Supplementary Material

A. Proofs

A.1. Proof of Lemma 7

Proof. Consider the following MDP:

Figure 6. In UNICYCLE, the goal is to stay in s_1 forever.

As illustrated above, $c(s,a) = \mathbb{1}_{s_2}$ is the cost function for this MDP. Let the expert perfectly optimize this function by always taking a_1 in s_1. Thus, we are in the $O(T)$-recoverable setting. Then, for any $\epsilon > 0$, if the learner takes a_2 in s_1 with probability ϵ, $J(\pi_E) - J(\pi) = \sum_{t=1}^{T} \epsilon(1-\epsilon)^{t-1}(T-t) = \Omega(\epsilon T^2)$. There is only one action in s_2 so it is not possible to have a nonzero classification error in this state. $\square$

A.2. Proof of Entropy Regularization Lemma

Lemma 8. ***Entropy Regularization Lemma:*** *By optimizing $U_j(\pi, f) - \alpha H(\pi)$ to a δ-approximate equilibrium, one recovers at worst a $Q_M\sqrt{\frac{2\delta}{\alpha}} + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$ equilibrium strategy for the policy player on the original game.*

Proof. We optimize in the $P(\mathbf{A}^T||\mathbf{S}^T)$ policy representation where strong duality holds and define the following:

$$\pi^R = \arg\min_{\pi\in\Pi}(\max_{f\in\mathcal{F}} U_j(\pi, f) - \alpha H(\pi))$$

First, we derive a bound on the distance between $\hat{\pi}$ and π^R. We define M as follows:

$$M(\pi) = \max_{f\in\mathcal{F}} U_j(\pi, f) - \alpha H(\pi) + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$$

M is an α-strongly convex function with respect to $||\cdot||_1$ because U is a max of linear functions, $-H$ is 1-strongly convex, and the third term is a constant. This tells us that:

$$M(\pi^R) - M(\hat{\pi}) \leq \nabla M(\pi^R)^T(\pi^R - \hat{\pi}) - \frac{\alpha}{2}||\pi^R - \hat{\pi}||_1^2$$

We note that because π^R minimizes M, the first term on the RHS is negative, allowing us to simplify this expression to:

$$\frac{\alpha}{2}||\pi^R - \hat{\pi}||_1^2 \leq M(\hat{\pi}) - M(\pi^R)$$

We now upper bound the RHS of this expression via the following series of substitutions:

$$\begin{aligned}
M(\hat{\pi}) &= \max_{f\in\mathcal{F}} U_j(\hat{\pi}, f) - \alpha H(\hat{\pi}) + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)\\
&\leq U_j(\hat{\pi}, \hat{f}) - \alpha H(\hat{\pi}) + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|) + \delta\\
&\leq U_j(\pi^R, \hat{f}) - \alpha H(\pi^R) + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|) + \delta\\
&\leq \max_{f\in\mathcal{F}} U_j(\pi^R, f) - \alpha H(\pi^R) + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|) + \delta\\
&= M(\pi^R) + \delta
\end{aligned}$$

Rearranging terms to get the desired bound on strategy distance:

$$\begin{aligned}
&M(\hat{\pi}) - M(\pi^R) \leq \delta\\
&\Rightarrow ||\pi^R - \hat{\pi}||_1^2 \leq \frac{2\delta}{\alpha}\\
&\Rightarrow ||\pi^R - \hat{\pi}||_1 \leq \sqrt{\frac{2\delta}{\alpha}}
\end{aligned}$$

Next, we prove that π^R is a $\alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$-approximate equilibrium strategy for the original, unregularized game. We note that $H(\pi) \in [0, T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)]$ and then proceed as follows:

$$\begin{aligned}
\max_{f\in\mathcal{F}} U_j(\pi^R, f) &= M(\pi^R) + \alpha H(\pi^R) - \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)\\
&\leq M(\pi^R)\\
&\leq \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)
\end{aligned}$$

The last line comes from the fact that playing the optimal strategy in the original game on the regularized game could at worst lead to a payoff of $\alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$. Therefore, the value of the regularized game can at most be this quantity. Recalling that the value of the original game is 0 and rearranging terms, we get:

$$\max_{f\in\mathcal{F}} U_j(\pi^R, f) - \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|) \leq 0 = \max_{f\in\mathcal{F}} \min_{\pi\in\Pi} U_j(\pi, f)$$

Thus by definition, π^R must be half of an $\alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$-approximate equilibrium strategy pair.

Next, let Q_M denote the absolute difference between the minimum and maximum Q-value. For a fixed f, the maximum amount the policy player could gain from switching to policies within an L_1 ball of radius r centered at the original policy is rQ_M by the bilinearity of the game and Hölder's inequality. Because the supremum over k-Lipschitz functions is known to be k-Lipschitz, this implies the same is true for the payoff against the best response f. To complete the proof, we can set $r = \sqrt{\frac{2\delta}{\alpha}}$ and combine this with the fact that π^R achieves in the worst case a payoff of $\alpha T(\ln|\mathcal{A}|+\ln|\mathcal{S}|)$ to prove that $\hat{\pi}$ can at most achieve a payoff of $Q_M\sqrt{\frac{2\delta}{\alpha}} + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$ on the original game, which establishes $\hat{\pi}$ as a $(Q_M\sqrt{\frac{2\delta}{\alpha}} + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|))$-approximate equilibrium solution.

□

A.3. Proof of Theorem 1

We proceed in cases.

Proof. We first consider the **primal case**. Our goal is to compute a policy $\hat{\pi}$ such that:

$$\max_{f\in\mathcal{F}} U_j(\hat{\pi}, f) \leq \delta$$

We prove that such a policy can be found efficiently by executing the following procedure for a polynomially large number of iterations:

1. For $t = 1 \dots N$, do:
   2. No-regret algorithm computes π^t.
   3. Set f^t to the best response to π^t.
4. Return $\hat{\pi} = \pi^{t^*}$, $t^* = \arg\min_t U_j(\pi^t, f^t)$.

Recall that via our no-regret assumption we know that

$$\frac{1}{N}\sum_t^N U_j(\pi^t, f^t) - \frac{1}{N}\min_{\pi\in\Pi}\sum_t^N U_j(\pi, f^t) \leq \frac{\beta_\Pi(N)}{N} \leq \delta$$

for some N that is poly$(\frac{1}{\delta})$. We can rearrange terms and use the fact that $\pi_E \in \Pi$ to upper bound the average payoff:

$$\frac{1}{N}\sum_t^N U_j(\pi^t, f^t) \leq \delta + \frac{1}{N}\min_{\pi\in\Pi}\sum_t^N U_j(\pi, f^t) \leq \delta$$

Using the property that there must be at least one element in an average that is at most the value of the average:

$$\min_t U_j(\pi^t, f^t) \leq \frac{1}{N}\sum_t^N U_j(\pi^t, f^t) \leq \delta$$

To complete the proof, we recall that f^t is chosen as the best response to π^t, giving us that:

$$\min_t \max_{f\in\mathcal{F}} U_j(\pi^t, f) \leq \delta$$

In words, this means that by setting $\hat{\pi}$ to the policy with the lowest loss out of the N computed, we are able to efficiently (within poly$(\frac{1}{\delta})$ iterations) find a δ-approximate equilibrium strategy for the policy player. Note that this result holds without assuming a finite $\mathcal{S}$ and $\mathcal{A}$ and does not require regularization of the policy. However, it requires us to have a no-regret algorithm over Π which can be a challenge for the reward moment-matching game.

We now consider the **dual case**. As before, we wish to find a policy $\hat{\pi}$ such that:

$$\max_{f\in\mathcal{F}} U_j(\hat{\pi}, f) \leq \delta$$

We run the following procedure on $U_j(\pi, f) - \alpha H(\pi)$:

1. For $t = 1 \dots N$, do:
   2. No-regret algorithm computes f^t.
   3. Set π^t to the best response to f^t.
4. Return $\hat{\pi} = \arg\min_{\pi\in\Pi} U_j(\pi, \overline{f}) - \alpha H(\pi)$.

By the classic result of (Freund and Schapire 1997), we know that the average of the N iterates produced by the above procedure (which we denote $\overline{f}$ and $\overline{\pi}$) is a δ'-approximate equilibrium strategy for some N that is poly$(\frac{1}{\delta'})$. Applying our Entropy Regularization Lemma, we can upper bound the payoff of $\overline{\pi}$ on the original game:

$$\sup_{f\in\mathcal{F}} U_j(\overline{\pi}, f) \leq Q_M\sqrt{\frac{2\delta'}{\alpha}} + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$$

We now proceed similarly to our proof of the Entropy Regularization Lemma by first bounding the distance between $\overline{\pi}$ and $\hat{\pi}$ and the appealing to the Q_m-Lipschitzness of U_j. Let $l(\pi) = U_j(\pi, \overline{f}) - \alpha H(\pi)$. Then, while keeping the fact that l is α-strongly convex in mind:

$$\begin{aligned}
&l(\hat{\pi}) - l(\overline{\pi}) \leq \nabla l(\hat{\pi})^T(\hat{\pi} - \overline{\pi}) - \frac{\alpha}{2}||\overline{\pi} - \hat{\pi}||_1^2 \\
&\Rightarrow \frac{\alpha}{2}||\overline{\pi} - \hat{\pi}||_1^2 \leq l(\overline{\pi}) - l(\hat{\pi}) + \nabla l(\hat{\pi})^T(\hat{\pi} - \overline{\pi}) \\
&\Rightarrow \frac{\alpha}{2}||\overline{\pi} - \hat{\pi}||_1^2 \leq l(\overline{\pi}) - l(\hat{\pi}) \\
&\Rightarrow \frac{\alpha}{2}||\overline{\pi} - \hat{\pi}||_1^2 \leq \delta' \\
&\Rightarrow ||\overline{\pi} - \hat{\pi}||_1 \leq \sqrt{\frac{2\delta'}{\alpha}}
\end{aligned}$$

As before, the second to last step follows from the definition of a δ'-approximate equilibrium. Now, by the bilinearity of the game, Hölder's inequality, and the fact that supremum over k-Lipschitz functions is known to be k-Lipschitz, we can state that:

$$\sup_{f\in\mathcal{F}} U_j(\hat{\pi}, f) \leq 2Q_M\sqrt{\frac{2\delta'}{\alpha}} + \alpha T(\ln|\mathcal{A}| + \ln|\mathcal{S}|)$$

To ensure that the LHS of this expression is upper bounded by δ, it is sufficient to set $\alpha = \frac{\delta}{2T(\ln|\mathcal{A}|+\ln|\mathcal{S}|)}$ and $\delta' = \frac{\delta^2\alpha}{32Q_M^2}$. Plugging in these terms, we arrive at:

$$\sup_{f\in\mathcal{F}} U_j(\hat{\pi}, f) \leq \frac{\delta}{2} + \frac{\delta}{2} \leq \delta$$

We note that in practice, α is rather sensitive hyperparameter of maximum entropy reinforcement learning algorithms (Haarnoja et al. 2018) and hope that the above expression might provide some rough guidance for how to set α. To complete the proof, note that N is poly$(\frac{1}{\delta'})$ and $\frac{1}{\delta'} = \frac{64Q_M^2 T(\ln|\mathcal{A}|+\ln|\mathcal{S}|)}{\delta^3}$. Thus, N is poly$(\frac{1}{\delta}, T, \ln|\mathcal{A}|, \ln|\mathcal{S}|)$).

□

B. Algorithm Derivations

B.1. AdVIL Derivation

We begin by performing the following substitution: $f = v - \mathcal{B}^\pi v$, where

$$\mathcal{B}^\pi v = \mathop{\mathbb{E}}_{\substack{s_{t+1}\sim\mathcal{T}(\cdot|s_t,a_t),\\ a_{t+1}\sim\pi(s_{t+1})}}[v]$$

is the expected Bellman operator under the learner's current policy. Our objective (2) then becomes:

$$\begin{aligned}\sup_{v\in\mathcal{F}} \sum_{t=1}^{T} &\mathop{\mathbb{E}}_{\tau\sim\pi}[v(s_t,a_t) - \mathcal{B}^\pi v(s_t,a_t)] \\ &- \mathop{\mathbb{E}}_{\tau\sim\pi_E}[v(s_t,a_t) - \mathcal{B}^\pi v(s_t,a_t)]\end{aligned}$$

This expression telescopes over time, simplifying to:

$$\sup_{v\in\mathcal{F}} \mathop{\mathbb{E}}_{\tau\sim\pi}[v(s_0,a_0)] - \sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi_E}[v(s_t,a_t) - \mathcal{B}^\pi v(s_t,a_t)]$$

We approximate $\mathcal{B}^\pi v$ via a single-sample estimate from the respective expert trajectory, yielding the following off-policy expression:

$$\sup_{v\in\mathcal{F}} \mathop{\mathbb{E}}_{\tau\sim\pi}[v(s_0,a_0)] - \sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi_E}[v(s_t,a_t) - \mathop{\mathbb{E}}_{a\sim\pi(s_{t+1})}[v(s_{t+1},a)]]$$

This resembles the form of the objective in ValueDICE (Kostrikov et al. 2019) but without requiring us to take the expectation of the exponentiated discriminator. We can further simplify this objective by noticing that trajectories generated by π_E and π have the same starting state distribution:

$$\sup_{v\in\mathcal{F}} \mathop{\mathbb{E}}_{\tau\sim\pi_E}[\sum_{t=1}^{T} \mathop{\mathbb{E}}_{a\sim\pi(s_t)}[v(s_t,a)] - v(s_t,a_t)] \tag{4}$$

We also note that this AdVIL objective can be derived straightforwardly via the Performance Difference Lemma.

B.2. AdRIL Derivation

Let $\mathcal{F}$ be a RKHS be equipped with kernel $K : (\mathcal{S}\times\mathcal{A}) \times (\mathcal{S}\times\mathcal{A}) \to \mathbb{R}$. On iteration k of the algorithm, consider a purely cosmetic variation of our IPM-based objective (2):

$$\sup_{c\in\mathcal{F}} \sum_{t=1}^{T} (\mathop{\mathbb{E}}_{\tau\sim\pi_k}[c(s_t,a_t)] - \mathop{\mathbb{E}}_{\tau\sim\pi_E}[c(s_t,a_t)]) = \sup_{c\in\mathcal{F}} L_k(c)$$

We evaluate the first expectation by collecting on-policy rollouts into a dataset $\mathcal{D}_k$ and the second by sampling from a fixed set of expert demonstrations $\mathcal{D}_E$. Assume that $|\mathcal{D}_k|$ is constant across iterations. Let $\mathcal{E}$ be the evaluation functional. Then, taking the functional gradient:

$$\begin{aligned}\nabla_c L_k(c) &= \sum_{t=1}^{T} \frac{1}{|\mathcal{D}_k|} \sum_{\tau}^{\mathcal{D}_k} \nabla_c \mathcal{E}[c;(s_t,a_t)] - \frac{1}{|\mathcal{D}_E|} \sum_{\tau}^{\mathcal{D}_E} \nabla_c \mathcal{E}[c;(s_t,a_t)] \\ &= \sum_{t=1}^{T} \frac{1}{|\mathcal{D}_k|} \sum_{\tau}^{\mathcal{D}_k} K([s_t,a_t],\cdot) - \frac{1}{|\mathcal{D}_E|} \sum_{\tau}^{\mathcal{D}_E} K([s_t,a_t],\cdot)\end{aligned}$$

where K could be an state-action indicator ($\mathbb{1}_{s,a}$) in discrete spaces and relaxed to a Gaussian in continuous spaces. Let $\overline{\mathcal{D}_k} = \bigcup_{i=0}^{k} \mathcal{D}_i$ be the aggregation of all previous $\mathcal{D}_i$. Averaging functional gradients over iterations of the algorithm (which, other than a scale factor that does not affect the optimal policy, is equivalent to having a constant learning rate of 1), we get the cost function our policy tries to minimize:

$$\begin{aligned}C(\pi_k) &= \sum_{i=0}^{k} \nabla_c L_i(c) \\ &= \sum_{t=1}^{T} \frac{1}{|\overline{\mathcal{D}_k}|} \sum_{\tau}^{\overline{\mathcal{D}_k}} K([s_t,a_t],\cdot) - \frac{1}{|\mathcal{D}_E|} \sum_{\tau}^{\mathcal{D}_E} K([s_t,a_t],\cdot)\end{aligned} \tag{5}$$

B.3. DAeQuIL Derivations

Let d_π denote the state-action visitation distribution of π. Then, DAeQuIL can be seen as Follow The Regularized Leader on the following sequence of losses:

1. $f_i = \arg\max_{f\in\mathcal{F}} \mathbb{E}_{s,a\sim d_{\pi_i}}[f(s,a) - f(s,\pi_E(s))]$
2. $l_i(\pi) = \mathbb{E}_{s\sim d_{\pi_i}}[f_i(s,\pi(s)) - f_i(s,\pi_E(s))]$

Solving the on-Q game proper would instead require $l_i'(\pi) = \mathbb{E}_{s\sim d_\pi}[f_i(s,\pi(s)) - f_i(s,\pi_E(s))]$ – for the state distribution to depend on the policy that is passed to the loss. While this would allow our previous no-regret analysis to apply as written, we would need to re-sample trajectories after every gradient step, a burden we'd like to avoid.

Let us consider the no-regret guarantee we get from the DAeQuIL losses:

$$\frac{1}{N}\sum_{t}^{N} l_t(\pi^t) - \frac{1}{N}\min_{\pi\in\Pi}\sum_{t}^{N} l_t(\pi) \leq \frac{\beta_\Pi(N)}{N} \leq \delta$$

Notice that $l_t(\pi^t) = \max_{f \in \mathcal{F}} U_3(\pi^t, f)$, the exact quantity we'd like to bound. The tricky part comes from the second term in the regret – under realizability, ($\pi_E \in \Pi$), this term is 0 and `DAeQuIL` directly finds a δ-approximate equilibrium for the on-Q game. Otherwise, we require the following weak notion of realizability to maintain the on-Q moment matching bounds: $\exists \pi' \in \Pi$ s.t.

$$\max_{d_\pi \in d_\Pi} \max_{f \in \mathcal{F}} \mathbb{E}_{s \sim d_\pi}[f(s, \pi'(a)) - f(s, \pi_E(a))] \leq O(\epsilon)$$

In words, this is saying that there exists a policy π' that can match expert moments up to ϵ on any state visitation distribution generated by a policy in Π. If we instead solved the on-Q game directly by using $l_i'(\pi)$, we would instead need the condition: $\exists \pi' \in \Pi$ s.t.

$$\max_{f \in \mathcal{F}} \mathbb{E}_{s \sim d_{\pi'}}[f(s, \pi'(a)) - f(s, \pi_E(a))] \leq O(\epsilon)$$

This weaker condition is concomitant with a much more computationally expensive optimization procedure.

C. Experimental Setup

C.1. Expert

We use the Stable Baselines 3 (Raffin et al. 2019) implementation of PPO (Schulman et al. 2017) and SAC (Haarnoja et al. 2018) to train experts for each environment, mostly using the already tuned hyperparameters from (Raffin 2020). Specifically, we use the modifications in Tables 4 and 5 to the Stable Baselines Defaults.

PARAMETER	VALUE
BUFFER SIZE	300000
BATCH SIZE	256
γ	0.98
τ	0.02
TRAINING FREQ.	64
GRADIENT STEPS	64
LEARNING RATE	7.3E-4
POLICY ARCHITECTURE	256 X 2
STATE-DEPENDENT EXPLORATION	TRUE
TRAINING TIMESTEPS	1E6

Table 4. Expert hyperparameters for HalfCheetah Bullet Task.

PARAMETER	VALUE
BUFFER SIZE	300000
BATCH SIZE	256
γ	0.98
τ	0.02
TRAINING FREQ.	64
GRADIENT STEPS	64
LEARNING RATE	7.3E-4
POLICY ARCHITECTURE	256 X 2
STATE-DEPENDENT EXPLORATION	TRUE
TRAINING TIMESTEPS	1E6

Table 5. Expert hyperparameters for Ant Bullet Task.

C.2. Baselines

For all learning algorithms, we perform 5 runs and use a common architecture of 256 x 2 with ReLU activations. For each datapoint, we average the cumulative reward of 10 trajectories. For offline algorithms, we train on $\{5, 10, 15, 20, 25\}$ expert trajectories with a maximum of 500k iterations of the optimization procedure. For online algorithms, we train on a fixed number of trajectories (5 for HalfCheetah and 20 for Ant) for 500k environment steps. For GAIL (Ho and Ermon 2016) and behavioral cloning (Pomerleau 1989), we use the implementation produced by (Wang et al. 2020). We use the changes from the default values in Tables 6 and 7 for all tasks.

ENV.	EXPERT	BC PERFORMANCE
HALFCHEETAH	2154	2083
ANT	2585	2526

Table 6. With enough data (25 trajectories) and 100k steps of gradient descent, behavioral cloning is able to solve all tasks considered, replicating the results of (Spencer et al. 2021). However, other approaches are able to perform better when there is less data available.

PARAMETER	VALUE
ENTROPY WEIGHT	0
L2 WEIGHT	0
TRAINING TIMESTEPS	5E5

Table 7. Learner hyperparameters for Behavioral Cloning.

For SQIL (Reddy et al. 2019), we build a custom implementation on top of Stable Baselines with feedback from the authors. As seen in Table 9, we use the similar parameters for SAC as we did for training the expert.

We modify the open-sourced code for ValueDICE (Kostrikov et al. 2019) to be actually off-policy with feedback from the authors. The publicly available version of the ValueDICE code uses on-policy samples to compute a regularization term, even when it is turned off in the flags. We release our version.[8] We use the default hyperparameters for all experiments (and thus, train for 500k steps).

PARAMETER	VALUE
NUM STEPS	1024
EXPERT BATCH SIZE	32

Table 8. Learner hyperparameters for GAIL.

PARAMETER	VALUE
γ	0.98
τ	0.02
TRAINING FREQ.	64
GRADIENT STEPS	64
LEARNING RATE	LINEAR SCHEDULE OF 7.3E-4

Table 9. Learner hyperparameters for SQIL.

C.3. Our Algorithms

In this section, we use **bold text** to highlight sensitive hyperparameters. Similarly to SQIL, `AdRIL` is built on top of the Stable Baselines implementation of SAC. `AdVIL` is written in pure PyTorch. We use the same network architecture choices as for the baselines. For `AdRIL` we use the hyperparameters in Table 10 across all experiments.

PARAMETER	VALUE
γ	0.98
τ	0.02
TRAINING FREQ.	64
GRADIENT STEPS	64
LEARNING RATE	LINEAR SCHEDULE OF 7.3E-4
f **UPDATE FREQ.**	1250

Table 10. Learner hyperparameters for `AdRIL`.

We note that `AdRIL` requires careful tuning of f **Update Freq.** for strong performance. To find the value specified, we ran trials with $\{1250, 2500, 5000, 12500, 25000, 50000\}$ and selected the one that achieved the most stable updates. In practice, we would recommend evaluating a trained policy on a validation set to set this parameter. We also note because SAC is an off-policy algorithm, we are free to initialize the learner by adding all expert samples to the replay buffer at the start, as is done for SQIL.

We change one parameter between environments for `AdRIL` – for HalfCheetah, we perform standard sampling from the replay buffer while for Ant we sample an expert trajectory with $p = \frac{1}{2}$ and a learner trajectory otherwise, similar to SQIL. We find that for certain environments, this modification can somewhat increase the stability of updates while for other environments it can significantly hamper learner performance. We recommend trying both options if possible but defaulting to standard sampling.

[8] `https://github.com/gkswamy98/valuedice`

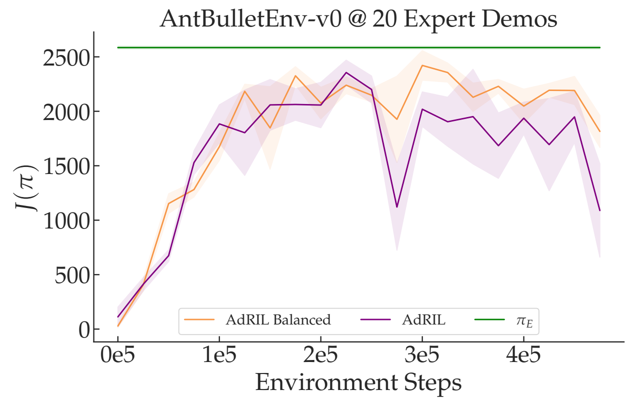

Figure 7. A comparison of balanced vs. unbalanced sampling for `AdRIL` on the Ant Environment. For certain tasks, balanced sampling can help with the stability of updates.

For `AdVIL`, we use the hyperparameters in Table 11 across all tasks. Emperically, small learning rates, large batch sizes, and regularization of both players are critical to stable convergence. We find that `AdVIL` converges significantly more quickly than ValueDICE, requiring only **50k steps for HalfCheetah and 100k Steps for Ant** instead of 500k steps for both tasks. However, we also find that running `AdVIL` for longer than these prescribed amounts can lead to a collapse of policy performance. Fortunately, this can easily be caught by watching for sudden and large fluctuations in policy loss after a long period of steady decreases. One can perform this early-stopping check without access to the environment.

PARAMETER	VALUE
η_π	8E-6
η_f	8E-4
BATCH SIZE	1024
f GRADIENT TARGET	0.4
f GRADIENT PENALTY WEIGHT	10
π ORTHOGONAL REGULARIZATION	1E-4
π MSE REGULARIZATION WEIGHT	0.2
NORMALIZE STATES WITH EXPERT DATA	TRUE
NORMALIZE ACTIONS TO [-1, 1]	TRUE
GRADIENT NORM CLIPPING	[-40, 40]

Table 11. Learner hyperparameters for `AdVIL`.

D. On-Q Experiments

We perform two experiments to tease out when one should apply `DAeQuIL` over DAgger. We first present results on a rocket-landing task from OpenAI Gym where behavioral cloning by itself is able to nearly solve the task, as has been previously noted (Spencer et al. 2021). To make the task more challenging, we truncate the last two dimensions of the state for the policy class, which corresponds to masking the location of the legs of the lander. We use two-layer neural networks with 64 hidden units as all our function classes, perform the optimization steps via ADAM with learning rate $3e-4$, and sample 10 trajectories per update. Here, we see `DAeQuIL` do around as well as DAgger (Fig. 8), with both algorithms quickly learning a policy of quality equivalent to that of the expert. We list the full parameters of the algorithms in Tables 12 and 13. As in the previous section, **bold text** highlights sensitive hyperparameters.

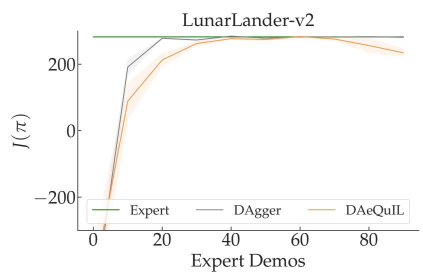

Figure 8. As behavioral cloning alone is able to nearly match the expert, DAgger and `DAeQuIL` perform around the same.

PARAMETER	VALUE
BATCH SIZE	32
GRADIENT STEPS π UPDATE	3E3
GRADIENT STEPS f UPDATE	**1E3**
f GRADIENT PENALTY TARGET	0
f GRADIENT PENALTY WEIGHT	5

Table 12. Learner hyperparameters for `DAeQuIL` on LunarLander-v2.

PARAMETER	VALUE
BATCH SIZE	32
GRADIENT STEPS π UPDATE	1E4

Table 13. Learner hyperparameters for DAgger on LunarLander-v2.

We next perform an experiment to show how careful curation of moments can allow `DAeQuIL` to significantly outperform DAgger at some tasks. Consider an operator trying to teach a drone to fly through a cluttered forest filled with trees. The operator has already trained a perception system that provides state information to the drone about whether a tree is infront of it. Because the operator is primarily concerned with safety, she only cares about making it through the forest, not the lateral location of the drone on the other side. She also tries to demonstrate a wide variety of evasive maneuvers as to hopefully teach the drone to generalize. We simulate such an operator and visualize the trajectories in Fig. 4, left.

Standard behavioral cloning with an ℓ_2 loss would fail at this task because it would attempt to reproduce the conditional mean action, leading the drone to fly straight into the tree. Unfortunately, DAgger inherits this flaw, and is therefore prone to producing a policy that crashes into the first tree it sees, as shown in Fig. 4, center.

For `DAeQuIL`, the operator leverages her knowledge of the problem and passes in two important moments: the perception system's imminent crash indicator and the absolute difference between the current and proposed headings. Whenever the former is on, the latter is a large value under the expert's distribution as they are trying to avoid the tree. So, the learner figures out that it should swerve out of the way of the tree. This leads to policies learned via `DAeQuIL` to be able to progress much further into the forest, as seen in Fig. 4, right.

Using the final position of executed trajectories as the cumulative reward, we see the following learning curves with `DAeQuIL` clearly out-performing DAgger (Fig. 9).

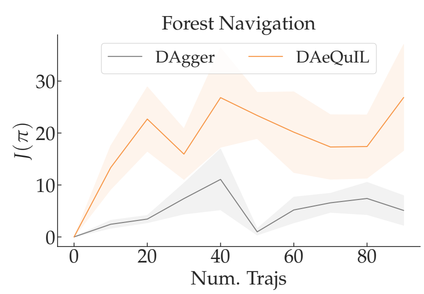

Figure 9. $J(\pi)$ is the longitudinal distance into the forest the learner is able to progress. All experiments are run on the forest layout shown in Fig. 4 and standard errors are computed across 10 trials.

We use the same function classes as the previous experiment but use a hidden size of 32 for the discriminator of `DAeQuIL`. We list the full set of parameters in Tables 14 and 15.

E. Additional Moment Types

E.1. A Fourth Moment Class: Mixed-Moment Value

We could instead plug in Q-moments to the reward moment payoff function U_1. Let $\mathcal{F}_V$ and $\mathcal{F}_{V_E}$ refer to the classes of policy and expert value functions. As before, we assume both of these classes are closed under negation and include the true value and expert value functions. For notational convenience, we assume both classes contain functions with

Parameter	Value
Batch Size	32
Gradient Steps π Update	2e3
ℓ_{BC} Scale	**5e-2**
Gradient Steps f Update	**1e3**
f Gradient Penalty Target	0
f Gradient Penalty Weight	5

Table 14. Learner hyperparameters for `DAeQuIL` on Forest Navigation.

Parameter	Value
Batch Size	32
Gradient Steps π Update	5e3

Table 15. Learner hyperparameters for DAgger on Forest Navigation.

type signatures $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$, with the second argument being ignored. Starting from the PDL, we can expand as follows:

$$\begin{aligned}
&J(\pi_E) - J(\pi) \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi_E}[Q_t^\pi(s_t,a_t) - \mathop{\mathbb{E}}_{a\sim\pi(s_t)}[Q_t^\pi(s_t,a)]] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi_E}[Q_t^\pi(s_t,a_t) - \mathop{\mathbb{E}}_{a\sim\pi(s_t)}[Q_t^\pi(s_t,a)]] \\
&\qquad + \mathop{\mathbb{E}}_{\tau\sim\pi}[Q_t^\pi(s_t,a_t) - Q_t^\pi(s_t,a_t)] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi_E}[Q_t^\pi(s_t,a_t)]] - \mathop{\mathbb{E}}_{\tau\sim\pi}[Q_t^\pi(s_t,a_t)]] \\
&\qquad + \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ a\sim\pi(s_t)}}[Q_t^\pi(s_t,a)] - \mathop{\mathbb{E}}_{\substack{\tau\sim\pi_E \\ a\sim\pi(s_t)}}[Q_t^\pi(s_t,a)] \\
&\leq \sup_{f\in\mathcal{F}_Q\cup\mathcal{F}_V} 2\sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi}[f(s_t,a_t)] - \mathop{\mathbb{E}}_{\tau\sim\pi_E}[f(s_t,a_t)]
\end{aligned}$$

The last step follows from the fact that $\sup_{a\in A} f(a) + \sup_{b\in B} f(b) \leq \sup_{c\in A\cup B} 2f(c)$. An analogous bound for $\mathcal{F}_{Q_E}$ and $\mathcal{F}_{V_E}$ can be proved by expanding the PDL in the reverse direction. We can use these expansions to provide bounds related to the reward-moment bound:

Lemma 9. ***Mixed Moment Value Upper Bound:*** *If $\mathcal{F}_Q/2T$ and $\mathcal{F}_V/2T$ spans $\mathcal{F}$ or $\mathcal{F}_{Q_E}/2T$ and $\mathcal{F}_{V_E}/2T$ do, then for all MDPs, π_E, and $\pi \leftarrow \Psi\{\epsilon\}(U_1)$, $J(\pi_E) - J(\pi) \leq O(\epsilon T^2)$.*

Proof. We start by expanding the imitation gap:

$$\begin{aligned}
&J(\pi_E) - J(\pi) \\
&\leq \sup_{f\in\mathcal{F}_Q\cup\mathcal{F}_V} 2\sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi}[f(s_t,a_t)] - \mathop{\mathbb{E}}_{\tau\sim\pi_E}[f(s_t,a_t)] \\
&\leq \sup_{f\in\mathcal{F}} 2 \mathop{\mathbb{E}}_{\tau\sim\pi} \sum_{t=1}^{T} 2Tf(s_t,a_t) - \mathop{\mathbb{E}}_{\tau\sim\pi_E} \sum_{t=1}^{T} 2Tf(s_t,a_t) \\
&= 4T^2 \sup_{f\in\mathcal{F}} U_1(\pi,f) \leq 4\epsilon T^2
\end{aligned}$$

The T in the second to last line comes from the scaling down of either the $(\mathcal{F}_Q, \mathcal{F}_V)$ or the $(\mathcal{F}_{Q_E}, \mathcal{F}_{V_E})$ pairs by T to fit into the function class $\mathcal{F}$. □

Lemma 10. ***Mixed Moment Value Lower Bound:*** *There exists an MDP, π_E, and $\pi \leftarrow \Psi\{\epsilon\}(U_1)$ such that $J(\pi_E) - J(\pi) \geq \Omega(\epsilon T)$.*

Proof. The proof of the *reward lower bound* holds verbatim. □

These bounds show that solving this game, which might be more challenging than the reward-moment game, appears to offer no policy performance gains. However, in the imitation learning from observation alone setting, where one does not have access to action labels, reward-matching might be impossible, forcing one to use an approach similar to the above. This is because value functions are pure functions of state, not actions. (Sun et al. 2019) give an efficient algorithm for this setting.

E.2. Combining Reward and Value Moments

For both the *off-Q* and *on-Q* setups, one can leverage the standard expansion of a Q-function into a sum of rewards to derive a flexible family of algorithms that allow one to include knowledge of both reward and Q moments. Explicitly, for the off-Q case:

$$\begin{aligned}
&J(\pi_E) - J(\pi) \\
&= \frac{1}{T}\Big(\mathop{\mathbb{E}}_{\substack{\tau\sim\pi_E \\ a\sim\pi(s_t)}} [\sum_{t=1}^{T} Q^\pi(s_t,a) - Q^\pi(s_t,a_t)]\Big) \\
&= \frac{1}{T}\Big(\mathop{\mathbb{E}}_{\substack{\tau\sim\pi_E \\ a\sim\pi(s_{t'})}} [\sum_{t=1}^{T}\sum_{t'=1}^{T'} r(s_{t'},a) - r(s_{t'},a_{t'}) \\
&+ Q^\pi_{T'}(s_{T'},a) - Q^\pi_{T'}(s_{T'},a_{T'})]\Big) \\
&\leq \max_{\substack{f\in\mathcal{F}_r \\ g\in\mathcal{F}_Q}} \frac{1}{T}\Big(\mathop{\mathbb{E}}_{\substack{\tau\sim\pi_E \\ a\sim\pi(s_t)}} [\sum_{t=1}^{T}\sum_{t'=t}^{T'} f(s_{t'},a) - f(s_{t'},a_{t'}) \\
&+ g(s_{T'},a) - g(s_{T'},a_{T'})]\Big)
\end{aligned} \tag{6}$$

Passing such a payoff to our oracle with $\mathcal{F}$ spanned by $\mathcal{F}_r/2 \times \mathcal{F}_Q/2T$ would recover the off-Q bounds.

This expansion begs the question of when it is useful. One answer is a standard bias/variance trade-off with different values of T', as has been explored in TD-Gammon (Tesauro 1995). We can provide an alternative answer by considering the limiting case – when the Q function is decomposed entirely into reward functions, the learner is required at timestep t to match the sum of future reward moments. An efficient algorithm for such a problem can be derived as a natural extension of Policy Search by Dynamic Programming (PSDP) (Bagnell et al. 2003), where, starting from $t = T - 1$, the learner matches expert moments one timestep in the future, before moving one step backwards in time along the expert's trajectory. While this approach has the same performance characteristics as off-Q algorithms, matching the class of reward moments might be simpler for some types of problems, like those with sparse rewards. However, it has the added complexity of producing a non-stationary policy.

We can perform an analogous expansion for the on-Q case by utilizing the reverse direction of the PDL:

$$\begin{aligned}
&J(\pi_E) - J(\pi) \\
&= \frac{1}{T}\Big(\mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ a\sim\pi_E(s_t)}} [\sum_{t=1}^{T} Q^{\pi_E}(s_t, a_t) - Q^{\pi_E}(s_t, a)]\Big) \\
&= \frac{1}{T}\Big(\mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ a\sim\pi_E(s_{t'})}} [\sum_{t=1}^{T} \sum_{t'=1}^{T'} r(s_{t'}, a_{t'}) - r(s_{t'}, a) \\
&+ Q^{\pi_E}_{T'}(s_{T'}, a_{T'}) - Q^{\pi_E}_{T'}(s_{T'}, a)]\Big) \\
&\leq \min_{\pi\in\Pi} \max_{\substack{f\in\mathcal{F}_r \\ g\in\mathcal{F}_{Q_E}}} \frac{1}{T}\Big(\mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ a\sim\pi_E(s_{t'})}} [\sum_{t=1}^{T} \sum_{t'=t}^{T'} f(s_{t'}, a_{t'}) - f(s_{t'}, a) \\
&+ g(s_{T'}, a_{T'}) - g(s_{T'}, a)]\Big) \qquad (7)
\end{aligned}$$

Passing such a payoff to our oracle with $\mathcal{F}$ spanned by $\mathcal{F}_r/2 \times \mathcal{F}_{Q_E}/2T$ would recover the on-Q bounds. A backwards-in-time dynamic-programming procedure is not possible for this expansion because of the need to sample trajectories from the policy at previous timesteps.