TARGETED ENVIRONMENT DESIGN FROM OFFLINE DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

In reinforcement learning (RL) the use of simulators is ubiquitous, allowing cheaper and safer agent training than training directly in the real target environment. However, this approach relies on the simulator being a sufficiently accurate reflection of the target environment, which is difficult to achieve in practice. Accordingly, recent methods have proposed an alternative paradigm, utilizing *offline* datasets from the target environment to train an agent, avoiding online access to either the target or any simulated environment but leading to poor generalization outside the support of the offline data. Here, we propose to combine these two paradigms to leverage both offline datasets and synthetic simulators. We formalize our approach as *offline targeted environment design* (OTED), which automatically learns a distribution over simulator parameters to match a provided offline dataset, and then uses the learned simulator to train an RL agent in standard online fashion. We derive an objective for learning the simulator parameters which corresponds to minimizing a divergence between the target offline dataset and the state-action distribution induced by the simulator. We evaluate our method on standard offline RL benchmarks and show that it yields impressive results compared to existing approaches, thus successfully leveraging both offline datasets and simulators for better RL.

1 INTRODUCTION

Recent years have witnessed the proliferation of reinforcement learning (RL) as a way to train agents to solve a variety of tasks, encompassing game playing Brown & Sandholm (2019); Silver et al. (2016) and robotics Akkaya et al. (2019); Schulman et al. (2015); Chiang et al. (2019), among other applications. These successes often rely on the use of high-fidelity simulators, allowing for an agent to be trained on large amounts of online data collected at relatively low cost. In many instances, for example in game playing, an accurate simulator is easy to devise. However, in other applications such as robotics Kalashnikov et al. (2018); Todorov et al. (2012) and health Ernst et al. (2006), a simulator is only a rough approximation to the real target environment, and properly designing the simulator to match a desired target environment can be a highly manual process Nachum et al. (2019a); Tolani et al. (2021); Krishnan et al. (2021), potentially requiring repeated online access to the target environment Chebotar et al. (2019); Ramos et al. (2019); Mehta et al. (2020).

Due to the difficulty in matching simulators to target environments and motivated by the successes of supervised learning on large static datasets, a recent line of works Fujimoto et al. (2019); Agarwal et al. (2020); Wu et al. (2019); Levine et al. (2020); Fu et al. (2020); Gulcehre et al. (2020) has proposed to circumvent online training altogether (whether on the simulator or in the target environment) and instead perform RL on an *offline* dataset of logged experience in the target environment, typically collected by an unknown behavior policy. This approach avoids any potential mismatch between training and test environment. However, existing algorithms for offline RL are highly liable to generalization errors, and so these methods often impose a strong regularization on the learned policy to maintain proximity to the offline dataset Fujimoto et al. (2019); Wu et al. (2019); Kostrikov et al. (2021); Yu et al. (2020). Thus, such methods can rarely approach the performance of online RL on an accurate simulator.

In this work, we propose to leverage both approximate simulators and offline datasets to train agents. Namely, we consider settings in which one has access to an offline dataset collected from a target environment as well an approximate (but nevertheless inaccurate) simulator. How can we leverage both of these to learn a near-optimal policy for the target environment?

To tackle this problem, we propose an approach which, at a high level, is composed of two stages: (1) use the offline dataset to learn a distribution over simulator parameters; (2) train an RL agent in an online fashion on the simulator using any off-the-shelf RL algorithm. The key question is then how to appropriately learn the simulator parameters from an offline dataset, which we term *offline targeted environment design* (OTED). For this problem, we devise an objective quantifying the divergence between the state-action distribution appearing in the offline dataset and the state-action distribution induced by a learned behavior policy acting in the simulator. We show that the simulator parameters and the target behavior policy can be jointly learned such that even a suboptimal behavior policy, that would otherwise perform poorly on the target environment, can help learn simulator parameters very effectively. We further show how the performance of the final returned policy can be improved by employing OTED not only for policy training but also for offline policy *selection* Yang et al. (2020); Fu et al. (2021). Namely, we show how one may use the OTED-learned simulators to both provide a set of candidate policies as well as evaluate those policies in order to rank and choose the best policy to return, reminiscent of cross-validation techniques popular supervised learning Arlot & Celisse (2010) but so far elusive in RL contexts (Paine et al., 2020).

We apply our method to a variety of domains, including MiniGrid Chevalier-Boisvert et al. (2018) and D4RL Fu et al. (2020), showing that our method is able to recover the target environment parameter using as few as 5 demonstrations and learn from high-dimensional image observations. We also show that an online agent trained in designed simulators yields improved performance compared to methods which leverage only simulators or only offline datasets, as well as existing methods for domain transfer Tobin et al. (2017). On the D4RL Fu et al. (2020) benchmark, we demonstrate that our method can even outperform an online SAC agent trained on the *ground-truth target* environment. Our method achieves up to 17 times higher score compared to previous state-of-the-art offline RL and behavior cloning (BC) methods as well as domain randomization. On datasets with medium-level behavior that can be collected without domain experts, we show more relative improvements where our model even reaches the performance of the offline RL methods trained on expert-level behaviors.

2 Related Work

Our work focuses on learning simulator parameters to match a target environment and is thus related to a large and diverse literature on *domain transfer* in RL. One of the most common and simple approaches to handling unknown simulator parameters is *domain randomization* Sadeghi & Levine (2016); Tobin et al. (2017); Matas et al. (2018); Chiang et al. (2019); Tolani et al. (2021); Krishnan et al. (2021), which suggests to randomize the setting of those unknown parameters during training, thus learning an agent which is robust to a wide distribution of possible simulators. While domain randomization has demonstrated successes, the process of choosing appropriate ranges for unknown parameters can be a highly manual process Andrychowicz et al. (2020), in which one must balance between too wide a range that hampers agent training and too narrow a range which may exclude the target environment parameterization Nachum et al. (2019a). While some works have proposed heuristic mechanisms to better automate this process Akkaya et al. (2019); Krishnan et al. (2021), others have explored incorporating online interaction with the environment into this tuning process Chebotar et al. (2019); Ramos et al. (2019); Mehta et al. (2020). In this work, we avoid any online interactions with the target environment during training, and only assume access to a static offline dataset.

Our approach of using an offline dataset to learn a simulator can be related to recent model-based approaches to offline RL Matsushima et al. (2020); Yu et al. (2020); Argenson & Dulac-Arnold (2020). In these works, the offline dataset learns approximate dynamics and reward models of the environment, and the agent optimizes behavior in this approximate model of the environment. Although this makes minimal assumptions on the target environment, using reward and dynamics models as proxies for the environment leads to extrapolation issues which necessitate strong regularizers on the learned policy. Moreover, these existing approaches mostly ignore the fact that in many scenarios, one already has an approximate simulator, albeit not differentiable; e.g., the physical laws in a continuous control environments are straightforward to implement Todorov et al. (2012), although parameters more specific to the environment (friction coefficient, actuator gain, gravity) are unknown. Leveraging these simulators and focusing learning on the unknown parameters, as performed by our approach, can thus improve over modelling the full dynamics from scratch.

Our proposed objective is based on a distribution matching loss. Similar losses have appeared in the past, especially in generative models Goodfellow et al. (2014), off-policy evaluation Nachum et al. (2019b), off-policy learning Nachum et al. (2019c), and imitation learning Ho & Ermon (2016); Stadie et al. (2017); Kostrikov et al. (2020). These predominantly focus on learning a policy via an adversarial objective and ignore the mismatch between simulators and real environments. Even when the distribution matching objective is optimal, this mismatch might give very suboptimal policies, due to potential poor support of the data distribution. This is a very realistic scenario since tuning simulator parameters accurately is a very tedious process and requires domain-experts.

3 BACKGROUND

We consider an RL setting in which we are given access to both an offline dataset of experience from the target environment and an approximate simulator, in which the agent can gather more experience. We elaborate on the notation and background relevant to this setting.

Reinforcement Learning We define an environment as a Markov Decision Process (MDP) given by a tuple $(S, A, \rho_0, T, r, \gamma)$ where S is a state space, A is an action space, ρ_0 is an initial state distribution, $T(s'|s, a)$ is a state transition distribution, $r : S \times A \to \mathbb{R}$ is a reward function over state and actions, and $\gamma \in [0, 1)$ is a discount factor. A stationary policy π in this environment is a function from states to distributions over actions. In reinforcement learning, we are interested in finding a policy π that maximizes the cumulative discounted returns: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 \sim \rho_0, a_t \sim \pi(s_t), s_{t+1} \sim T(s_t, a_t)\right]$. We will use $d^\pi(s, a)$ to denote the *state-action distribution* of π: $d^\pi(s, a) := (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \Pr[s_t = s, a_t = a | s_0 \sim \rho_0, a_t \sim \pi(s_t), s_{t+1} \sim T(s_t, a_t)]$.

Simulator vs. Target Environment Typically in online reinforcement learning, a policy is trained based on experience collected by the agent in a *simulator*. While this implicitly assumes that the simulator can perfectly mimic the target environment, in practice this is often not the case. The simulator typically only provides an approximation to the true initial state distribution ρ_0 and state transition distribution T.[1] For example in robotics applications, while physics simulators can be built, it is infeasible to accurately set all appropriate parameters (e.g., friction coefficients, actuator gains, wind speed). In this work, we assume such a parameterized simulator exists, and it is given by $\mathcal{M}(z) := (S, A, \rho_0(\cdot|z), T(\cdot|\cdot, \cdot, z), r, \gamma)$, where z denotes the parameters of the simulated environment. Naive approaches would either set z heuristically or using randomization methods. In this work, we aim to use offline datasets to learn z so that $\mathcal{M}(z)$ approximates the target MDP, which we denote as $\mathcal{M}^*$. We use $d^\pi(s, a|z)$ to denote the state-action distribution of π in the parameterized simulator $\mathcal{M}(z)$.

Offline Reinforcement Learning Offline RL, as an alternative to online RL on simulators, assumes that an agent cannot collect any online experience and is restricted to a dataset D of (s, a, s') collected by an unknown behavior policy μ; i.e., $(s, a) \sim d^\mu, s' \sim p(s, a)$. We use $d^D(s, a)$ to denote the distribution over state-action pairs appearing in D. When trained on a fixed dataset, previous state-of-the-art online reinforcement learning algorithms such as SAC Haarnoja et al. (2019) underperform due to extrapolation errors Fujimoto et al. (2019); Kumar et al. (2019), which stem from sparsity of the samples in the dataset. Accordingly, many RL algorithms designed specifically for the offline setting make use of strong regularizations to maintain proximity to the offline dataset Wu et al. (2019); Kostrikov et al. (2021); Yu et al. (2020).

4 OTED: OFFLINE TARGETED ENVIRONMENT DESIGN

We now describe our method for leveraging both offline datasets and simulators to learn return-maximizing policies, thus alleviating both of the above issues: (i) data sparsity and extrapolation

[1]For simplicity, we assume the reward function is known, although our method can be readily extended to handle unknown rewards.

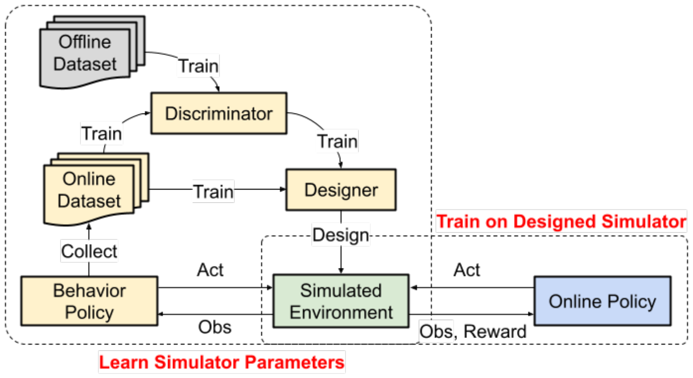


Figure 1: Outline of OTED: The simulator parameters are learned by the *designer*, which uses signals from a learned *discriminator* to match the distribution appearing in the offline dataset to that induced by a learned behavior policy acting in the environment. The learned simulator parameters are then used to train a target policy via standard online RL.

issues when performing offline learning and (ii) inaccuracy of an approximate simulator for online learning.

At a high-level, our algorithm can be summarized as follows, also illustrated in Figure 1.

1. Learn an approximate behavior policy $\hat{\mu}$ from D using any off-the-shelf imitation learning algorithm; e.g., behavioral cloning Pomerleau (1991) or ValueDICE Kostrikov et al. (2020).
2. Learn a distribution $q(z)$, also called *designer*, over simulator parameters so that $d^{\hat{\mu}}(s, a|z)$ approximates the state-action distribution appearing in D.
3. Use any off-the-shelf online RL algorithm – e.g., SAC Haarnoja et al. (2019) – to learn a return-maximizing policy π on the simulator $\mathcal{M}(z)$, where z is sampled anew from the learned distribution $q(z)$ at the beginning of each episode.

Steps (1) and (3) above are straightforward, and so we focus this section on first elaborating on step (2) below, and this is also summarized in Algorithm 1. We then continue to show how we reduce variance inherent in the procedural three-step process above by using replicated experiments, distributions over mixtures of simulator parameters, and model selection without online access (see Algorithm 2 for a pseudocode).

4.1 Learning Simulator Parameters via Distribution Matching

Given an approximate behavior policy $\hat{\mu}$, we propose to learn a distribution $q(z)$ over simulator parameters so that $d^{\hat{\mu}}(s, a|z)$ approximates the state-action distribution appearing in D. Thus, our objective for $q(z)$ is based on a probability divergence between $d^{\hat{\mu}}(s, a|z)$ and $d^D(s, a)$. To encourage better generalization, we also introduce a prior $p(z)$, and measure a divergence between $d^{\hat{\mu}}(s, a|z)q(z)$ and $d^D(s, a)p(z)$. Our proposed objective is given by,

$$D_{KL}(d^{\hat{\mu}}(s, a|z)q(z)\|d^D(s, a)p(z)) = \mathop{\mathbb{E}}_{\substack{z\sim q(z)\\(s,a)\sim d^{\hat{\mu}}(s,a|z)}}\left[-\log\frac{d^D(s, a)}{d^{\hat{\mu}}(s, a|z)}\right] + D_{KL}(q(z)|p(z)). \quad (1)$$

As desired, it is clear that the first term above encourages the parameter distribution $q(z)$ to match the state-action distribution of the approximate behavior policy $\hat{\mu}$ in the simulator to that of the dataset; meanwhile, the second term regularizes the simulator distribution from collapsing. We can immediately see that the objective above is an RL (actually, a bandit) objective for q, where the actions are z, the stochastic reward is given by $\log \frac{d^D(s,a)}{d^{\hat{\mu}}(s,a|z)}$ for $(s, a) \sim d^{\hat{\mu}}(s, a|z)$, and a relative entropy regularizer is applied to q. Therefore, one may use any off-the-shelf policy gradient algorithm to learn q, as long as one has access to the density ratios $\frac{d^D(s,a)}{d^{\hat{\mu}}(s,a|z)}$.

However, in general one does not have access to the densities $d^{\hat{\mu}}$ or d^D explicitly. Nevertheless, one can easily sample from either density; samples from $d^{\hat{\mu}}$ are given by running the simulator $\mathcal{M}(z)$ and samples from d^D are given by the dataset D itself. A number of algorithms exist for estimating density ratios of two distributions given sampling access. In our implementation, we use the discriminative approach based on Nguyen et al. (2010). Namely, we train a discriminator $g(s, a)$ to minimize the objective

$$\mathbb{E}_{(s,a)\sim d^{\hat{\mu}}}[-\log(1 - g(s, a))] + \mathbb{E}_{(s,a)\sim d^D}[-\log g(s, a)], \tag{2}$$

similar to the discriminator in GAIL Ho & Ermon (2016). When trained to optimality, the discriminator will satisfy the following, $\log \frac{g^*(s,a)}{1-g^*(s,a)} = \log \frac{d^D(s,a)}{d^{\mu}(s,a,z)}$, and this can be used as a reward signal for training q.

In practice, to allow for more flexibility in the trade-off between exploitation and safety, we extend the objective with a tunable parameter ϵ:

$$-\mathop{\mathbb{E}}_{\substack{z\sim q(z)\\(s,a)\sim d^{\hat{\mu}}(s,a|z)}}\left[\log \frac{d^D(s, a)}{d^{\hat{\mu}}(s, a|z)}\right] + \epsilon D_{KL}(q(z)|p(z)). \tag{3}$$

A small ϵ encourages q to exploit the signal from the log-ratios, while a large ϵ encourages q to stay close to the prior p. In addition to serving as a trade-off parameter, ϵ can also be interpreted as (i) a reward scaling term in front of log ratios, i.e., $\frac{1}{\epsilon} \log \frac{d^D(s,a)}{d^{\mu}(s,a|z)}$ or (ii) a distribution smoothing term applied to both d^{μ} and d^D, i.e., $\log \frac{(d^D(s,a))^{\frac{1}{\epsilon}}}{(d^{\mu}(s,a|z))^{\frac{1}{\epsilon}}}$.

Algorithm 1 Training of the simulator parameters and the discriminator

```
Input: Dataset D, initial behavior policy $\mu$, initialized discriminator $g$, tunable simulator $S$

1: Gather experience, $E$, by running the policy $\mu$ in the simulator $S$.
2: Estimate log-ratio, $\log \frac{g(E)}{1-g(E)}$, of the simulated experience using the discriminator.
3: Update $q(z)$ using the objective in Eq. 3 with $\frac{g(E)}{1-g(E)}$ as the reward.
4: Update $g(s, a)$ using mini-batches from $(E, D)$.
5: Repeat above.

Return: $(q(z), g(s, a))$
```

Algorithm 2 Training of a single online agent using off-policy data and multiple simulator models

```
Input: Simulator distributions $q_i(z)$ from $N$ different experiments, simulator $S$, empty replay $R$

1: Sample $i \sim \mathcal{U}[1, N]$ and set the simulator $S$ using the mode of the distribution $q_i(z)$.
2: Run the online agent $\pi(a|s; z)$ in the simulator and populate the replay with new experience.
3: Sample a mini-batch from the replay and update the agent parameters.
4: Repeat above.

Return: $\pi(a|s; z)$
```

Our objective, Eq. 1, resembles mutual information when the joint distribution is defined using the proposal distribution and marginal distributions are defined using expert data distribution. The main difference is that we use the behavior policy in the joint distribution and use expert policy in marginals while in mutual information they are the same. Through optimizing the behavior policy to mimic the expert better, the objective will get closer to a mutual information formulation.

4.2 Reducing Variance with Replicated Experiments

Our proposed three step process – approximate $\hat{\mu}$, then learn $q(z)$, and finally training – introduces noise at each step due to randomness in optimizers and learning algorithms. It is important to reduce the variance inherent in this process so that the returned policy is best prepared for evaluation in the target environment. We tackle this problem via two techniques: *simulator ensembles* and *simulator-based policy selection*.

Simulator ensembles To increase the chance that the policy transfers well to the target environment, we borrow ideas from domain randomization and replicate our step (2) to generate multiple simulator distributions $q_i(z)$. Although the distributions are all trained the same way, each one will induce slightly different parameter distributions due to randomization in learning. To mitigate the effects of this randomness, we train our online agent on a mixture of these distributions. More specifically, we set the simulator with new parameters at each online episode collection, with these parameters sampled uniformly from the mode of each parameter distribution $q_i(z)$.

Simulator-based policy selection Even with the ensemble approach described above, multiple online RL training runs on the simulator can result in drastically different policies, again due to randomness in the learning process. This presents the problem of offline policy selection, which is a well-known challenge for real-world RL Yang et al. (2020); Paine et al. (2020); Fu et al. (2021). We propose a simple approach to this. Given a set of policies π_j, we evaluate and rank them based on online performance in a learned simulator distribution $\overline{q}(z)$ on which the policies were trained. The best performing policy in the simulator is then the final returned policy.

One can view this offline evaluation as a form of rudimentary cross-validation, similar to existing techniques in offline policy selection which evaluate a learned policy according to Q-values learned *separate* from the Q-values used to train the policy Paine et al. (2020). Although the separately-trained Q-values (like $\overline{q}(z)$) are trained on the same offline dataset, this simple approach has been demonstrated to achieve strong performance, even compared to more sophisticated methods Fu et al. (2021).

5 Experiments

We evaluate OTED, in a variety of settings; different tasks, data sources and simulator parameterizations. We first evaluate our method against state-of-the-art offline RL methods on previous benchmarks including the D4RL benchmark Fu et al. (2020). Next, we experiment with a discrete MiniGrid Chevalier-Boisvert et al. (2018) environment to illustrate that OTED can also learn from high-dimensional image observations. We implemented our approach using ACME Hoffman et al. (2020) and TensorFlow Abadi et al. (2015) open-source libraries. We used a single CPU which required a total of 24 hours to train per seed. We report results with 5 different seeds.

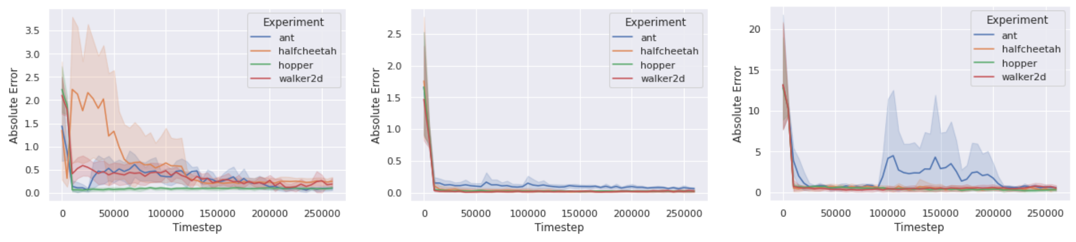
5.1 Setup

5.1.1 Continuous Environments

We use the Mujoco Todorov et al. (2012) simulator and study 4 different continuous control tasks that have been adopted by the offline RL community, *Ant, HalfCheetah, Hopper,* and *Walker2d*. We modify the simulators so that some of the physics parameters are unknown. We consider 3 different parameterizations: *geom-friction*, *actuator-gainprm*, and *gravity* that causes different effects on the underlying simulator physics. These are 1-dimensional parameters and we learn them separately as well as jointly in 3-dimensions. We constrain the search space for the *geom-friction* and *actuator-gainprm* parameters to be within $[-5, 5]$ and for the *gravity* parameter and joint parameters to be within $[-20, 20]$. See Appendix B for a list of ground truth design parameters.

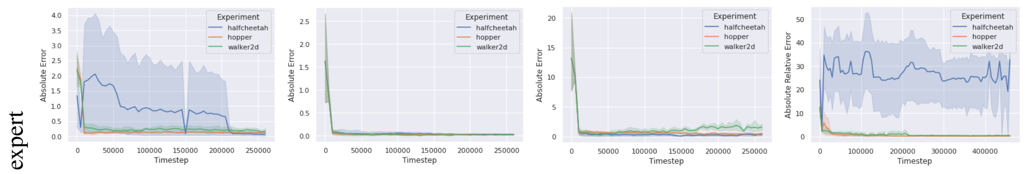
We use two different offline demonstration datasets for continuous environments: (i) 5 trajectories used in ValueDICE Kostrikov et al. (2020) and (ii) D4RL Fu et al. (2020) benchmark datasets divided into 3 levels, *medium*, *medium-expert*, and *expert*, reflecting the competency of the behavior policy used to collect the data. We use the ValueDICE objective to induce behavior policy for the first dataset and behavior cloning for the D4RL benchmark.

We compare OTED to the Domain randomization (DR) Tobin et al. (2017) and a custom baseline that we introduce called Interval Search with online feedback (IS). For DR, we set the simulator with a randomly sampled parameter from the corresponding interval at the beginning of each episode and train an online agent in the simulated environment. DR can be very conservative when the



(a) Friction (b) Gain PRM (c) Gravity

Figure 2: Absolute design error of OTED using different parameterizations on ValueDICE demonstrations. OTED reliably learns to recover the target MDP with a small error on all environments.



(a) geom-friction (b) actuator-gainprm (c) gravity (d) all parameters

Figure 3: Absolute error of OTED on different simulator parameters using the D4RL expert dataset. For *all* parameters, we used absolute relative error as the scale of each parameter is different.

dependency between the simulator and parameter is highly nonlinear. As an alternative, we introduce an interval search baseline (IS) where a smaller initial interval is gradually expanded over time based on the performance of an online agent. We first sample a random parameter value c to be the center of an initial search interval, i.e., $[c - 0.1, c + 0.1]$. We evaluate the performance of the agent at every 10000 steps and if the performance of the agent is improved, we expand the interval by increments of 0.1, i.e., the above interval would become $[c - 0.2, c + 0.2]$.

5.1.2 MiniGrid

We use a discrete *MiniGrid* environment where observations are gray-scale images of the current view of an agent. The agent is tasked with navigating the grid by avoiding obstacles to reach to a goal state. We parameterize a MiniGrid environment by the *ratio of obstacles* in the grid. For example, a ratio of 0.3 would give 30 obstacles with random shapes and locations. We use a 10-by-10 grid and both the agent and the goal are placed randomly on empty cells. We constrain the search space of the ratio parameter to be within $[0, 0.6]$ as beyond 60 obstacles, the goal is mostly unreachable. We further quantize this search interval into 13 discrete values by increments of 0.05, i.e., $\{0.0, 0.05, \cdots, 0.6\}$. We experiment with three different target environment parameters: $[0.2, 0.26, 0.3]$. Note that while 0.2 and 0.3 are within the support set of the search distribution, 0.26 is not where the best approximation is 0.25. To easily collect a diverse set of demonstrations and to estimate the KL-divergence term accurately, we use a randomly initialized behavior policy network whose weights are known and fixed.

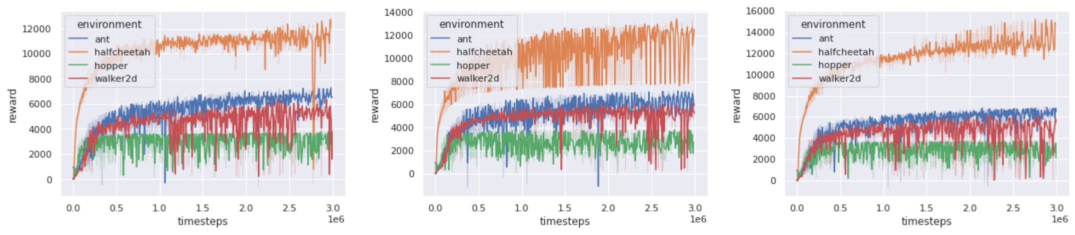
5.1.3 Metrics and Online Policy

We use two metrics to evaluate the accuracy of OTED: (i) *absolute error* which is the absolute difference between predicted parameter and ground truth parameter and (ii) *absolute relative error* which is the absolute error divided by the absolute value of the ground truth parameter. We use SAC as the online agent specifically as it is not designed to work in the offline RL setting. We call an online SAC agent trained in designed environments OTED-SAC and report its performance in the ground truth target environments.

5.2 Continuous Environment Results

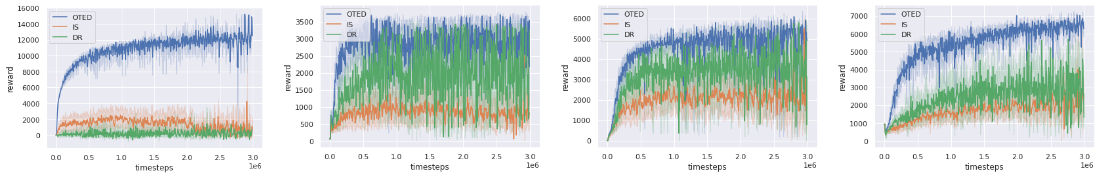
5.2.1 Error of the Continuous Simulator Parameter Predictions

We first evaluate OTED on 5 ValueDICE demonstrations. OTED recovers the unknown target environment parameter with low absolute error on all parameterizations (Figure 2). We observe that



(a) Friction (b) Gain PRM (c) Gravity

Figure 4: Performance of the OTED-SAC using ValueDICE dataset. On all parameters, it is on-par with the online SAC trained in the ground-truth target environment. On gravity parameter, we see better performance.



(a) HalfCheetah (b) Hopper (c) Walker2D (d) Ant

Figure 5: Comparison of OTED-SAC to baseline DR and IS methods on ValueDICE dataset.

different tasks show varying sensitivity to different parameters. For example, on *actuator-gainprm*, all environments exhibit low error and relatively low variance while on *geom-friction*, we observe relatively high error and high variance. Ant is one of the most challenging environments where we get higher error on all settings. We hypothesize that the reason is higher number of dimensions in the observation space (111 dimensions which is around 7 times higher than others) which makes it more difficult to learn using a small demonstration dataset.

OTED performs similarly on the D4RL expert dataset (Figure 3). As an interesting use case, HalfCheetah emits a very high variance on the *geom-friction* parameter early on but converges to a smaller error. This is due to one experiment giving a much higher error and taking much longer to find the correct parameter. See Appendix C for more results with different expert levels.

We see that OTED easily learns in Walker2D and Ant environments with very small absolute relative error. Sensitivity of HalfCheetah is also amplified, giving a high relative error.

5.2.2 Performance of the Online Agent with Designed Simulators

As OTED exhibits low absolute error on average, the natural question to ask is if this correlates with the performance of an online agent trained on designed simulators. In Figure 4, we plot the performance of OTED-SAC trained in tuned simulators using ValueDICE demonstrations. Across all experiments, we observe consistently good results where we achieve on par with an online state-of-the-art SAC agent that is trained on the ground truth target environment. We also compare with baseline domain transfer methods in Figure 5 where OTED-SAC consistently performs better. Figure 5 also demonstrates that a misspecified simulator would significantly hurt the performance of a SAC agent.

We present our results on D4RL benchmark in Table 1. For medium-level expert datasets, OTED-SAC clearly outperforms previous offline RL methods. The only exception is Hopper environment with the *gravity* parameter. Even though Walker2D environments exhibit higher variance on *geom-friction* parameter for medium and medium-expert level experts, OTED-SAC outperforms previous models and also performs better compared to other parameterizations. In majority of the cases, OTED-SAC achieves more than 100% performance, outperforming a baseline SAC agent trained on the ground-truth target environment.

	BC	F-BRC	MBOP	DR	IS	OTED geom friction	 actuator gainprm	 gravity
halfcheetah-medium	36.1	41.3	44.6	6.7	12.2	97.0	106.7	**118.8**
halfcheetah-medium-expert	35.8	93.3	105.9	6.7	12.2	80.0	109.3	**113.6**
halfcheetah-expert	107.0	108.4	-	6.7	12.2	**119.0**	109.6	110.5
hopper-medium	29.0	99.4	48.8	73.9	20.9	**107.3**	94.2	80.4
hopper-medium-expert	111.9	112.4	55.1	73.9	20.9	**114.0**	98.5	109.6
hopper-expert	109.0	112.3	-	73.9	20.9	**113.8**	100.4	72.3
walker2d-medium	6.6	78.8	41.0	46.2	26.9	**113.4**	97.3	109.1
walker2d-medium-expert	11.3	105.2	70.2	46.2	26.9	110.4	60.1	**121.4**
walker2d-expert	**125.7**	103.0	-	46.2	26.9	119.7	102.8	103.6

Table 1: Performance comparison of our work (OTED) to existing offline RL algorithms – behavioral cloning (BC), F-BRC Kostrikov et al. (2021), MBOP Argenson & Dulac-Arnold (2020) – as well as domain transfer algorithms – domain randomization (DR), interval search (IS) – on D4RL dataset.

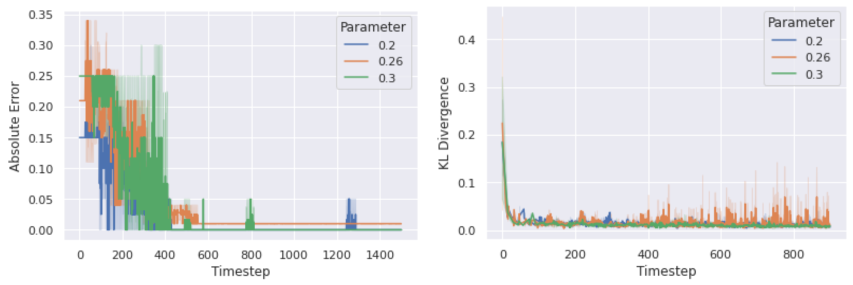


Figure 6: Absolute design error of OTED and KL-divergence between induced behavior policy and target policy in MiniGrid environment.

5.3 MiniGrid Environment Results

We show our MiniGrid results in Figure 6 where we present absolute error between learned parameter and ground truth parameter. After being trained for only several hundred steps, the discriminator learns to distinguish between generated and expert data, emitting a very strong signal for the designer to learn the simulator parameter. Note that 0.26 is not in the support set of the parameter distribution as we quantize by increments of 0.05 but OTED is still able to find the optimal prediction. This shows (i) the capability of OTED to generalize to observations that are not seen during training and (ii) the robustness to quantization errors. This also explains why it takes slightly longer for the discriminator to learn.

As suggested by small KL divergence, induced behavior policy closely mimics the target policy. This is crucial since the training data to both the discriminator and the designer come from interaction of the behavior policy with the simulator. While the mode of the distribution exhibits a much higher probability, we still observe a significant mass around the mode. This is expected as obstacles are randomly placed in the grid and ratio of obstacles from different observations might be similar even if they come from different environments.

6 Conclusion

We introduced a principled method, OTED, to learning simulator parameters of a target environment from offline datasets, combining two different paradigms, namely offline and online RL. We proposed a KL-divergence regularized distribution matching objective where a distribution over simulator parameters is learned to minimize the discrepancy between the offline dataset and state-action distribution of an online behavior policy in the induced simulator. By jointly learning the behavior policy and design distribution, we showed that OTED is able to recover the unknown parameter with high accuracy. When evaluated on previous benchmarks, including D4RL, OTED exhibited low absolute error on average. An online SAC agent (OTED-SAC) trained with designed simulators performed on par or better than previous state-of-the-art offline and online RL methods.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning, 2020.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

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

Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019. ISSN 0036-8075. doi: 10.1126/science.aay2400. URL `https://science.sciencemag.org/content/365/6456/885`.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979. IEEE, 2019.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019. doi: 10.1109/LRA.2019.2899918.

Damien Ernst, Guy-Bart Stan, Jorge Goncalves, and Louis Wehenkel. Clinical data based optimal sti strategies for hiv: a reinforcement learning approach. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 667–672. IEEE, 2006.

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

Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, Ziyu Wang, Alexander Novikov, Mengjiao Yang, Michael R Zhang, Yutian Chen, Aviral Kumar, et al. Benchmarks for deep off-policy evaluation. *arXiv preprint arXiv:2103.16596*, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062, 2019.

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

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

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf`.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL `https://arxiv.org/abs/2006.00979`.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto (eds.), *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pp. 651–673. PMLR, 29–31 Oct 2018.

Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation learning via off-policy distribution matching. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Hyg-JC4FDr`.

Ilya Kostrikov, Jonathan Tompson, Rob Fergus, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

Srivatsan Krishnan, Behzad Borojerdian, William Fu, Aleksandra Faust, and Vijay Janapa Reddi. Air learning: An ai research platform for algorithm-hardware benchmarking of autonomous aerial robots. *Mach Learn special issue on RL4RL*, 2021.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, 2019.

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

Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pp. 734–743. PMLR, 2018.

Tatsuya Matsushima, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu. Deployment-efficient reinforcement learning via model-based offline optimization. *arXiv preprint arXiv:2006.03647*, 2020.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pp. 1162–1176. PMLR, 2020.

Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*, 2019a.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *arXiv preprint arXiv:1906.04733*, 2019b.

Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*, 2019c.

XuanLong Nguyen, Martin J Wainwright, and Michael I Jordan. Estimating divergence functionals and the likelihood ratio by convex risk minimization. *IEEE Transactions on Information Theory*, 56(11):5847–5861, 2010.

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

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

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

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *arXiv preprint arXiv:1703.01703*, 2017.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

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

Varun Tolani, Somil Bansal, Aleksandra Faust, and Claire Tomlin. Visual navigation among humans with optimal control as a supervisor. *IEEE Robotics and Automation Letters*, 6(2):2288–2295, 2021. doi: 10.1109/LRA.2021.3060638.

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

Mengjiao Yang, Bo Dai, Ofir Nachum, George Tucker, and Dale Schuurmans. Offline policy selection under uncertainty. *arXiv preprint arXiv:2012.06919*, 2020.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.