Offline Learning from Demonstrations and Unlabeled Experience

Konrad Żołna[1] **Alexander Novikov**[1] **Ksenia Konyushkova**[1] **Caglar Gulcehre**[1]

Ziyu Wang[2] **Yusuf Aytar**[1] **Misha Denil**[1] **Nando de Freitas**[1] **Scott Reed**[1]

[1]DeepMind [2]Google Brain

kondiz@google.com

Abstract

Behavior cloning (BC) is often practical for robot learning because it allows a policy to be trained offline without rewards, by supervised learning on expert demonstrations. However, BC does not effectively leverage what we will refer to as *unlabeled experience*: data of mixed and unknown quality without reward annotations. This unlabeled data can be generated by a variety of sources such as human teleoperation, scripted policies and other agents on the same robot. Towards data-driven offline robot learning that can use this unlabeled experience, we introduce Offline Reinforced Imitation Learning (ORIL). ORIL first learns a reward function by contrasting observations from demonstrator and unlabeled trajectories, then annotates all data with the learned reward, and finally trains an agent via offline reinforcement learning. Across a diverse set of continuous control and simulated robotic manipulation tasks, we show that ORIL consistently outperforms comparable BC agents by effectively leveraging unlabeled experience.

1 Introduction

Robotic manipulation poses several obstacles to contemporary deep learning approaches. First, robotic interaction data may be costly and time-consuming to obtain, making online learning from scratch impractical. Second, task rewards may be unobtainable or highly sparse, making conventional reinforcement learning a challenge. Therefore, our objective is to develop an agent that can be trained offline from previously stored robot experience, as advocated by data-driven robotics approaches [5, 11, 13], without requiring reward annotations.

Behavior cloning (BC) agents can learn without task rewards by supervised learning on demonstration set. However, as a result, BC does not effectively use non-expert data, since it may be sub-optimal or from a different task. Offline RL could use all of the data, but only if rewards were provided.

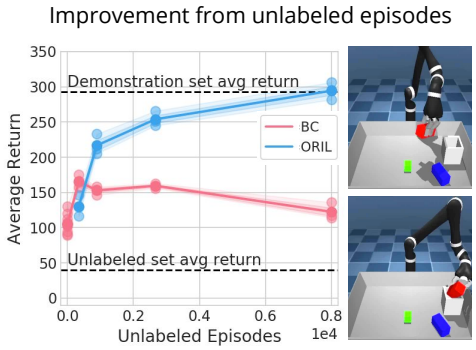

Figure 1: **Performance of ORIL vs BC.** Using the same 189 demonstrations, varying the number of unlabeled episodes for an Insertion task with Jaco arm. Our method leverages unlabeled episodes and eventually achieves expert level.

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2020.

Our proposed Offline Reinforced Imitation Learning (ORIL) agents combine the advantages of both, by the following: (1) learn a reward model by contrasting expert and unlabeled set observations, (2) annotate all stored data using the learned model, and (3) perform offline RL to learn a policy. Hence, unlabeled data is used to train both the reward model and the policy and value function. We observe that the performance improves as more unlabeled data is provided.

We find that our proposed approach benefits from incorporating components from several recently developed methods for adversarial imitation [40], positive-unlabeled reward modeling [38] and a state-of-the-art approach to offline reinforcement learning [37].

As we show experimentally, ORIL has the following desirable characteristics.

- Scalability – performance improves as unlabeled data is added.
- Robustness – works well even when the average quality of unlabeled data is low.
- Sample-efficiency – outperforms BC with a much smaller demonstration set.

In summary, ORIL is an effective method for sample-efficient offline policy learning from demonstrations, from pixels and *without* task rewards, that effectively leverages unlabeled experience.

2 Method

At a high level, our method consists of the following steps: (1) learn a reward function by contrasting expert and unlabeled observations, (2) annotate both demonstrations and unlabeled trajectories with the learned reward, and then (3) perform offline RL.

Let $\mathcal{D}_E$ be the set of expert demonstrations and $\mathcal{D}_U$ be the set of unlabeled trajectories. We assume that $\mathcal{D}_U$ can contain a mix of sub-optimal episodes for the targeted task, or behavior of another task, and also potentially some successful episodes, although proportionally fewer than in $\mathcal{D}_E$. Let $\tau := (s_1, a_1, ..., s_T, a_T) \in \mathcal{D}_E \cup \mathcal{D}_U$ be an episode of length T with observations s_t and actions a_t.[1] We assume that observations could be pixels or low-dimensional states such as joint angles.

2.1 Learning a reward function

We learn a reward model by contrasting expert and unlabeled trajectory states, similarly to how the discriminator is trained in Generative Adversarial Imitation Learning (GAIL) [14]. However, we learn the reward function from a fixed dataset, unlike GAIL which requires access to the environment to generate new trajectories. Consequently, our method is especially suitable for robotic settings, where executing policies during learning can be too costly or slow.

The simplest approach, which we refer to as *flat rewards*, is to label all states of expert demonstrations with a reward of 1, and all states of all unlabeled robot experience with a reward of 0. Although this strategy is naive, it has been shown to be successful in online settings [30].

We can go beyond this hard-labeling approach by training a reward function $R_\psi(s_t)$ with parameters ψ using the following cross-entropy loss:

$$\mathcal{L}_\psi(\mathcal{D}_E, \mathcal{D}_U) = \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log R_\psi(s_t)] + \mathbb{E}_{s'_t \sim \mathcal{D}_U}[-\log(1 - R_\psi(s'_t)], \tag{1}$$

where s_t and s'_t are sampled uniformly from all states in episodes in $\mathcal{D}_E$ and $\mathcal{D}_U$, respectively.

Note that the loss $\mathcal{L}_\psi$ is minimized when $R_\psi(s_t)$ assigns 1 to all expert observations and 0 to all unlabeled states, as in flat rewards. This limiting behavior is unsatisfactory because unlabeled trajectories can contain both successful or unsuccessful states, without us having access to this information. This existence of *false negatives* can hinder online adversarial imitation learning [42]. To overcome this problem, we adapt two approaches for addressing false negatives, which have previously proved to be helpful in online adversarial learning: positive-unlabeled (PU) learning [8, 38] and Task-Relevant Adversarial Imitation Learning (TRAIL) [40].

Positive-unlabeled (PU) Learning addresses the problem of false negatives directly [8]. The main idea is to obtain an estimate of model loss on negative examples, which are not directly available, by re-weighting losses that can be computed using only positive and unlabeled data. In our context,

[1] We assume Markov Decision Process (MDP) and follow the usual RL notation of Sutton and Barto [33].

negative examples would correspond to observations not indicating a success condition for the task, while positive examples would indicate a success condition (e.g. stacking blocks in the correct order). The PU Learning objective is described in the supplementary material; see [8, 7, 38] for a derivation.

TRAIL is an adversarial imitation learning method similar to GAIL, but in TRAIL the discriminator is constrained so as to *not* form spurious associations between visual features and expert labels. For example, the discriminator should not be able to distinguish whether early observations at the very start of an episode come from the demonstration or unlabeled set, since no meaningful behavior has yet been performed. TRAIL has been shown to improve performance on robotic manipulation tasks and also address the problem of false negatives [40].

We adapt TRAIL to the offline setting. We use early observations[2] to form TRAIL constraint sets (as proposed in the original paper). Further details are included in the supplementary material. We refer to the original paper for details [40].

2.2 Reward model regularization

Data augmentation We would like our reward model to generalize even when only a handful of episodes constitute $\mathcal{D}_E$. However, given a set $\mathcal{D}_E$ that is very limited in size, a reward model can achieve perfect score by simply memorizing all these expert states and blindly assigning reward 0 to all the other states. We applied extensive data augmentation to all reward model inputs during training (as in [40]), to alleviate this problem.

Training set split In the offline setting, we only have access to logged data and cannot interact with the environment. Learning the reward model on the entire dataset, and subsequently using the model to annotate the same dataset, is prone to overfitting. To overcome this, we split the unlabeled dataset in half and use only one half to train the reward model. We learn the policy (including critic) on the full dataset. We experimented with a few alternatives, but found this simple strategy to be sufficient for preventing overconfident reward estimates.

2.3 Training an offline RL agent

To train our offline RL agent, we use Critic-Regularized Regression (CRR) [37], a state-of-the-art offline reinforcement learning method. The method assumes access to rewards and we alleviate that by using rewards issued by a reward model (trained as explained in Subsections 2.1 and 2.2).

CRR is advantageous because its policy update is simply a loss-weighted version of behavior cloning, where the per-example loss weights are determined by the critic network. The critic, or state-action value function, is trained via Q-learning on the reward-annotated data. In the most basic form of CRR, the policy loss is

$$\mathcal{L}_\phi = \mathbb{E}_{s_t, a_t \sim \mathcal{D}_E \cup \mathcal{D}_U} \big[-\log P(a_t|\pi_\phi(s_t)) \mathbb{I}\left[Q_\theta(s_t, a_t) > Q_\theta(s_t, \pi_\phi(s_t))\right] \big], \tag{2}$$

where π_ϕ and Q_θ are the trained policy and critic parameterized by ϕ and θ, respectively.

Note that the policy update in Equation 2 is identical to that of behavior cloning, but scaled by an indicator. The indicator function $\mathbb{I}(\cdot)$ is equal to 1 if the stored action a_t is judged to be better than the sampled policy action $a'_t \sim \pi_\phi(s_t)$ according to the current state action value function Q_θ, and is equal to 0 otherwise. As in [37, 4], we train the critic together with the policy using distributional Q-learning, using the learned reward function explained in Subsections 2.1 and 2.2.

3 Experimental setup

3.1 Environments

We conduct experiments on two simulated continuous control domains shown in Figure 2: Robotic Manipulation and the DeepMind Control Suite.

Robotic Manipulation The first domain consists of four block manipulation tasks using a simulated Kinova Jaco arm in a 20cm by 20cm basket. We use 64×64 pixel observations, and episodes are 400 steps. The environment reward is sparse (1 if task is solved and 0 otherwise).

[2] We define early observations as the first 10 observations in each episode.

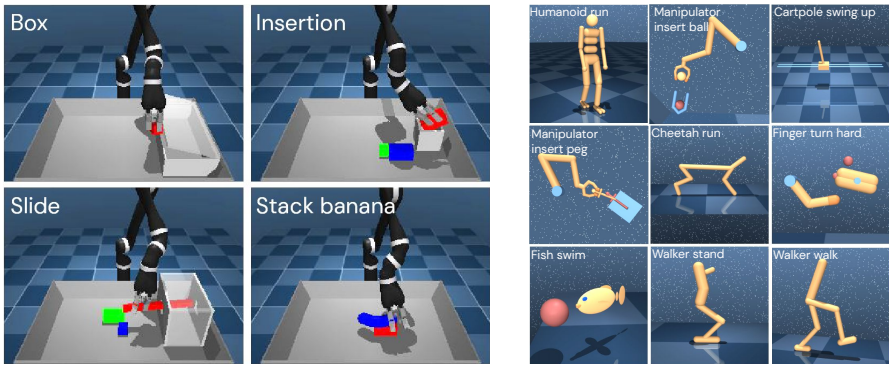

Figure 2: **Robotic Manipulation** (left) is a set of block manipulation tasks with a simulated Kinova Jaco arm in a 20x20cm basket. **DeepMind Control Suite** (right) is a set of popular continuous control environments with tasks of varying difficulty, including locomotion and simple object manipulation.

DeepMind Control Suite The second domain is the DeepMind Control Suite, which contains a variety of continuous control tasks involving locomotion and simple manipulation. Observations consist of joint angles and velocities, and action spaces vary depending on the task. The episodes are 1000 steps and the the environment reward is continuous, maximum 1 per step.

3.2 Datasets

In practice, the set $\mathcal{D}_E$ would be collected by an expert and $\mathcal{D}_U$ is meant to be built based on the logged interactions of the agent from the past.

In this paper, we build these datasets based on existing offline RL datasets. Given a set of logged episodes, we extract a small subset of well performing episodes and treat them as demonstrations $\mathcal{D}_E$. The rest constitute the unlabeled set $\mathcal{D}_U$. We use ground truth rewards only to perform this split and discard the rewards afterwards. Table 1 lists all tasks used in this paper and shows the total numbers of episodes and the sizes of expert sets for each task. We explain the process for each domain below.

Table 1: **Datasets statistics.** The total number of episodes (total) and corresponding number of demonstrations ($\#\mathcal{D}_E$) per task.

Task	Total	$\#\mathcal{D}_E$
Box	8000	161
Insertion	8000	189
Slide	8000	189
Stack banana	8000	298
Cartpole swingup	40	2
Cheetah run	300	3
Finger turn hard	500	9
Fish swim	200	1
Humanoid run	3000	53
Manipulator insert ball	1500	30
Manipulator insert peg	1500	23
Walker stand	200	4
Walker walk	200	6

Robotic Manipulation The datasets for this environment are prepared as in [37]. There are 8000 episodes for each of four tasks and approximately 25% of them are successful (i.e. at least half of the episode steps issue reward 1). Each of these successful episodes was randomly chosen to be in $\mathcal{D}_E$ set with $\frac{1}{16}$ chance.

DeepMind Control Suite We use the open source RL Unplugged datasets [13]. We define an episode as positive if its episodic reward is among top 20% episodes for the task. Each of these positives was randomly chosen to constitute $\mathcal{D}_E$ set with $\frac{1}{10}$ chance. We note that resulting $\mathcal{D}_E$ sets are very small – usually less than 10 episodes (see Table 1).

3.3 Baseline and ablated models

We implement two variants of behavioral cloning:

BC$_{pos}$ Behavioral cloning on positive (expert) data only. This baseline is trained only on $\mathcal{D}_E$ and hence, it is not exposed to low performing episodes in $\mathcal{D}_U$. On the other hand, it may not be able to generalize due to the very limited size of its training set ($\mathcal{D}_E$).
BC$_{all}$ Behavioral cloning on all data. BC$_{all}$ may generalize better than BC$_{pos}$, due to access to a much larger dataset, but its performance may be worse if the quality of data in $\mathcal{D}_U$ is low.

Our method, ORIL, is composed of a batch RL agent (CRR [37]) with a learned reward model. To understand the contribution of each of its components we implement the following variants.

FR This method, similarly to SQIL [30], assumes flat rewards. It does not train a reward model and hence leverages only the CRR component of our method.
ORIL$_R$ This method uses a reward model with generic regularization (data augmentation and splitting) as described in Subsection 2.2. It uses neither PU learning nor TRAIL.
ORIL$_P$ and **ORIL$_T$** Improved ORIL$_R$ which additionally uses PU learning or TRAIL, respectively.

In following sections, when we use **ORIL** without subscript we mean ORIL$_P$ by default, since this variant performed best overall (see Subsection 4).

We also experimented with **off-policy GAIL** [16] applied directly to our offline setting (with D4PG actor-critic [4]). However, the final returns were close to zero with a learnt reward function (suggesting that obtaining policy gradients via offline algorithm is crucial) and therefore we do not report detailed results. Finally, we trained **CRR** with ground-truth rewards to obtain performance upper-bounds.

3.4 Hyperparameters

We used the same hyperparameters (including network architectures) as in CRR [37]. For the reward model, we used exactly the same network architecture as for the critic, but we replaced the final layer with a simple MLP predicting one value followed by sigmoid to scale its predictions.

We train all our models for $1e6$ learner steps and always with 3 seeds. We compute the mean return between $5e5$ and $1e6$ learner steps for each seed. This yields three numbers (one per seed) for every agent and we report mean and standard deviation across these numbers.

4 Experimental results

4.1 Robotic Manipulation

The learning curves for ORIL and both BC variants are shown in Figure 3.

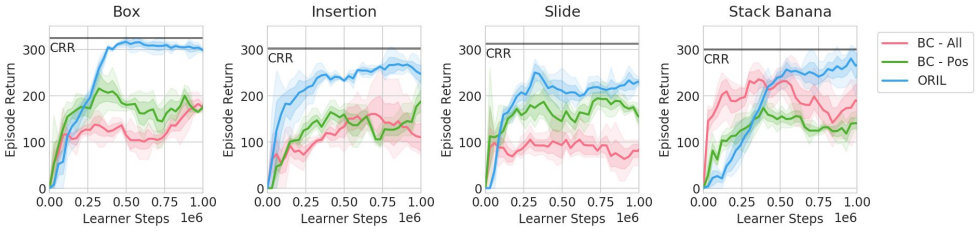

Figure 3: **Robotic Manipulation results.** We compare BC$_{all}$ (BC trained on all data), BC$_{pos}$ (BC trained only on demonstration data) and ORIL. ORIL improves over baselines by leveraging the unlabeled experience.

ORIL closely approaches CRR upper-bound (trained with ground-truth rewards) and outperforms the baselines on all four tasks, showing that ORIL is well suited to make effective use of the unlabeled, mixed quality, trajectories. There is no clear winner between BC$_{pos}$ and BC$_{all}$ which suggests that the quality of the unlabeled data for the considered tasks varies. The results including ablated variants of ORIL are presented in Table 2.

Table 2: **ORIL ablations on Robotic Manipulation domain.** We ablate and compare different variations of BC (BC$_{all}$ and BC$_{pos}$) and ORIL. For ORIL we compare, ORIL (the original method), ORIL$_T$ (using TRAIL instead of PU-learning) and ORIL$_R$ (with general purpose only regularization – no PU learning nor TRAIL). Finally, we have FR which uses flat rewards (and hence does not train a reward model). We also provide per task performance upper-bounds obtained with CRR using ground-truth rewards (in italic).

Task	**BC$_{all}$**	**BC$_{pos}$**	**ORIL**	**ORIL$_T$**	**ORIL$_R$**	**FR**	***CRR***
Box	158 $\pm$ 5	180 $\pm$ 7	**305** $\pm$ 3	**295** $\pm$ 8	**303** $\pm$ 5	**302** $\pm$ 6	*325 $\pm$ 4*
Insertion	146 $\pm$ 8	139 $\pm$ 5	**260** $\pm$ 3	**266** $\pm$ 3	241 $\pm$ 4	100 $\pm$ 4	*302 $\pm$ 12*
Slide	103 $\pm$ 2	181 $\pm$ 5	**214** $\pm$ 13	**224** $\pm$ 19	173 $\pm$ 13	58 $\pm$ 10	*312 $\pm$ 9*
Stack Banana	210 $\pm$ 12	129 $\pm$ 7	**257** $\pm$ 7	178 $\pm$ 25	232 $\pm$ 8	208 $\pm$ 5	*300 $\pm$ 3*

ORIL outperforms the other methods in all four tasks, showing 20-100% higher returns than BC-based methods. Our method works well with both PU learning (ORIL) and TRAIL (ORIL$_T$). When only general purpose regularization is used (ORIL$_R$) the performance drops on all tasks but one.

FR preforms significantly worse than ORIL$_R$. FR lags even behind the BC baselines on two tasks. It primarily indicates that learning a reward model is helpful. It also suggests that the reward model should be regularized, as FR effectively emulates a fully overfitted reward model which minimizes the loss $\mathcal{L}_\psi(\mathcal{D}_E, \mathcal{D}_U)$ in Equation 1.

4.2 DeepMind Control Suite

We present results comparing our method and BC baselines in Figure 4.

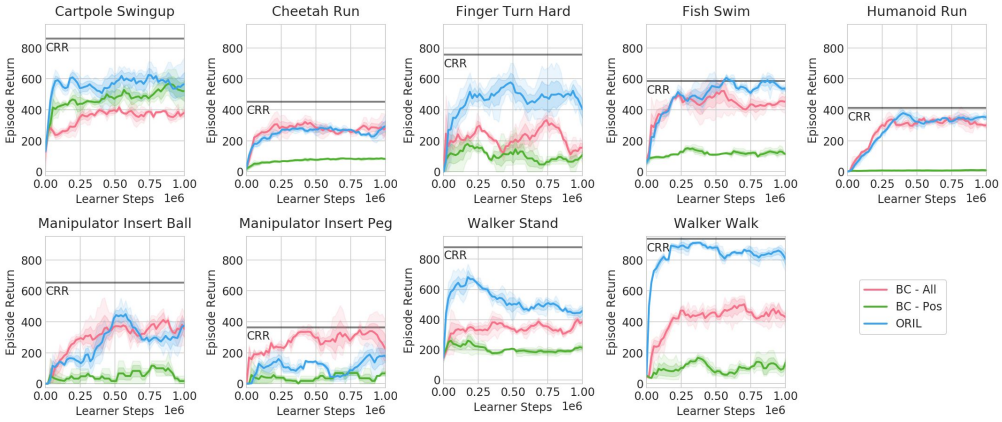

Figure 4: **DeepMind Control Suite results.** Our method (ORIL) is the best on 8 tasks (although BC$_{all}$ is equally good on 3 of them). BC$_{pos}$ is usually the worst due to the very limited set of demonstrations.

Our method outperforms the baselines on 5 out of 9 tasks. For the other three tasks, the performance is comparable with BC$_{all}$, and ORIL is behind for only one task (Manipulator Insert Peg). ORIL is able to reach (or closely approach) CRR upper-bounds even though our method does not need ground-truth rewards (see, e.g. Finger Turn Hard, Walker Walk).

BC$_{pos}$ performs poorly compared to other methods due to very small sizes of $\mathcal{D}_E$ for this domain. It usually scores below 200. Supervised learning on all data (BC$_{all}$) leads to significantly better results, suggesting that the data in $\mathcal{D}_U$ is of relatively high quality. We also found that the performance of BC (but not of ORIL) is reduced when low-quality data is added to $\mathcal{D}_U$ (see Section 5.2).

5 Data set ablations

5.1 Varying the amount of unlabeled data

In this experiment we vary the amount of unlabeled experience available for agent training. Figure 5 shows that adding more unlabeled data improves the performance of ORIL, whereas BC does not benefit from the extra data as consistently, and sometimes performs worse.

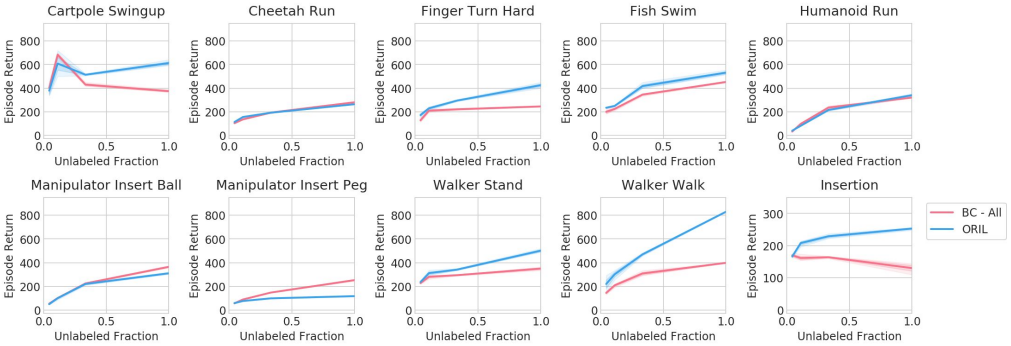

Figure 5: **Ablating the number of unlabeled trajectories.** We investigate the effect of unlabeled trajectories on agent performance. ORIL's performance clearly improves as the number of unlabeled data increases, whereas BC$_{all}$ either does not benefit from the extra data as much, or performs worse.

5.2 Adding low performance data to the unlabeled set

The average data quality for DeepMind Control Suite tasks is relatively high, as evidenced by the fact that BC_{all} performs better than BC_{pos}. We conducted a set of experiments to check if BC_{all} and our method are robust when the ratio of low performing episodes in $\mathcal{D}_U$ increases. We doubled the sizes of the original datasets for four tasks by adding episodes generated only by low performing policies. We kept $\mathcal{D}_E$ as before, and hence all the extra episodes landed in $\mathcal{D}_U$.

We consider four tasks: Finger Turn Hard and Walker Walk (where our method is clearly better than the baselines on the original set), and Fish Swim and Humanoid Run (where the difference is smaller). We report average evaluation scores for the original and amended datasets in Table 3.

Table 3: **Robustness to the low-quality data.** ORIL is more robust than BC to adding low-quality data that has episodes with low average rewards (indicated by "Amended").

Environment	Dataset	Scores	
		ORIL	BC_{all}
Finger Turn Hard	Original	**406** $\pm$ 11.12	250 $\pm$ 7.31
	Amended	**446** $\pm$ 17.79 (+10%)	200 $\pm$ 1.18 (-20%)
Fish Swim	Original	**518** $\pm$ 10.43	458 $\pm$ 5.15
	Amended	**448** $\pm$ 3.76 (-13%)	349 $\pm$ 4.07 (-23%)
Humanoid Run	Original	**349** $\pm$ 2.51	296 $\pm$ 25.91
	Amended	**254** $\pm$ 5.48 (-27%)	217 $\pm$ 3.74 (-26%)
Walker Walk	Original	**829** $\pm$ 15.43	396 $\pm$ 5.70
	Amended	**836** $\pm$ 5.00 (+1%)	248 $\pm$ 8.16 (-37%)

As expected, BC_{all} always performs worse when exposed to low-quality episodes. The largest performance drop is for Walker Walk which is 37%.

ORIL is shown to be significantly more robust. For two tasks, there is no drop in performance at all, and there is even a small improvement for Finger Turn Hard. For the other two tasks the additional low-quality data negatively influence ORIL performance. However, ORIL still outperforms the baseline significantly. It confirms that CRR (the offline RL agent) using learned rewards is able to filter out transitions which would reduce performance if trained on (see Equation 2).

5.3 Varying the number of demonstration episodes

As mentioned in Subsection 3.2, only 10% of positive episodes are used to constitute demonstration sets for DeepMind Control Suite tasks. As presented in our main experiments, ORIL is always significantly better than BC_{pos} in this challenging setting (see Section 4) which indicates the much better sample complexity of our method.

Here, we analyze if BC_{pos} can reach ORIL performance when more (25%) or even all positives are used to build $\mathcal{D}_E$. Figure 6 shows the effect of increasing the proportion of demonstrations.

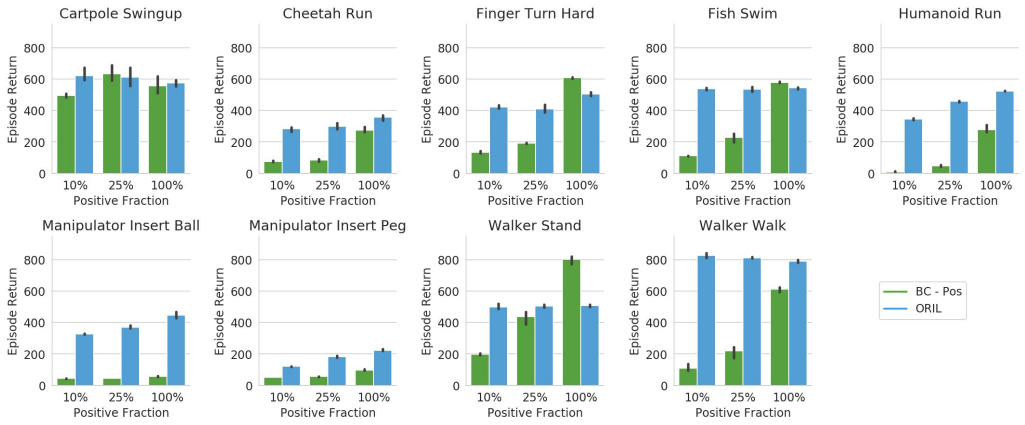

Figure 6: **Control suite results.** ORIL is more robust to decreasing the amount of demonstrations than BC_{pos}.

BC_{pos} achieves better performance when more positives are used, but still performs worse than ORIL. It indicates that ORIL is able to leverage signal present even in low-quality episodes.

ORIL maintains its performance for all three versions of $\mathcal{D}_E$ sets. It suggests that the reward model can generalize well to unlabeled data even when trained with a very limited number of demonstrations. We note that sizes of demonstrations sets are usually below 10 for the tasks considered.

6 Related work

Imitation is an important topic in the study of intelligence and consequently there is a vast body of literature on this topic. Osa et al. [23] present a comprehensive and authoritative review of imitation in robotics and machine learning. We refer readers to this review and provide a brief summary next.

The most obvious way to imitate is to mimic exactly what another agent is doing via supervised learning. This is often referred to as behaviour cloning (BC) [28]. As with all supervised learning methods, BC works best when large datasets of high quality demonstrations are available. This high sample complexity is caused by the fact that BC performs poorly in states that are far away from the training data, though some techniques have been proposed to mitigate this problem [31]. Moreover, since the focus of BC is on mimicking, it will not learn to supersede demonstrators or learn to select what to imitate to achieve a goal more effectively.

Inverse RL [22, 1] provides an alternative to BC. The focus is on learning a reward function from high-quality expert trajectories, and using this reward function to learn a policy. Recent examples of this approach [9, 14, 19, 10, 20, 39, 2], have been particularly successful when learning from states. However, learning effectively from raw input video streams remains a challenge, see for example [40]. These approaches typically require the online execution of RL agents for many trials, making them too costly for real robots. In contrast, in this paper we learn the rewards and policies directly from logged data, obviating the need for online execution during learning.

Demonstrations are often used to improve RL agents by guiding exploration [21, 29, 34, 27, 35, 41, 24]. This is often achieved by replaying these demonstrations, alongside recent online experiences, in model-free RL approaches. While we use demonstrations, we emphasize again that we learn purely from logged data, and without online deployment during learning.

Learning from positive and unlabeled data is a practical challenge in many applications, such as medical diagnosis and robotics [38]. PU learning approaches aim at solving this problem, and the idea of unbiased risk estimators was a breakthrough [8, 7, 15]. Xu and Denil [38] successfully applied PU learning in adversarial imitation and supervised reward learning.

Offline RL, often referred to as batch RL, has been reviewed by Lange et al. [17] and more recently by Levine et al. [18]. Many algorithmic variants have beep proposed over the last two years, including among others BCQ [12], MARWIL [36], BAIL [6], ABM [32] AWR [26], and CRR [37]. There has also been progress in benchmarking and offline hyper-parameter selection [13, 25].

Most offline RL works assume that rewards are given. The work of Cabi et al. [5] instead use a preference elicitation interface allowing humans to sketch reward curves, in order to learn reward functions. Similar to our case, these learned rewards are then provided to an offline RL agent (D4PG [4] in the case of [5] versus CRR [37] in this work). Our method differs mainly in that we do not need human annotators in the reward learning process, apart from potentially producing demonstration data. Also, our reward learning method uses all available demonstration and unlabeled set data, whereas [5] only learn rewards from sketched episodes.

7 Conclusion

We proposed offline reinforced imitation learning (ORIL) to enable learning from both demonstrations and a large unlabeled set of experiences without reward annotations. Leveraging unlabeled data allows ORIL to consistently outperform comparable BC agents. Given a modest number of demonstrations, adding more unlabeled experience improves the performance of ORIL across a diverse set of continuous control and simulated robotic manipulation tasks. We showed that ORIL is competitive with state-of-the-art methods that use ground-truth rewards, while ORIL itself does *not* rely on ground-truth rewards.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[2] Nir Baram, Oron Anschel, Itai Caspi, and Shie Mannor. End-to-end differentiable adversarial imitation learning. In *ICML*, 2017.

[3] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributional policy gradients. In *ICLR*, 2018.

[4] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *ICLR*, 2018.

[5] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, Oleg Sushkov, David Barker, Jonathan Scholz, Misha Denil, Nando de Freitas, and Ziyu Wang. Scaling data-driven robotics with reward sketching and batch reinforcement learning. In *RSS*, 2020.

[6] Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, Qing Deng, and Keith Ross. BAIL: Best-action imitation learning for batch deep reinforcement learning. In *NeurIPS*, 2019.

[7] Marthinus Du Plessis, Gang Niu, and Masashi Sugiyama. Convex formulation for learning from positive and unlabeled data. In *ICML*, 2015.

[8] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. In *SIGKDD*, 2008.

[9] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML*, 2016.

[10] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *ICLR*, 2018.

[11] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *Preprint arXiv:2004.07219*, 2020.

[12] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

[13] Çaglar Gülçehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel J. Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, Jerry Li, Mohammad Norouzi, Matthew D. Hoffman, Ofir Nachum, George Tucker, Nicolas Heess, and Nando de Freitas. RL unplugged: Benchmarks for offline reinforcement learning. In *NeurIPS*, 2020.

[14] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NeurIPS*, 2016.

[15] Ryuichi Kiryo, Gang Niu, Marthinus C Du Plessis, and Masashi Sugiyama. Positive-unlabeled learning with non-negative risk estimator. In *NeurIPS*, 2017.

[16] Ilya Kostrikov, Kumar Krishna Agrawal, Sergey Levine, and Jonathan Tompson. Addressing sample inefficiency and reward bias in inverse reinforcement learning. *Preprint arXiv:1809.02925*, 2018.

[17] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*. Springer Berlin Heidelberg, 2012.

[18] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *Preprint arXiv:2005.01643*, 2020.

[19] Yunzhu Li, Jiaming Song, and Stefano Ermon. InfoGAIL: Interpretable imitation learning from visual demonstrations. In *NeurIPS*, 2017.

[20] Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *Preprint arXiv:1707.02201*, 2017.

[21] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *ICRA*, 2018.

[22] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[23] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7, 2018.

[24] Tom Le Paine, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, Gabriel Barth-Maron, Ziyu Wang, Nando de Freitas, and Worlds Team. Making efficient use of demonstrations to solve hard exploration problems. In *ICLR*, 2020.

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

[26] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *Preprint arXiv:1910.00177*, 2019.

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

[28] Dean A Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *NeurIPS*, 1989.

[29] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *RSS*, 2018.

[30] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. SQIL: imitation learning via reinforcement learning with sparse rewards. In *ICLR*, 2020.

[31] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[32] Noah Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *ICLR*, 2020.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[34] Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *Preprint arXiv:1707.08817*, 2017.

[35] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. In *ICRA*, 2019.

[36] Qing Wang, Jiechao Xiong, Lei Han, peng sun, Han Liu, and Tong Zhang. Exponentially weighted imitation learning for batched historical data. In *NeurIPS*, 2018.

[37] Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. In *NeurIPS*, 2020.

[38] Danfei Xu and Misha Denil. Positive-unlabeled reward learning. In *CoRL*, 2020.

[39] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei A. Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and imitation learning for diverse visuomotor skills. In *RSS*, 2018.

[40] Konrad Zolna, Scott Reed, Alexander Novikov, Sergio Gomez Colmenarej, David Budden, Serkan Cabi, Misha Denil, Nando de Freitas, and Ziyu Wang. Task-relevant adversarial imitation learning. In *CoRL*, 2020.

[41] Konrad Zolna, Negar Rostamzadeh, Yoshua Bengio, Sungjin Ahn, and Pedro O Pinheiro. Reinforced imitation in heterogeneous action space. *Preprint arXiv:1904.03438*, 2020.

[42] Konrad Zolna, Chitwan Saharia, Leonard Boussioux, David Yu-Tung Hui, Maxime Chevalier-Boisvert, Dzmitry Bahdanau, and Yoshua Bengio. Combating false negatives in adversarial imitation learning. In *AAAI*, 2020.

Supplementary material

A Training procedure details

A.1 Algorithm details

We train a policy, critic and reward model simultaneously as shown in Algorithm 1. However, the reward model can be trained separately first, and then used to annotate datasets $\mathcal{D}_E$ and $\mathcal{D}_U$ with reward estimates. Both these approaches result in similar agent performance.

Algorithm 1 implements ORIL with PU learning (ORIL$_P$). The data augmentation is done as in [40]. The critic update is based on the discrepancy between the current estimate of the action-value and the TD-update (where the next step reward is estimated by reward model R_ψ). This discrepancy is measured by a divergence measure or a metric D, such as KL-divergence or squared error. In this paper, we use a distributional Q-function as in [37] and use the divergence measure proposed in [3].

Algorithm 1: Offline Reinforced Imitation Learning

Input: critic, policy and reward networks: Q_θ, π_ϕ and R_ψ, and its target versions: $Q_{\theta'}$, $\pi_{\phi'}$, $R_{\psi'}$, divergence measure D, expert dataset $\mathcal{D}_E$ and unlabeled dataset $\mathcal{D}_U$, hyperparameter η, number of updates $n_{updates}$;

Begin: Split $\mathcal{D}_U$ in half to get $\mathcal{D}_U^1$ and $\mathcal{D}_U^2$.
for $n_{updates}$ **do**
  `/* Reward learning */`
  Sample expert and unlabeled batches, i.e. $\mathcal{B}_E \subset \mathcal{D}_E$ and $\mathcal{B}_U \subset \mathcal{D}_U^1$;
  Augment images in $\mathcal{B}_E$ and $\mathcal{B}_U$ to obtain augmented batches $\overline{\mathcal{B}_E}$ and $\overline{\mathcal{B}_U}$;
  Train reward model using augmented batches $\overline{\mathcal{B}_E}$ and $\overline{\mathcal{B}_U}$:
    Update reward network parameters ψ with gradient:

$$-\nabla_\psi \eta \mathbb{E}_{s_t \sim \overline{\mathcal{B}_E}}[-\log(R_\psi(s_t))] + \mathbb{E}_{s_t \sim \overline{\mathcal{B}_U}}[-\log(1 - R_\psi(s_t))] - \eta \mathbb{E}_{s_t \sim \overline{\mathcal{B}_E}}[-\log(1 - R_\psi(s_t))]$$

  `/* Policy and critic learning */`
  Sample additional unlabeled batch from $\mathcal{D}_U^2$, i.e. $\mathcal{B}_U^2 \subset \mathcal{D}_U^2$;
  Concatenate (not augmented) expert and both unlabeled batches: $\mathcal{B} = \mathcal{B}_E \cup \mathcal{B}_U^1 \cup \mathcal{B}_U^2$;
  Apply CRR updates using rewards predicted by $R_{\psi'}$:
    Update critic network parameters θ with gradient:
      $-\nabla_\theta \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{B}} D\big[Q_\theta(s_t, a_t), R_{\psi'}(s_{t+1}) + \gamma \mathbb{E}_{a \sim \pi_{\phi'}(s_{t+1})} Q_{\theta'}(s_{t+1}, a)\big]$;
    Update policy network parameters ϕ with gradient:
      $-\nabla_\phi \mathbb{E}_{(s_t, a_t) \sim \mathcal{B}} \log \pi_\phi(a_t|s_t) \mathbb{I}\left[Q_\theta(s_t, a_t) > Q_\theta(s_t, \pi_\phi(s_t))\right]$;

  `/* Target networks update */`
  Update the target networks every N steps by copying parameters: $\theta' \leftarrow \theta,\ \ \phi' \leftarrow \phi,\ \psi' \leftarrow \psi$;
end

A.2 Hyperparameters

We parametrize the critic, policy and reward models with neural networks. The policy and critic architectures are identical to the ones used in Critic Regularized Regression work [37], and the reward model architecture is inspired by the critic network. All networks are described in details below and presented in Figure 7.

We use the same hyperparameters as in [37]. We train all our models for $1e6$ learner steps and always with 3 seeds. We compute the mean return between $5e5$ and $1e6$ learner steps for each seed. This gives three values of mean returns (one per seed) for every agent and we report mean, and standard deviation across these results.

For pixel based tasks (i.e. Robotic Manipulation tasks), pixels are encoded with a residual CNN (see Figure 7, bottom). Two separate image encoders are trained. One of them is shared between critic and policy networks, and another is used (and trained) solely by the reward model.

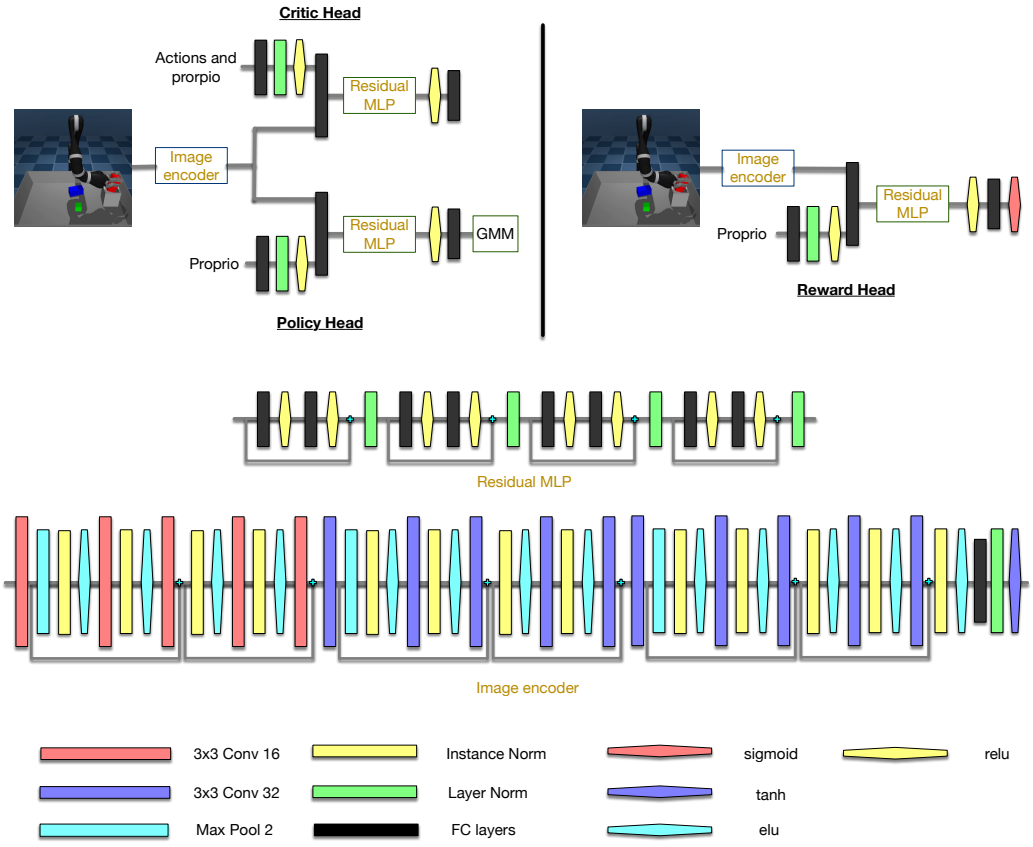

Figure 7: **The architectures used by the critic, policy and reward models.** There are two instances of the image encoder, one is shared between the critic and policy networks while the other is part of the reward network. All models implement the same residual MLP but it is never shared. GMM stands for Gaussian Mixture Model which outputs the final policy prediction (see details in the main text). This figure is based on the architecture figure from CRR paper [37].

All networks process proprio states (for critic, they are concatenated with actions) with one layer MLP to obtain preliminary representations which are then concatenated with image representations (for pixel based tasks) and further processed. The final layers depend on the network purpose (see details below).

Policy head Proprioceptive input is processed with a fully connected layer, layer normalization and tanh activation function, then concatenated with pixel encoding (when present) and passed into a residual MLP (see Figure 7, top left). The output of the MLP defines the policy which is a mixture of five multivariate Gaussians. Specifically, the output of the MLP is contains: five mean vectors; five vectors that (after passing through a softplus function) define diagonal of the covariance matrix of each Gaussian; and five scalars that define mixture log-probabilities.

Critic head Proprioceptive input concatenated with actions is processed through a fully connected layer, layer normalization and tanh, then concatenated with pixel encoding (when present) and passed into a residual MLP (see Figure 7, top left). The output of the MLP defines a discrete distribution of the distributional critic.

Reward head Proprioceptive input is processed through a fully connected layer, layer normalization and tanh, then concatenated with pixel encoding (when present) and passed into a residual MLP (see Figure 7, top right). The output is proccesed by a fully connected layer to obtain a scalar which is then scaled to $(0, 1)$ by applying sigmoid.

B Positive-unlabeled learning

If we treat reward learning as a binary decision problem to distinguish success ($\mathcal{D}_E$) and failure ($\mathcal{D}_F$) trajectories, we can write the loss for the reward model R as

$$\eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(R(s_t))] + (1-\eta)\mathbb{E}_{s_t \sim \mathcal{D}_F}[-\log(1-R(s_t))], \tag{3}$$

where η is the proportion of the trajectory space corresponding to success. This corresponds to a more general form of Equation 1 where before $\eta = 0.5$ was assumed.

The key insight of PU-learning [8, 7] is to observe that the second term in the loss (computed for $\mathcal{D}_F$) can be written in terms of a dataset of unlabeled trajectories $\mathcal{D}_U$ (that contains an unknown mixture of successes and failures) and the dataset of positive trajectories $\mathcal{D}_E$:

$$(1-\eta)\mathbb{E}_{s_t \sim \mathcal{D}_F}[-\log(1-R(s_t))] = \mathbb{E}_{s_t \sim \mathcal{D}_U}[-\log(1-R(s_t))] - \eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(1-R(s_t))]. \tag{4}$$

This allows the loss for the reward model to be rewritten in a way that avoids any dependence on explicitly labeled failures

$$\eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(R(s_t))] + \mathbb{E}_{s_t \sim \mathcal{D}_U}[-\log(1-R(s_t))] - \eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(1-R(s_t))] \tag{5}$$

In this paper we treat η as a hyperparameter and set it to $\eta = 0.5$ throughout.

C Task-relevant adversarial imitation learning

TRAIL proposes to constrain the GAIL discriminator such that it is *not* able to distinguish between certain, preselected expert and agent observations which do not contain task behavior. For example, the discriminator should not be able to distinguish whether early observations at the very start of an episode come from the demonstration or unlabeled set, since no meaningful behavior has yet been performed.

We adapt TRAIL to the offline setting and use early observations to form our constraint sets. Specifically, we construct subsets of early states, i.e. $\mathcal{D}'_U = \{s_t \in \mathcal{D}_U \mid t < 10\}$, and analogously $\mathcal{D}'_E$ for $\mathcal{D}_E$. We compute reward loss as in Equation 1 for the original datasets and also, separately, for the states from the constraint sets. The loss optimized by the reward model is the following:

$$L_\psi(\mathcal{D}_E, \mathcal{D}_U) - \mathbf{1}_{R_\psi(\mathcal{D}'_U) > R_\psi(\mathcal{D}'_E)} L_\psi(\mathcal{D}'_E, \mathcal{D}'_U), \tag{6}$$

where $\mathbf{1}_{R_\psi(\mathcal{D}'_E) > R_\psi(\mathcal{D}'_U)}$ is one if average prediction R_ψ for expert early observations ($\mathcal{D}'_E$) is higher than for unlabeled early observations ($\mathcal{D}'_U$). Intuitively, we use early observations to first control if the reward model is overfitting, and if so, we use them again to compute reversed loss to regularize discriminator. We refer to the original paper [40] for motivation and detailed description.

The original paper [40] proposes another indicator for applying reversed loss, but we find the average prediction for early observations used by us to work similarly well.