# Continuous Control with Action Quantization from Demonstrations

**Robert Dadashi** [*1] **Léonard Hussenot** [*1 2] **Damien Vincent** [1] **Sertan Girgin** [1]
**Anton Raichuk** [1] **Matthieu Geist** [1] **Olivier Pietquin** [1]

## Abstract

In this paper, we propose a novel Reinforcement Learning (RL) framework for problems with continuous action spaces: Action Quantization from Demonstrations (AQuaDem). The proposed approach consists in learning a discretization of continuous action spaces from human demonstrations. This discretization returns a set of plausible actions (in light of the demonstrations) for each input state, thus capturing the priors of the demonstrator and their multimodal behavior. By discretizing the action space, any discrete action deep RL technique can be readily applied to the continuous control problem. Experiments show that the proposed approach outperforms state-of-the-art methods such as SAC in the RL setup, and GAIL in the Imitation Learning setup. We provide a website with interactive videos: https://google-research.github.io/aquadem/ and make the code available: https://github.com/google-research/google-research/tree/master/aquadem.

## 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., 2020), Reinforcement Learning (RL) holds a tremendous potential for solving sequential decision making problems. RL relies on Markov Decision Processes (MDP) (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 discrete action setup, where an agent faces a countable set of actions, and the continuous action setup, where an agent faces an uncountable set of actions. When the number of actions is finite and small, computing the maximum of the action-value function is straightforward (and implicitly defines a greedy 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 & 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 & Agrawal, 2020; Andrychowicz et al., 2020) 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 space to a discrete set of meaningful actions. Demonstrations typically consist of transitions experienced by a human in the targeted environment, performing the task at hand or interacting without any specific goal *i.e.* play. The side effect of using demonstrations to discretize the action space is to filter out useless/hazardous actions, thus focusing the search on relevant actions and possibly facilitating exploration. Besides, using a set of actions rather than a single one (Behavioral Cloning) enables to capture the multimodality of behaviors in the demonstrations.

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. We formalize this paradigm, provide a neural im-

[*]Equal contribution [1]Google Research, Brain Team [2]Univ. de Lille, CNRS, Inria Scool, UMR 9189 CRIStAL. Correspondence to: Robert Dadashi <dadashi@google.com>.

*Proceedings of the $39^{th}$ International Conference on Machine Learning*, Baltimore, Maryland, USA, PMLR 162, 2022. 

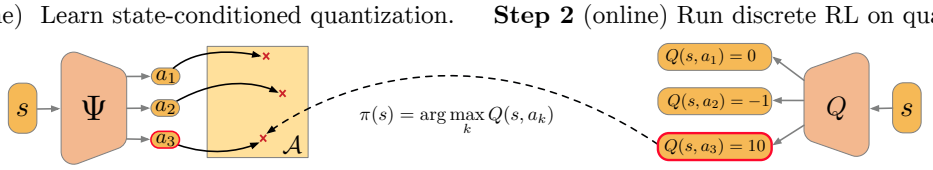


*Figure 1.* Visualization of the AQuaDem framework (offline) with a downstream algorithm (online).

plementation 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 (demonstrations of a human playing in an environment but not solving any specific task), and Imitation Learning. We test the resulting methods on challenging manipulation tasks and show that they outperform state-of-the-art continuous control methods both in terms of sample-efficiency and performance on every setup.

## 2. Preliminaries

**Markov Decision Process.** We model the sequential decision making problem as a Markov Decision Process (MDP) (Puterman, 2014; Sutton & Barto, 2018). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition kernel, $r$ is the expected reward function, $\gamma$ the discount factor and $\rho_0$ the initial state distribution. Throughout the paper, we distinguish discrete action spaces, which simply amount to a set $\{1, \dots, K\}$, from continuous action spaces which consist in an interval of $\mathbb{R}^d$ where $d$ is the dimensionality of the action space. A stationary policy $\pi$ is a mapping from states to distributions over actions. The value function $V^\pi$ of a policy $\pi$ is defined as the expected discounted cumulative reward from starting in a particular state and acting according to $\pi$: $V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_t \sim \pi(s_t), s_{t+1} \sim \mathcal{P}(s_t, a_t))\right]$. An optimal policy $\pi^*$ maximizes the value function $V^{\pi^*}$ for all states. The action-value function $Q^\pi$ is defined as the expected discounted cumulative reward from starting in a particular state, taking an action and then acting according to $\pi$: $Q^\pi(s, a) = r(s, a) + \gamma\, \mathbb{E}\left[V^\pi(s') | s' \sim \mathcal{P}(s, a)\right]$.

**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 $s'$: $\mathcal{T}^\pi(Q)(s, a) := r(s, a) + \gamma\, \mathbb{E}\left[Q(s', a') | a' \sim \pi(s), s' \sim \mathcal{P}(s, a)\right]$. Value Iteration (VI) (Bertsekas, 2000) is the basis for methods using the Bellman equation to derive algorithms estimating the optimal policy $\pi^*$. The prototypical example is the $Q$-learning algorithm (Watkins & Dayan, 1992), which is the basis of *e.g.* DQN (Mnih et al., 2015), 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 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.

## 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.

### 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 $k^{\text{th}}$ 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 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 relevant 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 $f_\Phi$ that minimizes the reconstruction error between predicted actions and actions in the dataset $\mathcal{D}$. To ease notations, we will conflate the function $f_\Phi$ with its parameters $\Phi$ and simply note it $\Phi : \mathcal{S} \mapsto \mathcal{A}$. The objective is thus to minimize: $\min_\Phi \mathbb{E}_{s,a\sim\mathcal{D}} \|\Phi(s) - a\|^2$. Instead, we propose to learn a set of $K$ actions $\Psi_k(s)$ for each state by minimizing the following loss:

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

where the temperature $T$ is a hyperparameter. Equation (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 3.2 and provite a formal justification in Appendix B. Equation (1) is also interpretable in the context of Gaussian mixture models (see Appendix A). The $\Psi$ 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 $\Psi$ at each state.

### 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 2.

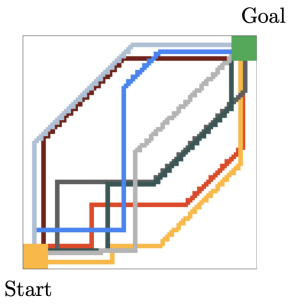


*Figure 2.* Grid world environment with a start state (bottom left), and a goal state (top right). Actions are continuous, and give the direction in which the agent take a step with fixed length. The stochastic demonstrator moves either right or up in the bottom left of the environment then moves diagonally until reaching and edge and goes either up or right to the target. The demonstrations are represented in the different colors.

We define a neural network $\Psi$ and optimize its parameters by minimizing the objective function defined in Equation (1) (implementation details can be found in Appendix G.1). We display the resulting candidate actions across the state space in Figure 3. As each color of the arrows depicts a single head of the $\Psi$ 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 3) 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 and in the analysis of the actions learned in a more realistic setup in Appendix C.

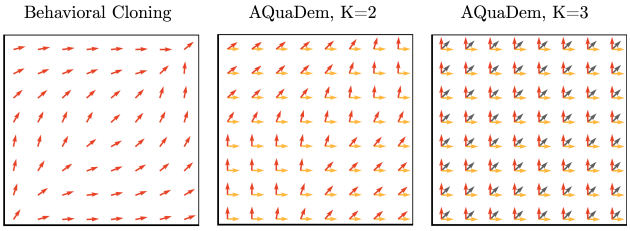


*Figure 3.* Visualisation of the actions learned by BC and the candidate actions learned with AQuaDem for $K = 2$ and $K = 3$ and $T = 0.01$. Each color represents a head of the $\Psi$ network.

**Influence of the temperature.** The temperature controls the degree of smoothness of the soft-minimum defined in Equation (1). We show that with larger temperatures, the soft-minimum converges to the average which is well represented in Figure 4 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.

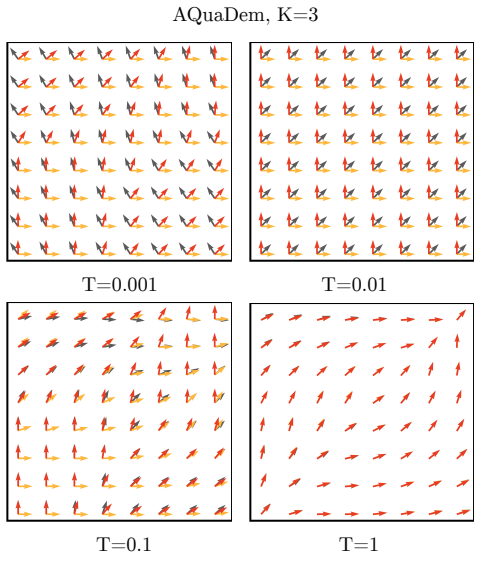


*Figure 4.* Influence of the temperature on resulting candidate actions learned with AQuaDem.

### 3.3. 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.

## 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 and we provide videos of all the agents trained in the website. We also provide results for Offline RL in Appendix F.

### 4.1. Reinforcement Learning with demonstrations

**Setup.** In the Reinforcement Learning with 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

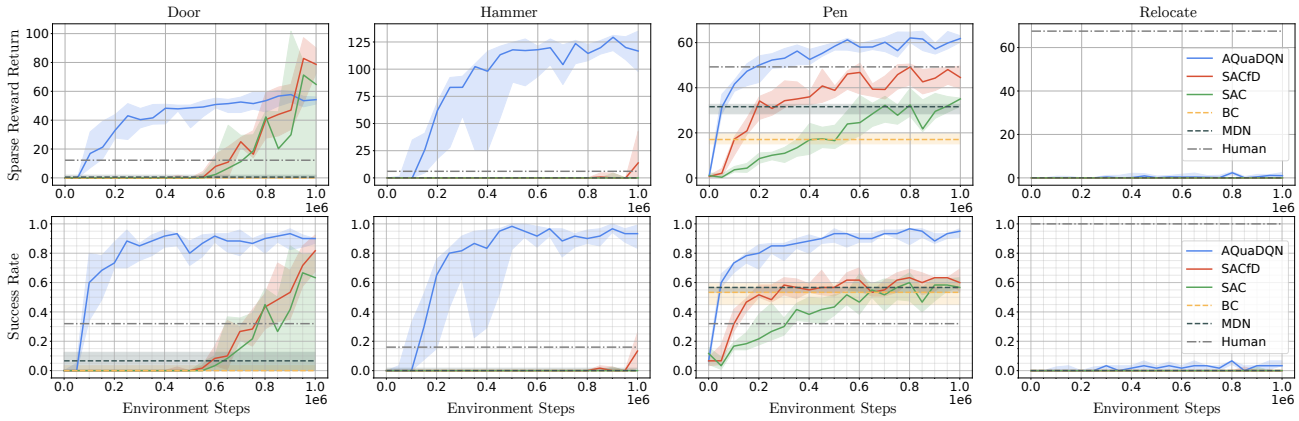


*Figure 5.* Performance of AQuaDQN against SAC and SACfD. Agents are evaluated every 50k environment steps over 30 episodes. We represent the median performance in terms of success rate (bottom) and returns (top) as well as the interquartile range over 10 seeds.

hard. We consider the Adroit tasks (Rajeswaran et al., 2017) represented in Figure 11, 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., 2015) 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. The implementation is from Acme (Hoffman et al., 2020). 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.

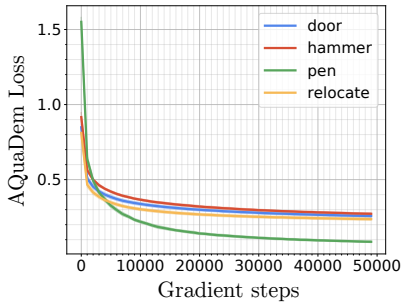


*Figure 6.* AQuaDem discretization loss.

**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 6 shows the AQuaDem loss through the training procedure of the discretization step, and the Figure 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.

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 that when

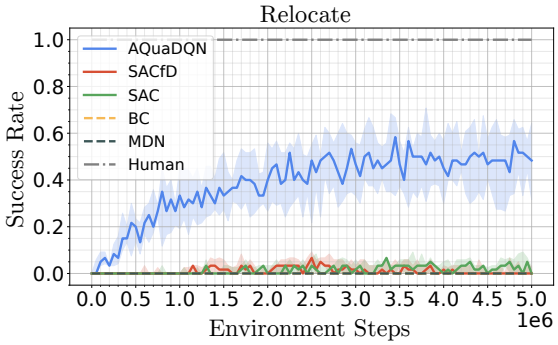


*Figure 7.* Performance of AQuaDQN against SAC and SACfD baselines when all are tuned on the Relocate environment. We represent the median performance in terms of success rate as well as the interquartile range over 10 seeds.

tuned uniquely on the Relocate environment and with more environment interactions, AQuaDQN manages to reach a 50% 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 with 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 (https://youtu.be/MyZzfA7RFnw).

### 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., 2021). A number of existing works (Ho & Ermon, 2016; Ghasemipour et al., 2019; Dadashi et al., 2021) 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 & 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) as well as BC with multiple Mixture Density Networks (MDN) (Bishop, 1994).

**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 and MDN are 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 8. 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 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, BC and MDN. Notice that our results corroborate previous work (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, GAIL and MDN 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 (https://youtu.be/PxNIPx93slk).

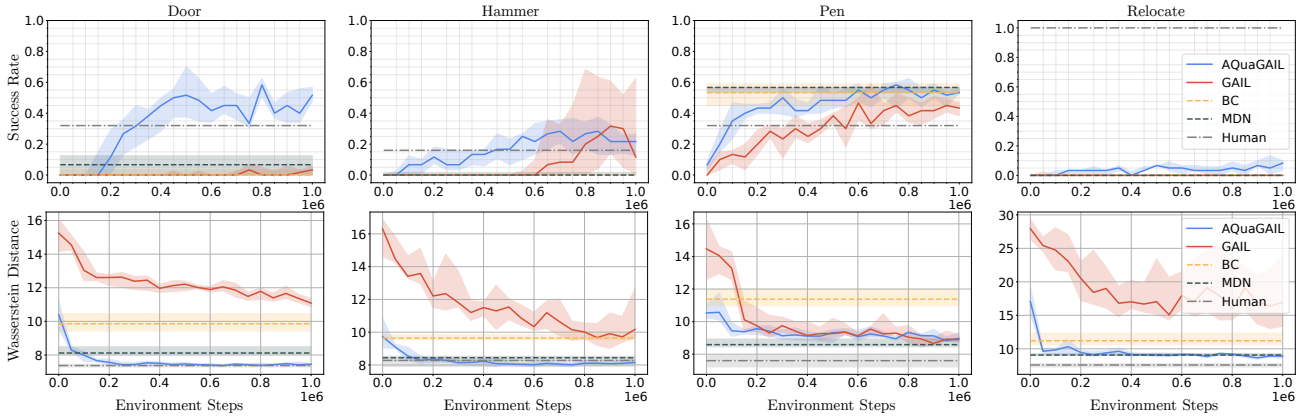


*Figure 8.* Performance of AQuaGAIL against GAIL, BC and MDN baselines. Agents are evaluated every 50k environment steps over 30 episodes. We represent the median success rate (top) on the task as well as the Wasserstein distance (bottom) of the agent's state distribution to the expert's state distribution as well as the interquartile range over 10 seeds.

### 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 11, for which we acquired play data. We expand on the environment as well as the data collection procedure in the Appendix G.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 9. 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.

## 5. 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., 2020; Vieillard et al., 2021; Tang

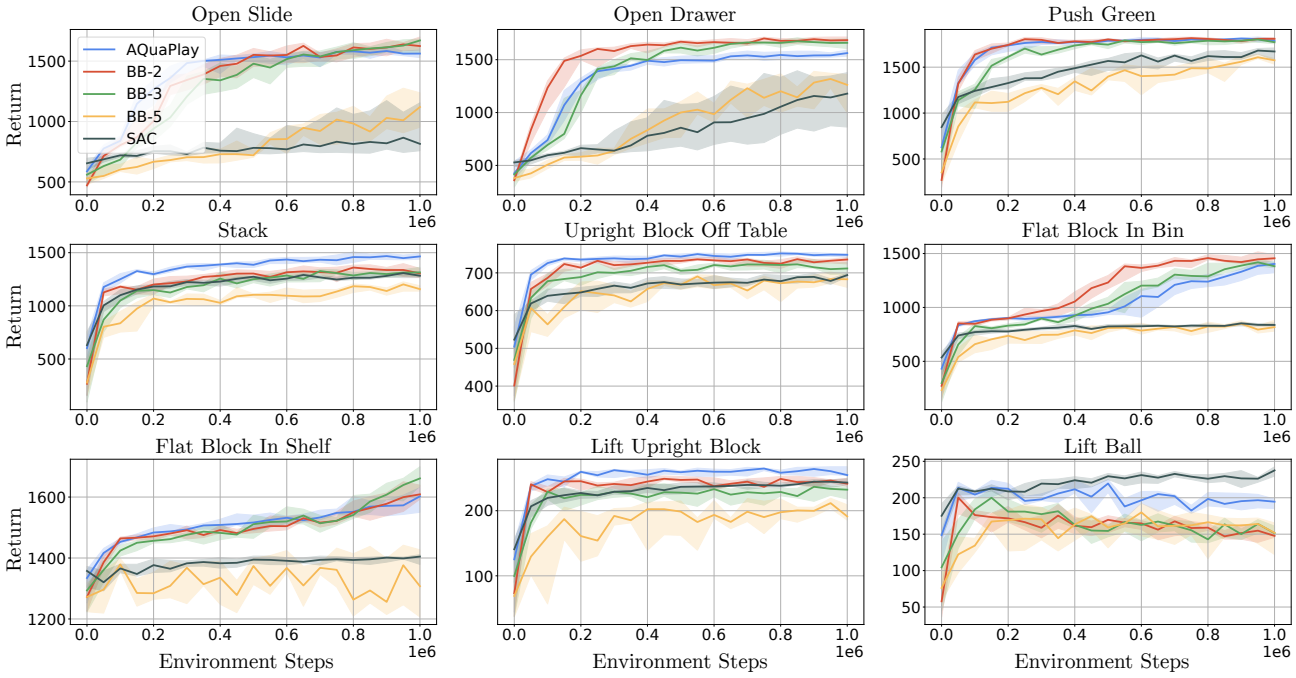


*Figure 9.* Performance of AQuaPlay against SAC and "bang-bang" baselines. Agents are evaluated every 50k environment steps over 30 episodes. We represent the median return as well as the interquartile range over 10 seeds.

& 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 parametrization of continuous control methods, including SAC, by a Bernoulli parametrization to sample extremal actions (still relying on a sampling-based optimisation 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., 2015; 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 & 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 *small* discrete action spaces as in 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 & Veloso (2007); Calinon & 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.

## 6. Perspectives and Conclusion

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. 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., 2019; Wang et al., 2019) or in offline RL (Fujimoto & 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. 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.

## References


Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *International Conference on Machine Learning*, 2017. 8

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 2020. 1, 7

Asadi, K., Parikh, N., Parr, R. E., Konidaris, G. D., and Littman, M. L. Deep radial-basis value functions for continuous control. In *AAAI Conference on Artificial Intelligence*, 2021. 8

Barto, A., Mirolli, M., and Baldassarre, G. Novelty or surprise? *Frontiers in psychology*, 2013. 4

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013. 5

Bellman, R. A markovian decision process. *Indiana University Mathematics Journal*, 1957. 2

Bellman, R., Glicksberg, I., and Gross, O. On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 1956. 7

Berner, C., Brockman, G., Chan, B., Cheung, V., Dkbiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 1

Bertsekas, D. P. *Dynamic programming and optimal control*. Athena scientific Belmont, 2000. 2

Bishop, C. M. Mixture density networks. 1994. 6, 14

Brantley, K., Sun, W., and Henaff, M. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2019. 9

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 2012. 9

Bushaw, D. W. Differential equations with a discontinuous forcing term. Technical report, Stevens Inst Of Tech Hoboken NJ Experimental Towing Tank, 1953. 1, 7

Calinon, S. and Billard, A. Incremental learning of gestures by imitation in a humanoid robot. In *ACM/IEEE international conference on Human-robot interaction*, 2007. 9

Chernova, S. and Veloso, M. Confidence-based policy learning from demonstration using gaussian mixture models. In *International Conference on Autonomous Agents and Multiagent Systems*, 2007. 9

Dadashi, R., Hussenot, L., Geist, M., and Pietquin, O. Primal wasserstein imitation learning. *International Conference on Learning Representations*, 2021. 6

Ding, Y., Florensa, C., Phielipp, M., and Abbeel, P. Goal-conditioned imitation learning. *Advances in Neural Information Processing Systems*, 2019. 9

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015. 8

Flamary, R., Courty, N., Gramfort, A., Alaya, M. Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N., Gautheron, L., Gayraud, N. T., Janati, H., Rakotomamonjy, A., Redko, I., Rolet, A., Schutz, A., Seguy, V., Sutherland, D. J., Tavenard, R., Tong, A., and Vayer, T. Pot: Python optimal transport. *Journal of Machine Learning Research*, 2021. 6

Florence, P., Lynch, C., Zeng, A., Ramirez, O., Wahid, A., Downs, L., Wong, A., Lee, J., Mordatch, I., and Tompson, J. Implicit behavioral cloning. *arXiv preprint arXiv:2109.00137*, 2021. 9

Fox, R., Berenstein, R., Stoica, I., and Goldberg, K. Multi-task hierarchical imitation learning for home automation. In *International Conference on Automation Science and Engineering (CASE)*. IEEE, 2019. 9

Fujimoto, S. and Gu, S. S. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021. 9

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018. 2, 8

Ghasemipour, S. K. S., Zemel, R., and Gu, S. A divergence minimization perspective on imitation learning methods. *Conference on Robot Learning*, 2019. 6

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, 2016. 8

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019. 7

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018a. 2, 6, 8

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b. 15

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, 2018. 2, 5, 17

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016. 6

Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. 5, 15, 17

Hussenot, L., Andrychowicz, M., Vincent, D., Dadashi, R., Raichuk, A., Ramos, S., Momchev, N., Girgin, S., Marinier, R., Stafiniak, L., et al. Hyperparameter selection for imitation learning. In *International Conference on Machine Learning*, 2021. 6

Jarrett, D., Bica, I., and van der Schaar, M. Strictly batch imitation learning by energy-based distribution matching. *Advances in Neural Information Processing Systems*, 2020. 9

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *Conference on Robot Learning*, 2018. 8

Kannan, H., Hafner, D., Finn, C., and Erhan, D. Robodesk: A multi-task reinforcement learning benchmark. `https://github.com/google-research/robodesk`, 2021. 7, 17

Kipf, T., Li, Y., Dai, H., Zambaldi, V., Sanchez-Gonzalez, A., Grefenstette, E., Kohli, P., and Battaglia, P. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, 2019. 9

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 2000. 1, 8

Kostrikov, I., Agrawal, K. K., Dwibedi, D., Levine, S., and Tompson, J. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *International Conference on Learning Representations*, 2019. 6

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021. 15

Krishnan, S., Fox, R., Stoica, I., and Goldberg, K. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning*, 2017. 9

Kroemer, O., Daniel, C., Neumann, G., Van Hoof, H., and Peters, J. Towards learning hierarchical skills for multi-phase manipulation tasks. In *International Conference on Robotics and Automation*. IEEE, 2015. 8

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020. 15

Le, H., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., and Daumé, H. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning*, 2018. 9

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020. 15

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016. 8

Lim, S., Joseph, A., Le, L., Pan, Y., and White, M. Actor-expert: A framework for using q-learning in continuous action spaces. *arXiv preprint arXiv:1810.09103*, 2018. 8

Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. Learning latent plans from play. In *Conference on Robot Learning*, 2020. 2, 4, 7, 9

Mandlekar, A., Xu, D., Wong, J., Nasiriany, S., Wang, C., Kulkarni, R., Fei-Fei, L., Savarese, S., Zhu, Y., and Martín-Martín, R. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021. 4

Manschitz, S., Kober, J., Gienger, M., and Peters, J. Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems*, 2015. 9

Metz, L., Ibarz, J., Jaitly, N., and Davidson, J. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017. 1, 8

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 2015. 2, 5, 17

Neunert, M., Abdolmaleki, A., Wulfmeier, M., Lampe, T., Springenberg, T., Hafner, R., Romano, F., Buchli, J., Heess, N., and Riedmiller, M. Continuous-discrete reinforcement learning for hybrid control in robotics. In *Conference on Robot Learning*, 2020. 8

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999. 2

Ng, A. Y., Russell, S. J., et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000. 2

Orsini, M., Raichuk, A., Hussenot, L., Vincent, D., Dadashi, R., Girgin, S., Geist, M., Bachem, O., Pietquin, O., and Andrychowicz, M. What matters for adversarial imitation learning? *arXiv preprint arXiv:2106.00672*, 2021. 6

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation*. IEEE, 2009. 9

Pomerleau, D. A. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 1991. 2, 3, 6

Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. 1, 2

Rahmatizadeh, R., Abolghasemi, P., Behal, A., and Bölöni, L. From virtual demonstration to real-world manipulation using lstm and mdn. In *AAAI Conference on Artificial Intelligence*, 2018. 9

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *Robotics: Science and Systems*, 2017. 5, 17

Ryu, M., Chow, Y., Anderson, R., Tjandraatmadja, C., and Boutilier, C. Caql: Continuous action q-learning. *International Conference on Learning Representations*, 2020. 8

Sakryukin, A., Raissi, C., and Kankanhalli, M. Inferring DQN structure for high-dimensional continuous control. In *International Conference on Machine Learning*, 2020. 1, 8

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *International Conference on Learning Representations*, 2016. 17

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, 2015. 8

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 8

Seyde, T., Gilitschenski, I., Schwarting, W., Stellato, B., Riedmiller, M., Wulfmeier, M., and Rus, D. Is bang-bang control all you need? solving continuous control with bernoulli policies. In *Advances in Neural Information Processing Systems*, 2021. 8

Shankar, T., Tulsiani, S., Pinto, L., and Gupta, A. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations*, 2019. 9

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016. 1

Simmons-Edler, R., Eisner, B., Mitchell, E., Seung, S., and Lee, D. Q-learning for continuous actions with cross-entropy guided policies. *arXiv preprint arXiv:1903.10605*, 2019. 8

Singh, A., Liu, H., Zhou, G., Yu, A., Rhinehart, N., and Levine, S. Parrot: Data-driven behavioral priors for reinforcement learning. *International Conference on Learning Representations*, 2020. 9

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018. 2

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 2000. 8

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017. 9

Tang, Y. and Agrawal, S. Discretizing continuous action space for on-policy optimization. In *AAAI Conference on Artificial Intelligence*, 2020. 1, 7

Tassa, Y., Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., and Heess, N. dm_control: Software and tasks for continuous control, 2020. 8

Tavakoli, A., Pardo, F., and Kormushev, P. Action branching architectures for deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2018. 1, 7

Tavakoli, A., Fatemi, M., and Kormushev, P. Learning to represent action values as a hypergraph on the action vertices. *Internation Conference in Learning Representations*, 2021. 1, 8

Tessler, C., Tennenholtz, G., and Mannor, S. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems*, 2019. 8

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, 2016. 17

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017. 2, 5, 17

Vieillard, N., Pietquin, O., and Geist, M. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 2020. 5, 17

Vieillard, N., Andrychowicz, M., Raichuk, A., Pietquin, O., and Geist, M. Implicitly regularized rl with implicit q-values. *arXiv preprint arXiv:2108.07041*, 2021. 7

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019. 1

Wang, R., Ciliberto, C., Amadori, P. V., and Demiris, Y. Random expert distillation: Imitation learning via expert policy support estimation. In *International Conference on Machine Learning*, 2019. 9

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 1992. 2

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992. 1, 8

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019. 9

Yu, T., Finn, C., Xie, A., Dasari, S., Zhang, T., Abbeel, P., and Levine, S. One-shot imitation from observing humans via domain-adaptive meta-learning. *Robotics: Science and Systems*, 2018. 9

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008. 2

## A. 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 $\sigma$ 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 (1) where the variance $\sigma^2$ 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 4.

## B. 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:

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

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

## C. Action visualization in a high-dimensional environment

For the Door environment (see Figure 11) we represent the actions candidates learned using the AQuaDem framework with videos that can be found in the the website. 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 $\Psi$ network). Interestingly, the video shows that actions are very state dependent (except some default 0-action) and evolve smoothly through time.

## D. 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.

## E. Sanity Check Baselines

We provide in Figure 10 the results of the SAC implementation from Acme (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., 2018b) where the adaptive temperature was introduced. The results are consistent with the original paper and the larger relevant literature.

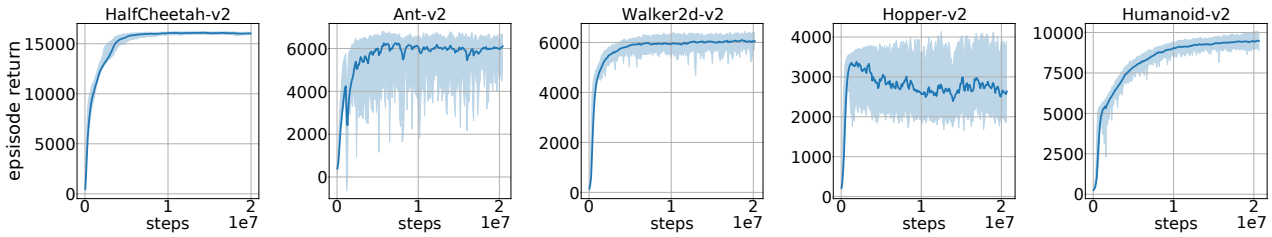


*Figure 10.* SAC median and interquartile range on 10 seeds on the 5 Open Gym environments.

## F. 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:

$$\min_{Q} \mathbb{E}_{s,a\sim\mathcal{D}} \Big[\alpha\big(\log \sum \exp(Q(s,.)) - Q(s,a)\big) + \frac{1}{2}(Q - \mathcal{T}^*Q)^2(s,a)\Big]. \tag{2}$$

The CQL loss defined in Equation (2) 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 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 (2)) and we use Munchausen DQN, with the same hyperparameters as AQuaDQN (Section G.4.1) 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 1.* Averaged normalized scores on MuJoCo locomotion tasks. The AQuaCQL results are averaged over 10 seeds and 10 evaluation episodes.

# G. Implementation

## G.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.

## G.2. Environments

We considered the Adroit environments and the Robodesk environments, for which we described the observation space and the action space in Table 2.

| Environment | Observation Space | Action Space |
|---|---|---|
| Door | 39 | 28 |
| Hammer | 46 | 26 |
| Pen | 45 | 24 |
| Relocate | 39 | 30 |
| Robodesk | 76 | 5 |

*Table 2.* Environment description of the Adroit and Robodesk observation and action space.

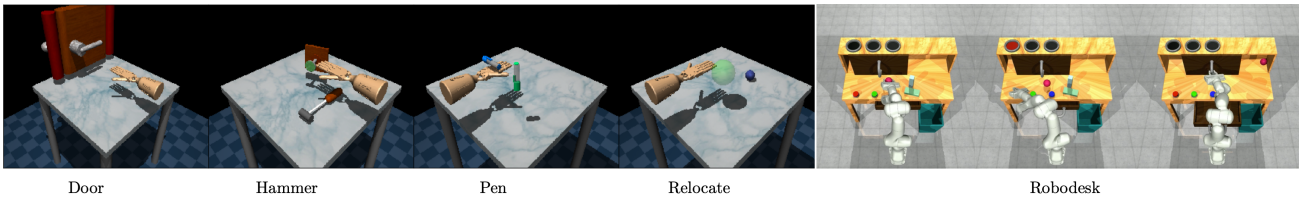


*Figure 11.* Visualizations of the Adroit and Robodesk environments.

**Adroit** The Adroit environments (Rajeswaran et al., 2017) consists in a shadow hand solving 4 tasks (Figure 11). 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.

### G.3. Hyperparameter Selection Procedure

In the section we provide the hyperparameter selection procedure for the different setups. For the RLfD setting (Section 4.1) and the IL setting (Section 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.

### G.4. Reinforcement Learning with demonstrations

#### G.4.1. 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., 2015), 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 3. The complete breakdown of the influence of each hyperparameter is provided in the website.

When selecting hyperparameters specifically for Relocate, for Figure 7, the main difference in the chosen values is a dropout rate set to 0.

#### G.4.2. 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

| 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 3.* Hyperparameter sweep for the AQuaDQN agent

the best hyperparameters for SAC are the same for SACfD. The HP sweep for SAC and SACfD can be found in Table 4 and Table 5. The complete breakdown of the influence of each hyperparameter is provided in 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 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 5.* Hyperparameter sweep for SACfD.

### G.5. Imitation Learning

#### G.5.1. 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 G.4.1. The hyperparameter sweep for AQuaGAIL can be found in Table 6. The complete breakdown of the influence of each hyperparameter is provided in the website

| 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 6.* Hyperparameter sweep for the AQuaGAIL agent.

### G.5.2. GAIL

We used the same discriminator architecture and hyperparameters as the one described in Section G.5.1. The direct RL agent is the SAC algorithm whose architecture and hyperparameters are described in Section G.4.2. The hyperparameter sweep for GAIL can be found in Table 7. The complete breakdown of the influence of each hyperparameter is provided in the website.

| 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 7.* Hyperparameter sweep for the discriminator part of the GAIL agent.

### G.5.3. 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 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 in 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 8.* Hyperparameter sweep for the BC agent.

#### G.5.4. MIXTURE DENSITY NETWORKS

The MDN network is trained similarly as the action candidates network from 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 add another head from the common hidden layer that represents the *logits* $p_i$ of each action candidate $a_i$. We train the MDN network with a modified version of the AQuaDem loss using the Adam optimizer and dropout regularization:

$$\begin{cases} \min_{\Psi} \mathop{\mathbb{E}}_{s,a\sim\mathcal{D}} \big[ -T \log \big( \sum_{k=1}^{K} \exp(\frac{-\|\Psi_k(s) - a\|^2}{T}) \big) \big], \\ \min_{p} \mathop{\mathbb{E}}_{s,a\sim\mathcal{D}} \big[ -\sum_{k=1}^{K} \frac{\exp(p_k)}{\sum_{k'} \exp(p_{k'})} \exp(\frac{-\|\Psi_k(s) - a\|^2}{T}) \big]. \end{cases}$$

At inference, we sample actions with respect to the logits.

| Hyperparameter | Possible values |
|---|---|
| learning rate | 3e-5, 0.0001, **0.0003**, 0.001, 0.003 |
| input dropout rate | **0**, 0.1 |
| hidden dropout rate | **0**, 0.1 |
| temperature | 0.0001, **0.001** |
| # actions | 5, **10**, 20 |

*Table 9.* Hyperparameter sweep for the MDN agent.

### G.6. Reinforcement Learning with play data

#### G.6.1. SAC

We used the exact same implementation as the one described in Section G.4.2. The HP sweep can be found in Table 10.

| 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 10.* 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.

### G.6.2. DQN WITH NAIVE DISCRETIZATION

We used the exact same implementation as the one described in Section G.4.1, 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").

### G.6.3. AQUAPLAY

We used the exact same implementation as the one described in Section G.4.1, 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 11.

| 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 11.* Hyperparameter sweep for the AQuaPlay agent. The best hyperparameter set was chosen as the one that maximizes the performance on average on all tasks.