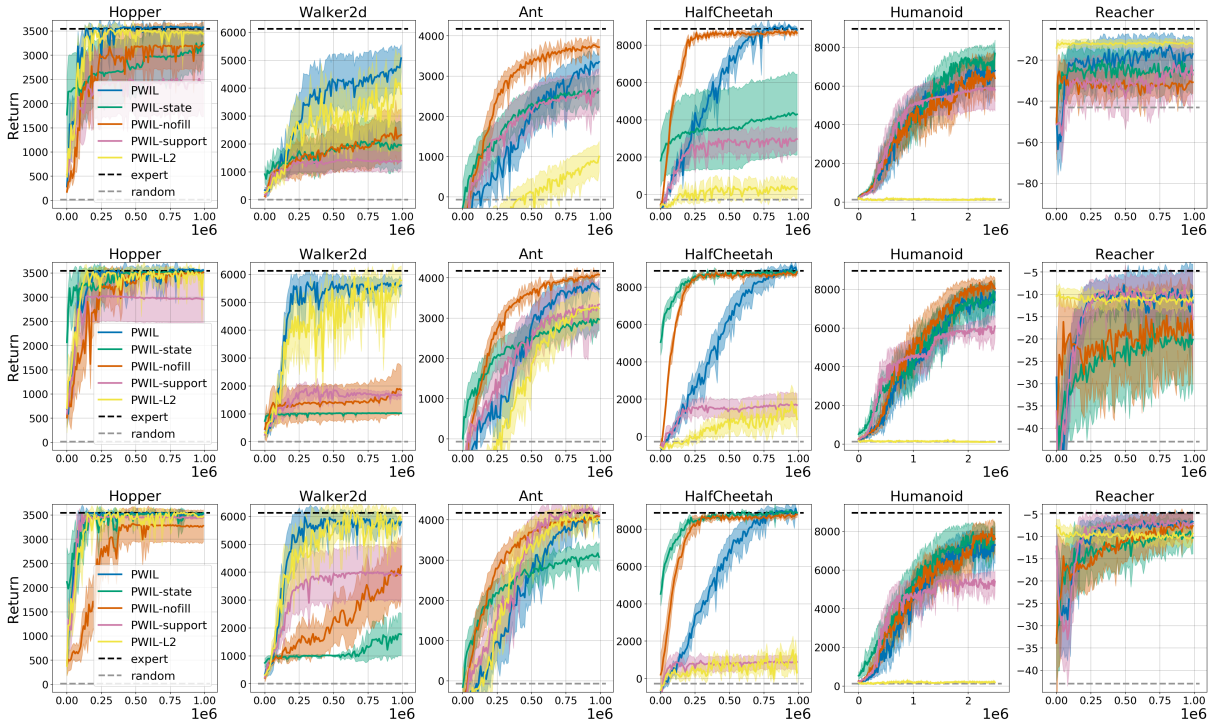


Figure B.4: Evaluation performance of different variants of PWIL over 10 rollouts and 10 seeds, reported every 10k environment steps over 1M steps. The return is in term of the environment's original reward. Top row: 1 demonstration, middle row: 4 demonstrations, bottom row: 11 demonstrations.

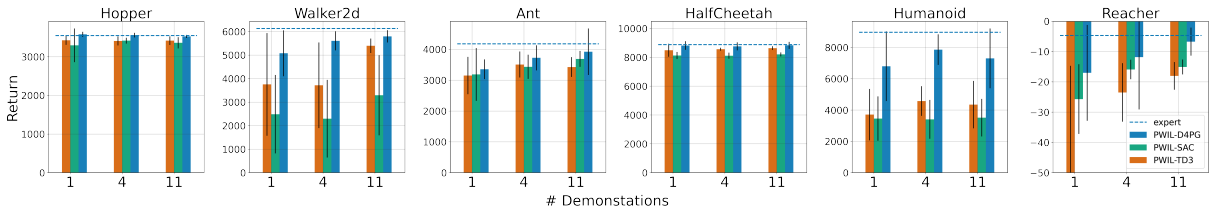


Figure B.5: Mean and standard deviation of the evaluation performance of PWIL-D4PG, PWIL-SAC and PWIL-TD3 at the 1M environment interactions mark (2.5M for Humanoid). Results are computed over 10 seeds and 10 episodes for each seed.

B.4 Door Opening Experiments

In this section we present the details of the experiment presented in Section 4.4.4.

B.4.1 Environment

The default door opening environment has a fixed horizon of 200 timesteps. We add an early termination condition to the environment when the door is opened to demonstrate that PWIL extends to tasks where there is no incentive for survival. Note that similarly to AIRL [Fu et al., 2017], early termination states are considered to be absorbing. Therefore if the expert reaches an early termination state early, its state distribution as a large weight on these states, which leads the PWIL agent to similarly reach the early termination state. 25 demonstrations were collected from humans using a virtual reality system (Kumar and Todorov [2015]).

B.4.2 Embedding

As we assume that we only have access to the visual rendering of the demonstrations (with resolution 84×84), we build a lower dimensional latent space. We use a self-supervised learning method: TCC, to learn the latent space. We used the classification version of the algorithm detailed in Dwibedi et al. [2019] (Section 3.2).

The encoding network is the following: a convolutional layer with 128 filters of size 8 and stride 4, a relu activation, a convolutional layer with 64 filters of size 4 and stride 2, a relu activation, a convolutional layer with 64 filters of size 3 and stride 1, a relu activation, a linear layer of size 128, a relu activation and an output layer of size 32.

We separated the demonstrations into a train set of 20 demonstrations and a validation set of 5 demonstrations. Similarly to Dwibedi et al. [2019], we select the encoding network by taking the one that maximizes the alignment according to Kendall's tau score on the validation set.

We ran an hyperparameter search over the batch size $\{1, 4, 16, 32\}$, the subsampling ratio $\{1, 4, 8\}$ and the learning rate $\{10^{-2}, 10^{-3}, 10^{-4}\}$ (using the Adam optimizer). The best performing encoder had an average Kendall's tau score of 0.98 on the training set and 0.91 on the validation set.

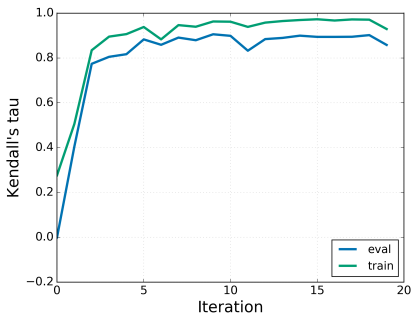


Figure B.6: Kendall's tau score of the encoding network on the train and validation sets. An iteration corresponds to 1000 gradient updates.

B.4.3 Agent

In this setup, we found that SAC leads to better results than D4PG. We detail the implementation of SAC below.

The actor is a 3-layer network that outputs the parameters of a normal distribution with diagonal covariance matrix. We enforce the terms of the covariance matrix to be greater than 10^{-3}. The first layer has size 256 with relu activation, the second layer has size 256 with relu activation and the last layer has size 48 (the action space has size 24).

The critic is a 3-layer network. We pass the concatenation of the state and the action of the environment as an input. There are two hidden layers of size 256 with relu activation. We do use a twin critic.

We use an adaptive temperature α as described by Haarnoja et al. [2018c].

For the three losses we use the Adam optimizer, and we use Polyak averaging for the critic networks with a rate τ. We ran an hyperparameter search over the learning rates $\lambda_a, \lambda_c, \lambda_\alpha$ in $\{10^{-4}, 3 \times 10^{-4}\}$, over τ $\{0.01, 0.005\}$, over the batch size $\{64, 128, 256\}$ and over the number of interactions with the environment between each actor, critic and temperature update $\{1, 4, 8\}$.

B.5 Algorithm Details

B.5.1 Rundown

We present an example rundown of PWIL in Figure B.7.

B.5.2 Runtime

We detail the runtime of PWIL in this section. A single step reward computation consists in computing the distance from the current observation (a vector with dimension $|\mathcal{S}| + |\mathcal{A}|$ with D vectors (the expert observations), which has complexity $\mathcal{O}((|\mathcal{S}| + |\mathcal{A}|)D)$. The next step is to get the minimum of these distances (complexity $\mathcal{O}(D)$), for $\lceil \frac{D}{T} \rceil$ times (the number of times the algorithm executes the while loop). Therefore the complexity of a single step reward computation is $\mathcal{O}((|\mathcal{S}| + |\mathcal{A}|)D + \frac{D^2}{T})$ and the complexity of the reward computation of an episode is $\mathcal{O}((|\mathcal{S}| + |\mathcal{A}|)DT + D^2)$.

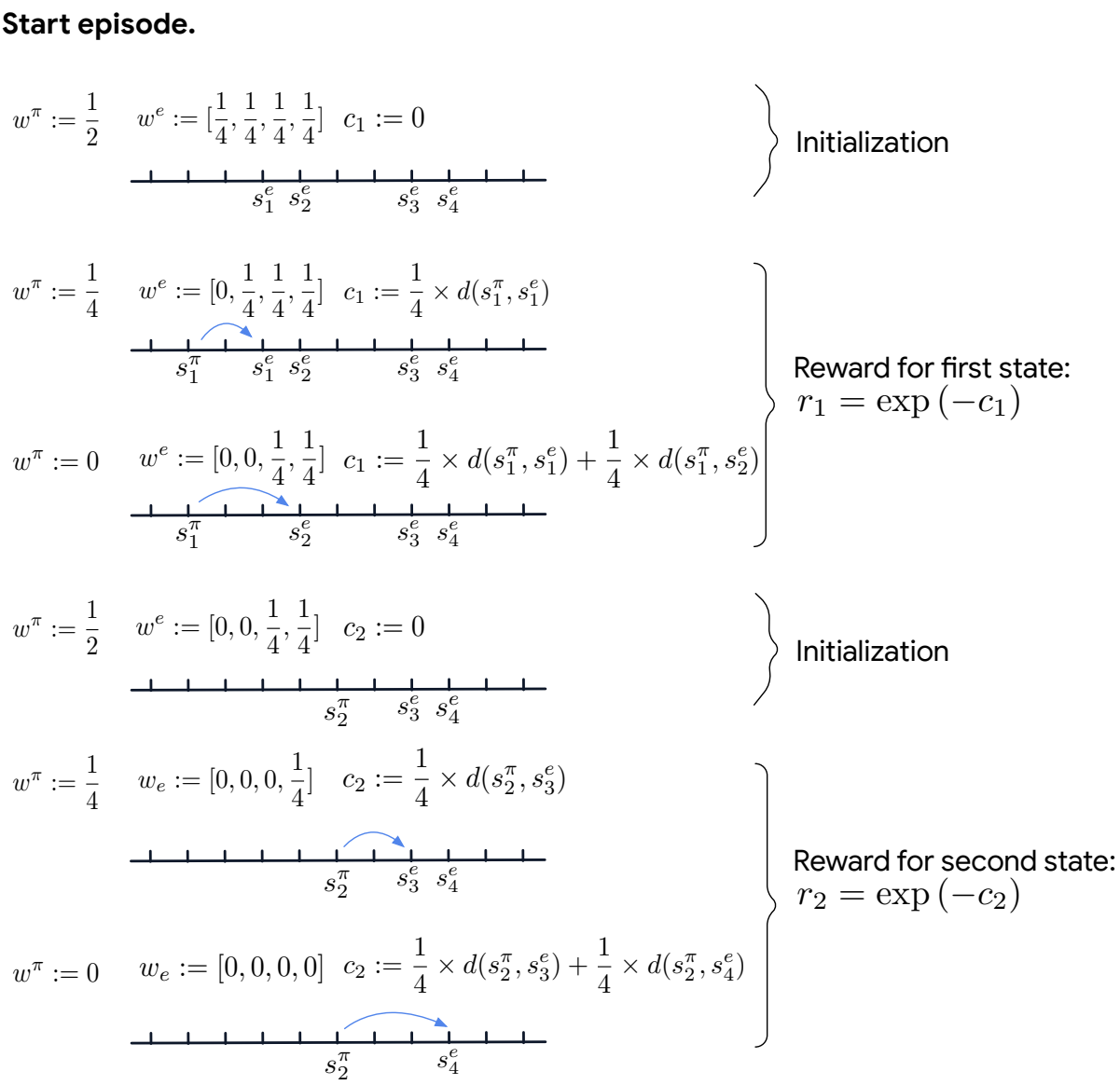


Figure B.7: PWIL example rundown. We drop the dependency on the action. The example task has an horizon $T = 2$, and the demonstration set has $D = 4$ elements. We take $f : c \mapsto \exp(-c)$.

Appendix C

Hyperparameter Selection for Imitation Learning: Details and additional figures.

C.1 Experimental details

All experiments use MLP networks (optionally with dropout [Srivastava et al., 2014]) and Adam [Kingma and Ba, 2015] optimizer. Some experiments use observation normalization which means that the observation coordinates are rescaled to have the mean equal 0 and standard deviation equal 1. The normalization statistics are computed using the dataset of expert trajectories.

Both, AIL and PWIL use SAC [Haarnoja et al., 2018a] with an entropy constraint [Haarnoja et al., 2018c] to train the policy and optionally use absorbing states introduced in DAC [Kostrikov et al., 2019]. When evaluating SAC policies, we decrease the stochasticity of the policy by sampling each actions 5 times and averaging the sampled actions.

C.1.1 Behavioral Cloning

The HP sweep for BC can be found in Table C.1.

C.1.2 Adversarial Imitation Learning

The HP sweep for AIL can be found in Table C.2. The AIL experiment samples the policy reward function from the following options [Kostrikov et al., 2019, Ghasemipour et al., 2020]:

- $\log D(s, a)$,
- $-\log(1 - D(s, a))$,
- $\log D(s, a) - \log(1 - D(s, a))$.

To prevent the discriminator saturation, we subtract the entropy of the discriminator output (treated as a Bernoulli distribution) from the discriminator loss.

Table C.1: Hyperparameter sweep for BC.

Hyperparameter	Possible values
gradient updates	6000
batch size	256
learning rate	10^{-5}, 10^{-4}
weight decay coefficient	0, 0.01, 0.1
observation normalization	True, False
policy output	deterministic
# of policy hidden layers	1, 2, 3
hidden layer size	16-256
activation function	ReLU, tanh
input dropout rate	0, 0.15, 0.3
hidden dropout rate	0, 0.25, 0.5

Table C.2: Hyperparameter sweep for AIL.

Hyperparameter	Possible values
discriminator training	
# of gradient steps per env step	1
training batch size	256
# of hidden layers	2
hidden layer size	256
activation function	ReLU
learning rate	10^{-6}–$3 \cdot 10^{-4}$
entropy coefficient	10^{-4}–10^{-2}
RL reward function	See App. C.1.2
observation normalization	True, False
absorbing states	True, False
RL training	See Table C.3.

C.1.3 Primal Wasserstein Imitation Learning

The hyperparameter sweep for PWIL can bew found in Table C.4. PWIL prefills the SAC replay buffer with a number of transitions from the expert dataset[1].

C.1.4 Random Network Distillation metric

This metric uses a first frozen feed-forward network f with layer sizes $(128, 128, 128, 128)$ as a target and a second learnable network $\hat{f}$ with layer sizes $(128, 128)$ to predict the output of the former. The latter is trained with Adam and a learning rate of 0.001 to minimize the mean-squared error for 100 epochs on the demonstration dataset. The corresponding metric is given by $\exp(-||f(obs) - \hat{f}(obs)||)$.

[1]The reward for these transitions is set to α where α is the coefficient used in the PWIL reward definition [Dadashi et al., 2021a]. If the number of requested transitions is higher than the size of the expert dataset, we include multiple copies of each transition.

Table C.3: Hyperparameter sweep for SAC.

Hyperparameter	Possible values
environment steps	1M
replay buffer size	1M
gradient steps per env step	1
learning rate	$3 \cdot 10^{-5}$ – $1 \cdot 10^{-3}$
batch size	128, 256, 512
discount factor	0.9, 0.97, 0.99
target network coefficient	0.001 – 0.03
reward scale	0.01 – 1
target entropy	-0.5 per dimension
policy	
action distribution	Gaussian + tanh
# of hidden layers	2
hidden layer size	256
activation function	ReLU

Table C.4: Hyperparameter sweep for PWIL.

Hyperparameter	Possible values
observation normalization	True, False
absorbing states	True, False
reward coefficient α	0.5, 1, 5, 10
reward coefficient β	0.5, 1, 5, 10
# of expert transitions in the buffer	0, 5k, 50k
RL training	See Table C.3.

C.2 Additional results

C.2.1 Ranking property of the proposed metrics

Whether a metric is good for HP selection mostly depends on its capacity to preserve the ranking between policies induced by the oracle return. Assuming a set of policies whose expected return is evenly distributed over an interval, we can introduce two tasks that can serve as an intrinsic evaluation of the proxy metrics. The first task aims at ranking all the policies according to the chosen metric. The performance of this ranking is given by the Spearman correlation with the ranking provided by the oracle return. The second task aims at differentiating the set of good policies, i.e. policies whose oracle return is above a threshold (chosen to be 75% of the expert performance), from the set of poor policies: we compute the probability that the chosen metric is lower for a policy sampled randomly from the set of good policies than for a policy sampled randomly from the set of poor policies. This probability also corresponds to the ROC-AUC of a classifier that would be based on the same metric.

To create a set of policies, we train some agents using one of the IL algorithms configured as described in Sec. C.1 and collect for each HP configuration the policies from different stages of training. The resulting set of policies is highly imbalanced in terms of environment returns. To get policies evenly

distributed in terms of environment returns, we finally rebalance the set of policies. The two tasks are repeated for each set of policies obtained using BC, AIL and PWIL and the corresponding scores are averaged.

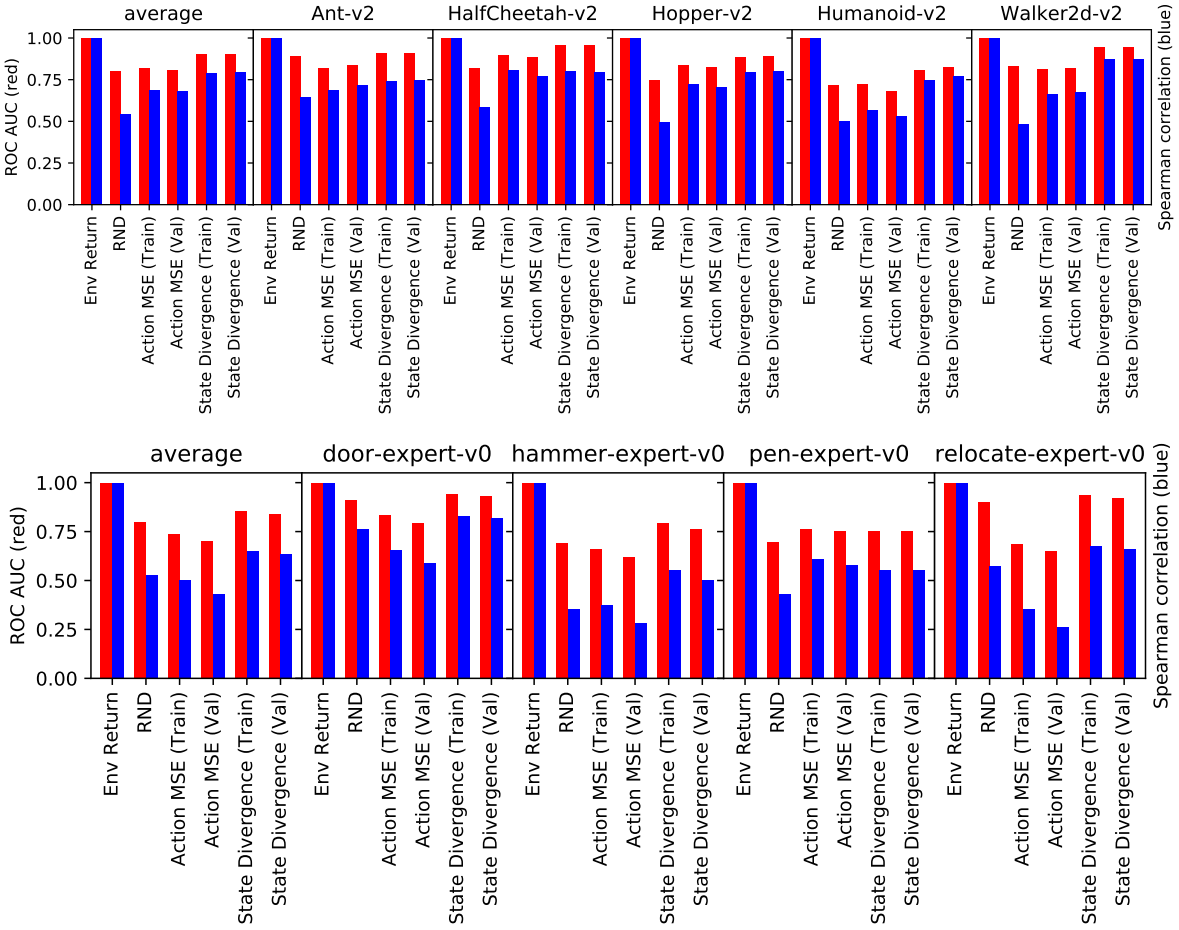


Figure C.1: ROC-AUC (red) and Spearman correlation (blue) of the proposed metrics with respect to the oracle return on a set of Mujoco and D4RL adroit environments.

Fig. C.1 clearly shows the metrics somewhat preserve the ranking induced by the oracle return. The best metric on those two ranking tasks is the state divergence, suggesting this could be the most suitable metric for HP selection.

C.2.2 Selecting the algorithm

The first row of Fig. 5.2 gives the episode returns for the different proposed metrics when the imitation learning algorithm is itself considered as an hyperparameter. The results suggest the action MSE metric favors BC even when BC is actually not the best algorithm. We provide in Fig. C.2 an evidence of this hypothesis through a detailed view of all the partially-trained models from different HP configurations for each of the three imitation learning algorithms. The models with best action MSE correspond to models

learned with BC although they are clearly not the best models in this case.

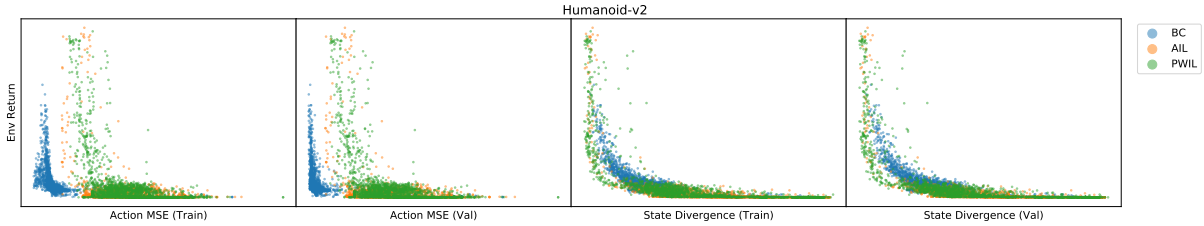


Figure C.2: Values of the environment return and of the action MSE and state divergence metrics for all the models (across HPs and training steps) of the three different imitation learning algorithms.

C.2.3 Transfer

We include in Fig.5.6 the performance of algorithms when transferring HPs from one environment to another when the early stopping is performed on the state divergence as this metric was the most promising according to Sec. 5.3.1. We include here the results of the same experiments when early stopping is performed on the oracle return (C.3) or when no early stopping is performed (C.4).

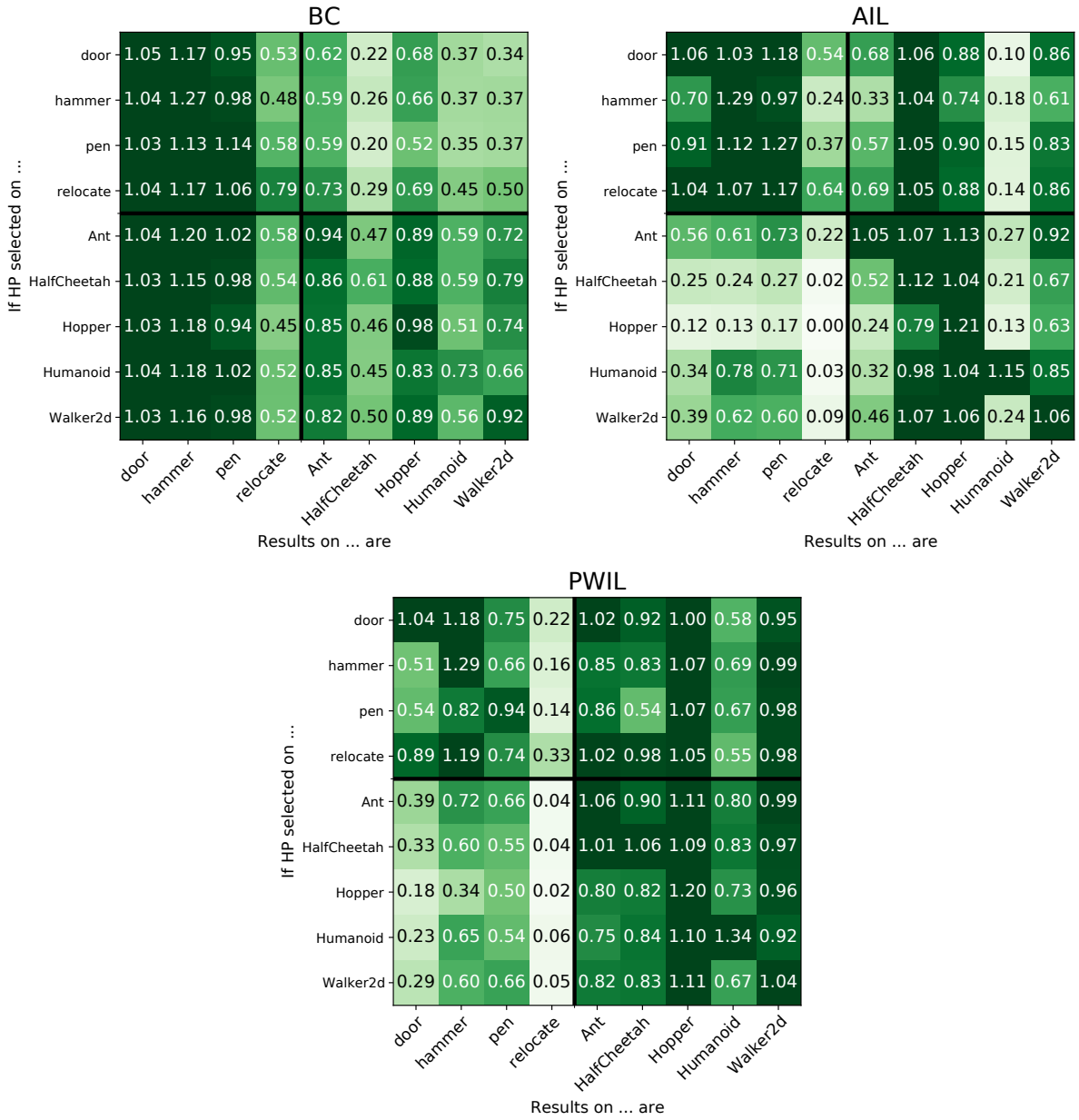


Figure C.3: HP transfer results for individual validation-test environments pairs. Rows correspond to different validation environments and columns to different test environments. Early stopping is performed using the oracle return.

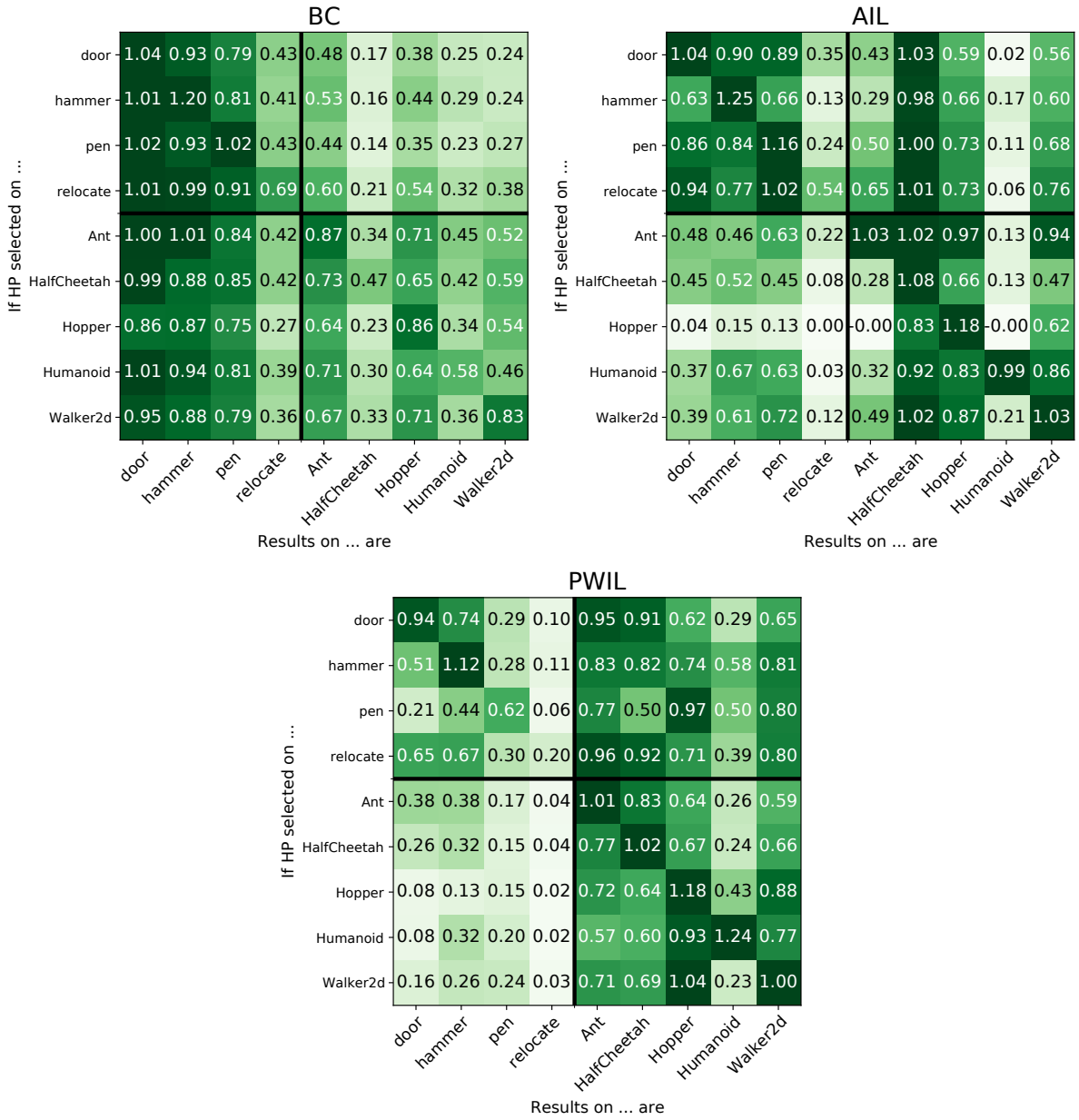


Figure C.4: HP transfer results for individual validation-test environments pairs. Rows correspond to different validation environments and columns to different test environments. No early stopping is performed.

Appendix D

Action Quantization from Demonstrations: Details and additional figures.

Contents

D.1 Ablation Study

In this section, we provide two ablations of the AQuaDQN algorithm. The first ablation is to learn a fixed set of actions independently of the state (which reduces to K-means). The second ablation consists in using random actions rather than the actions learned by the AQuaDem framework (the actions are given by the AQuaDem network, randomly initialized and not trained). We use the same hyperparameters as the one selected for AQuaDQN. In each case, for a number of actions in $\{5, 10, 25\}$, the success rate of the agent is 0 for all tasks throughout the training procedure.

D.2 Sanity Check Baselines

We provide in Figure D.1 the results of the SAC implementation from Hoffman et al. [2020] on the 5 classical OpenAI Gym environments, for 20M steps. The hyperparameters are the ones of the SAC paper [Haarnoja et al., 2018c] where the adaptive temperature was introduced. The results are consistent with the original paper and the larger relevant literature.

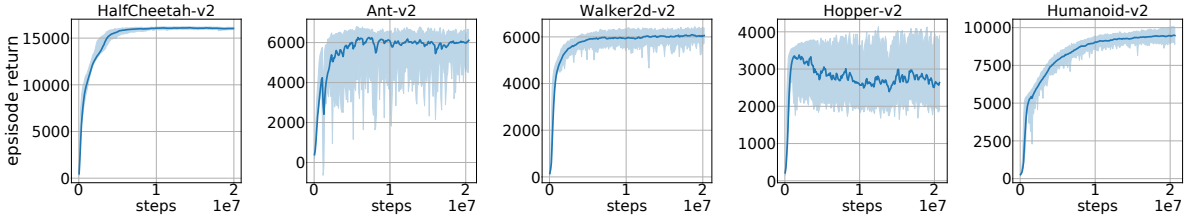


Figure D.1: SAC median and interquartile range on 10 seeds on the 5 Open Gym environments.

D.3 Offline Reinforcement Learning

In this section, we lead the analysis of the AQuaDem framework in the context of Offline Reinforcement Learning [Levine et al., 2020]. In the following, we no longer assume that the agent can interact with the environment, and learn a policy from a fixed set of transitions $\mathcal{D}$. We use the CQL [Kumar et al., 2020] algorithm together with the AQuaDem discretization. A simplistic variant of the CQL algorithm minimizes the following objective:

$$\min_Q \mathbb{E}_{s,a\sim\mathcal{D}}\Big[\alpha\big(\log\sum\exp(Q(s,.)) - Q(s,a)\big) + \frac{1}{2}(Q-\mathcal{T}^*Q)^2(s,a)\Big]. \tag{D.1}$$

The CQL loss defined in Equation (D.1) is natural for discrete action space, due to the logsumexp term, which is the regularizer of the Q-values that prevents out-of-distribution extrapolation). In continuous action spaces, the logsumexp term needs to be estimated. In the authors' implementation, the sampling logic is intricate has it relies on 3 different distributions. We evaluate the resulting algorithm on the D4RL locomotion tasks and provide performance against state-of-the-art offline RL algorithms. We use the results reported by Kostrikov et al. [2021]. We provide results in Table D.1 and show that AQuaCQL is competitive with considered baselines.

The architecture of the AQuaDem network has a common 3 layers of size 256 with relu activation, and a subsequent hidden layer of size 256 with relu activation for each action. We do not use dropout regularization and trained the network for 5.10^6 steps with batches of size 256. We use $\alpha = 5$ in the CQL loss (Equation (D.1)) and we use Munchausen DQN, with the same hyperparameters as AQuaDQN (Section D.4.4) except for the discount factor ($\gamma = 0.995$) and train it for 10^6 gradient steps.

Environment	BC	TD3+BC	CQL	IQL	AQuaCQL
halfcheetah-medium-v2	42.6	**48.3**	44.0	**47.4**	44.5 ± 2.5
hopper-medium-v2	52.9	59.3	58.5	**66.3**	58.5 ± 2.5
walker2d-medium-v2	75.3	**83.7**	72.5	78.3	82.1 ± 0.4
halfcheetah-medium-replay-v2	36.6	**44.6**	**45.5**	**44.2**	40.5 ± 2.5
hopper-medium-replay-v2	18.1	60.9	**95.0**	**94.7**	90.3 ± 2.1
walker2d-medium-replay-v2	26.0	**81.8**	77.2	73.9	80.8 ± 1.43
halfcheetah-medium-expert-v2	55.2	**90.7**	**91.6**	86.7	**88.3 ± 3**
hopper-medium-expert-v2	52.5	98.0	**105.4**	91.5	86.7 ± 6.9
walker2d-medium-expert-v2	**107.5**	**110.1**	**108.8**	**109.6**	108.1 ± 0.3
total	466.7	**677.4**	**698.5**	**692.4**	**679.9 ± 22.**

Table D.1: Averaged normalized scores on MuJoCo locomotion tasks. The AQuaCQL results are averaged over 10 seeds and 10 evaluation episodes.

D.4 Implementation

D.4.1 Grid World Visualizations

We learn the discretization of the action space using the AQuaDem framework. The architecture of the network is a common hidden layer of size 256 with relu activation, and a subsequent hidden layer of size 256 with relu activation for each action. We minimize the AQuaDem loss using the Adam optimizer with the learning rate 0.0003 and the dropout regularization rate 0.1 on 20000 gradient steps.

D.4.2 Environments

We considered the Adroit environments and the Robodesk environments, for which we described the observation space and the action space in Table D.2.

Environment	Observation Space	Action Space
Door	39	28
Hammer	46	26
Pen	45	24
Relocate	39	30
Robodesk	76	5

Table D.2: Environment description of the Adroit and Robodesk observation and action space.

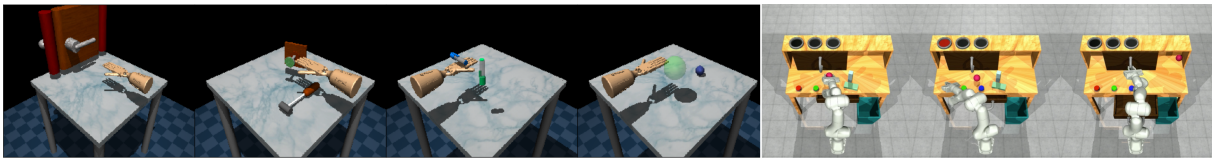


Figure D.2: Visualizations of the Adroit and Robodesk environments.

Adroit The Adroit environments [Rajeswaran et al., 2017] consists in a shadow hand solving 4 tasks (Figure D.2). The environments come with demonstrations which are gathered using virtual reality by a human.

Robodesk The Robodesk environment [Kannan et al., 2021] consists of a simulated Franka Emika Panda robot interacting with a table where multiple tasks are possible. The version of the simulated robot in the Robodesk environment only includes 5 DoFs (vs the 7 DoFs available, 2 were made not controllable). We evaluate AQuaPlay on the 9 base tasks described in Robodesk: `open_slide`, `open_drawer`, `push_green`, `stack`, `upright_block_off_table`, `flat_block_in_bin`, `flat_block_in_shelf`, `lift_upright_block`, `lift_ball`.

We used the RLDS creator `github.com/google-research/rlds-creator` to generate play data, together with a Nintendo Switch Pro Controller. The data is composed by 50 episodes of approximately 3 minutes where the goal of the demonstrator is to *interact* with the different elements of the environment.

D.4.3 Hyperparameter Selection Procedure

In the section we provide the hyperparameter selection procedure for the different setups. For the RLfD setting (Section 7.4.1) and the IL setting (Section 7.4.2) the number of hyperparameters is prohibitive to perform grid search. Therefore, we propose to sample hyperparameters uniformly within the set of all possible hyperparameters. For each environment, we sample 1000 configurations of hyperparameters, and train each algorithm including the baselines. We compute the average success rate of each individual value on the top 50% of all corresponding configurations (since poorly performing configurations are less informative) and select the best performing hyperparameter value independently. This procedure enables to **1)** limit combinatorial explosion with the number of hyperparameters **2)** provide a fair evaluation between the baselines and the proposed algorithms as they all rely on the same amount of compute. In the the website, we provide histograms detailing the influence of each hyperparameter. For the RLfP setting, we fixed the parameters related to the DQN algorithm with the ones selected in the RLfD setting to limit the hyperparameter search, which enables to perform grid search for 3 seeds, and select the best set of hyperparameters.

D.4.4 Reinforcement Learning with demonstrations

AQuaDQN

We learn the discretization of the action space using the AQuaDem framework. The architecture of the network is a common hidden layer of size 256 with relu activation, and a subsequent hidden layer of size 256 with relu activation for each action. We minimize the AQuaDem loss using the Adam optimizer and dropout regularization.

We train a DQN agent on top of the discretization learned by the AQuaDem framework. The architecture of the Q-network we use is the default LayerNorm architecture from the Q-network of the ACME library [Hoffman et al., 2020], which consists in a hidden layer of size 512 with layer normalization and tanh activation, followed by two hidden layers of sizes 512 and 256 with elu activation. We explored multiple Q-value losses for which we used the Adam optimizer: regular DQN [Mnih et al., 2013], double DQN with experience replay [Van Hasselt et al., 2016, Schaul et al., 2016], and Munchausen DQN [Vieillard et al., 2020]; the latter led to the best performance. We maintain a fixed ratio of demonstration episodes and agent episodes in the replay buffer similarly to Hester et al. [2018]. We also provide as a hyperparameter an optional minimum reward to the transitions of the expert to have a denser reward signal. The hyperparameter sweep for AQuaDQN can be found in Table D.3. The complete breakdown of the influence of each hyperparameter is provided on the website.

Hyperparameter	Possible values
aquadem learning rate	3e-5, 0.0001, **0.0003**, 0.001, 0.003
aquadem input dropout rate	0, **0.1**, 0.3
aquadem hidden dropout rate	0, **0.1**, 0.3
aquadem temperature	0.0001, **0.001**, 0.01
aquadem # actions	**10**, 15, 20
dqn learning rate	0.00003, **0.0001**, 0.003
dqn n step	1, **3**, 5
dqn epsilon	0.001, 0.01, **0.1**
dqn ratio of demonstrations	0, 0.1, **0.25**, 0.5
dqn min reward of demonstrations	None, **0.01**

Table D.3: Hyperparameter sweep for the AQuaDQN agent

When selecting hyperparameters specifically for Relocate, for Figure 7.6, the main difference in the chosen values is a dropout rate set to 0.

SAC and SACfD

We reproduced the authors' implementations (with an adaptive temperature) and use MLP networks for both the actor and the critic with two hidden layers of size 256 with relu activation. We use an Adam optimizer to train the SAC losses. We use a replay buffer of size 1M, and sample batches of size 256. We introduce a parameter of gradient updates frequency n which indicates a number of n gradient updates on the SAC losses every n environment steps. SACfD is a version of SAC inspired by DDPGfD [Vecerik et al., 2017] where we add expert demonstrations to the replay buffer of the SAC agent with a ratio between the agent episodes and the demonstration episodes which is a hyperparameter. We also provide as a hyperparameter an optional minimum reward to the transitions of the expert to have a denser reward signal. We found that the best hyperparameters for SAC are the same for SACfD. The HP sweep for SAC and SACfD can be found in Table D.4 and Table D.5. The complete breakdown of the influence of each hyperparameter is provided on the website.

Hyperparameter	Possible values
learning rate	3e-5, **1e-4**, 0.0003
n step	1, 3, **5**
tau	**0.005**, 0.01, 0.05
reward scale	0.1, 0.3, **0.5**

Table D.4: Hyperparameter sweep for SAC.

Hyperparameter	Possible values
learning rate	3e-5, **1e-4**, 0.0003
n step	1, 3, **5**
tau	**0.005**, 0.01, 0.05
reward scale	0.1, **0.3**, 0.5
ratio of demonstrations	0, **0.001**, 0.1, 0.25
mini reward of demonstrations	None, **0.01**, **0.1**

Table D.5: Hyperparameter sweep for SACfD.

D.4.5 Imitation Learning

AQuaGAIL

We learn the discretization of the action space using the AQuaDem framework. The architecture ofthe network is a common hidden layer of size 256 with relu activation, and a subsequent hidden layer of size 256 with relu activation for each action. We minimize the AQuaDem loss using the Adam optimizer and dropout regularization. The discriminator is a MLP whose number of layers, number of units per layers are hyperparameters. We use the Adam optimizer with two possible regularization scheme: dropout and weight decay. The discriminator outputs a value p from which we compute three possible rewards $-\log(p)$, $-0.5\log(p) + \log(1-p)$, $\log(1-p)$ corresponding to the reward balance hyperparameter. The direct RL algorithm is Munchausen DQN, with the same architecture and hyperparameters described in Section D.4.4. The hyperparameter sweep for AQuaGAIL can be found in Table D.6. The complete breakdown of the influence of each hyperparameter is provided on the website.

GAIL

We used the same discriminator architecture and hyperparameters as the one described in Section D.4.5. The direct RL agent is the SAC algorithm whose architecture and hyperparameters are described in Section D.4.4. The hyperparameter sweep for GAIL can be found in Table D.7. The complete breakdown of the influence of each hyperparameter is provided on the website.

Behavioral Cloning

The BC network is a MLP whose number of layers, number of units per layers and activation functions are hyperparameters. We use the Adam optimizer with two possible regularization scheme: dropout and weight decay. The observation normalization hyperparameter is set to True when each dimension of the

Hyperparameter	Possible values
discriminator learning rate	1e-7, 3e-7, **1e-6**, 3e-5, 1e-4
discriminator num layers	**1**, 2
discriminator num units	16, **64**, 256
discriminator regularization	none, **dropout**, weight decay
discriminator weight decay	5, 10, 20
discriminator input dropout rate	**0.5**, 0.75
discriminator hidden dropout rate	**0.5**, 0.75
discriminator observation normalization	**True**, False
discriminator reward balance	0., **0.5**, 1.
dqn learning rate	**3e-5**, 1e-4, 3e-4
dqn n step	**1**, 3, 5
dqn epsilon	0.001, **0.01**, 0.1
aquadem learning rate	3e-5, 1e-4, **3e-4**, 1e-3, 3e-3
aquadem temperature	0.0001, **0.001**, 0.01
aquadem num actions	**10**, 15, 20
aquadem input dropout rate	**0**, 0.1, 0.3
aquadem hidden dropout rate	**0**, 0.1, 0.3

Table D.6: Hyperparameter sweep for the AQuaGAIL agent.

Hyperparameter	Possible values
discriminator learning rate	1e-7, **3e-7**, 1e-6, 3e-5, 1e-4
discriminator num layers	**1**, 2
discriminator num units	16, 64, **256**
discriminator regularization	none, dropout, **weight decay**
discriminator weight decay	5, **10**, 20
discriminator input dropout rate	0.5, 0.75
discriminator hidden dropout rate	0.5, 0.75
discriminator observation normalization	**True**, False
discriminator reward balance	0., **0.5**, 1.
sac learning rate	3e-5, **1e-4**, 3e-4
sac n step	1, 3, **5**
sac tau	0.005, 0.01, **0.05**
sac reward scale	0.1, 0.3, **0.5**

Table D.7: Hyperparameter sweep for the discriminator part of the GAIL agent.

observation are centered with the mean and standard deviation of the observations in the demonstration dataset. The complete breakdown of the influence of each hyperparameter is provided on the website.

Hyperparameter	Possible values
learning rate	1e-5, 3e-5, 1e-4, **3e-4**, 1e-3
num layers	**1**, 2. 3
num units	16, 64, **256**
activation	relu, **tanh**
observation normalization	**True**, False
weight decay	**0**, .01, 0.1
input dropout rate	**0**, 0.15, 0.3
hidden dropout rate	**0**, 0.25, 0.5

Table D.8: Hyperparameter sweep for the BC agent.

D.4.6 Reinforcement Learning with play data

SAC

We used the exact same implementation as the one described in Section D.4.4. The HP sweep can be found in Table D.9.

Hyperparameter	Possible values
learning rate	1e-5, 3e-5, 3e-4, **1e-4**
n step	1, 3, **5**
reward scale	0.1, **1**, 10
tau	0.005, **0.01**, 0.05

Table D.9: Hyperparameter sweep for SAC for the Robodesk environment. The best hyperparameter set was chosen as the one that maximizes the performance on average on all tasks.

DQN with Naive Discretization

We used the exact same implementation as the one described in Section D.4.4, and also use the best hyperparameters found in the RLfD setting. As the action space is $(-1, 1)^5$, we use three different discretization meshes: $\{-1, 1\}$, $\{-1, 0, 1\}$, $\{-1, -0.5, 0., 0.5, 1\}$ which induce a discrete action space of dimension $2^5, 3^5, 5^5$ respectively. We refer to the resulting algorithm as BB-2, BB-3, and BB-5 (where BB stands for "Bang-bang").

AQuaPlay

We used the exact same implementation as the one described in Section D.4.4, and also use the best hyperparameters found in the RLfD setting for the Munchausen DQN agent. We performed a sweep on the discretization step that we report in Table D.10.

Hyperparameter	Possible values
learning rate	0.0001, 0.0003, **0.001**
dropout rate	0, **0.1**, 0.3
temperature	**1e-4**, 1e-3, 1e-2
# actions	10, 20, **30**, 40

Table D.10: Hyperparameter sweep for the AQuaPlay agent. The best hyperparameter set was chosen as the one that maximizes the performance on average on all tasks.