User-Oriented Robust Reinforcement Learning

Haoyi You, Beichen Yu, Haiming Jin*, Zhaoxing Yang, Jiahui Sun

Shanghai Jiao Tong University, Shanghai, China
{yuri-you, polarisybc, jinhaiming, yiannis, jhsun1997}@sjtu.edu.cn

Abstract

Recently, improving the robustness of policies across different environments attracts increasing attention in the reinforcement learning (RL) community. Existing robust RL methods mostly aim to achieve the max-min robustness by optimizing the policy's performance in the worst-case environment. However, in practice, a user that uses an RL policy may have different preferences over its performance across environments. Clearly, the aforementioned max-min robustness is oftentimes too conservative to satisfy user preference. Therefore, in this paper, we integrate user preference into policy learning in robust RL, and propose a novel *User-Oriented Robust RL (UOR-RL)* framework. Specifically, we define a new *User-Oriented Robustness (UOR)* metric for RL, which allocates different weights to the environments according to user preference and generalizes the max-min robustness metric. To optimize the UOR metric, we develop two different UOR-RL training algorithms for the scenarios with or without a priori known environment distribution, respectively. Theoretically, we prove that our UOR-RL training algorithms converge to near-optimal policies even with inaccurate or completely no knowledge about the environment distribution. Furthermore, we carry out extensive experimental evaluations in 6 MuJoCo tasks. The experimental results demonstrate that UOR-RL is comparable to the state-of-the-art baselines under the average-case and worst-case performance metrics, and more importantly establishes new state-of-the-art performance under the UOR metric.

Introduction

Recently, reinforcement learning (RL) raises a high level of interest in both the research community and the industry due to its satisfactory performance in a variety of decision-making tasks, such as playing computer games (Mnih et al. 2013), autonomous driving (Kiran et al. 2021), robotics (Kober, Bagnell, and Peters 2013). Among existing RL methods, model-free ones, such as DQN (Mnih et al. 2013), DDPG (Silver et al. 2014), PPO (Schulman et al. 2017), which typically train policies in simulated environments, have been widely studied. However, it is highly possible that there exist discrepancies between the training and execution environments, which could severely degrade the performance of the trained policies. Therefore, it is of significant importance to robustify RL policies across different environments.

Existing studies of RL robustness against environment discrepancies (Wiesemann, Kuhn, and Rustem 2013; Rajeswaran et al. 2016; Tessler, Efroni, and Mannor 2019; Curi, Bogunovic, and Krause 2021) mostly aim to achieve the max-min robustness by optimizing the performance of the policy in the worst-case environment. However, such max-min robustness could oftentimes be overly conservative, since it only concentrates on the performance of the policy in the worst case, regardless of its performance in any other case. As a matter of fact, it is usually extremely rare for the worst case (e.g., extreme weather conditions in autonomous driving, power failure incidence in robot control) to happen in many applications. Therefore, we should take the environment distribution into consideration and pay more attention to the environments with higher probabilities though they are not the worst.

Besides, a user that uses an RL policy for real-world tasks may have different preferences over her performance across environments. Furthermore, for the same decision-making task, the preferences of different users may vary, resulting that they are in favor of different policies. For instance, in computer games, some users prefer to attack others and take an aggressive policy, while others may prefer a defensive policy. Therefore, user preference is a crucial factor that should be considered in RL policy training, which is ignored by the max-min robustness.

Due to the significance of user preference and environment distribution, we design a new *User-Oriented Robustness (UOR)* metric, which integrates both user preference and environment distribution into the measurement of robustness. Specifically, the UOR metric allocates different weights to different environments, based on user preference, environment distribution, and relative policy performance. In fact, the max-min robustness is a special case of the UOR metric that the user prefers extremely conservative robustness, and thus, the UOR metric is a generalization of the max-min robustness. Hence, in this paper, we focus on optimizing the UOR metric during policy training, which can help obtain policies better aligned with user preference.

To optimize the UOR metric, we propose the *User-Oriented Robust RL (UOR-RL)* framework. One of the training algorithms of the UOR-RL framework, namely

*Corresponding author.

Distribution-Based UOR-RL (DB-UOR-RL), takes the environment distribution as input to help optimize the UOR metric. In real-world applications, however, the environment distribution may sometimes be unknown to the user. To tackle such case, we design another training algorithm, namely *Distribution-Free UOR-RL (DF-UOR-RL)*, which works even without any knowledge of the environment distribution. Both algorithms evaluate the UOR metric and use it to update the policy, while they differ in their approaches for UOR metric evaluation, because of the different prior knowledge of the environment distribution.

Theoretically, under several mild assumptions, we prove that UOR-RL guarantees the following series of desirable properties. For DB-UOR-RL, we prove that with $O(\frac{1}{\epsilon^d})$ computational complexity, where d denotes the dimension of the parameter that parameterizes the environment, the output policy of DB-UOR-RL is ϵ-suboptimal to the optimal policy under the UOR metric. Furthermore, even when DB-UOR-RL takes an inaccurate empirical environment distribution as input, we prove that, as long as the total variation distance between the empirical distribution and the accurate one is no larger than $O(\epsilon^d)$, the output policy of DB-UOR-RL is still guaranteed to be ϵ-suboptimal to the optimal policy. For DF-UOR-RL, though without any prior knowledge of the environment distribution, our proof shows that DF-UOR-RL could still generate an ϵ-suboptimal policy with $O(\frac{1}{\epsilon^{2d+4}})$ computational complexity.

The contributions of this paper are summarized as follows.

- We propose a user-oriented metric for robustness measurement, namely UOR, allocating different weights to different environments according to user preference. To the best of our knowledge, UOR is the first metric that integrates user preference into the measurement of robustness in RL.
- We design two UOR-RL training algorithms for the scenarios with or without a priori known environment distribution, respectively. Both algorithms take the UOR metric as the optimization objective so as to obtain policies better aligned with user preference.
- We prove a series of results, through rigorous theoretical analysis, showing that our UOR-RL training algorithms converge to near-optimal policies even with inaccurate or entirely no knowledge about the environment distribution.
- We conduct extensive experiments in 6 MuJoCo tasks. The experimental results demonstrate that UOR-RL is comparable to the state-of-the-art baselines under the average-case and worst-case performance metrics, and more importantly establishes new state-of-the-art performance under the UOR metric.

Problem Statement

Preliminary

We introduce parameterized Markov Decision Process (PMDP) (Rajeswaran et al. 2016) represented by a 6-tuple $(\mathcal{S}, \mathcal{A}, \gamma, \mathbb{S}_0, T, R)$, as it is the basis of the UOR-PMDP that will be defined in Section . $\mathcal{S}$ and $\mathcal{A}$ are respectively the set of states and actions. γ is the discount factor. $\mathbb{S}_0 \in \Delta(\mathcal{S})$ denotes the initial state distribution[1]. Furthermore, different from the traditional MDP, a PMDP's transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{P} \to \Delta(\mathcal{S})$ and reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{P} \to \Delta(\mathbb{R})$ take an additional environment parameter p as input, with $\mathbb{R}$ denoting the set of all real numbers. The environment parameter p is a random variable in range $\mathcal{P} \subset \mathbb{R}^d$, following a probability distribution $\mathbb{D}$. In PMDP, the parameter p is sampled at the beginning of each trajectory, and keeps constant during the trajectory. We consider the scenario where p is unknown to the user during execution, but our work can extend to the scenario with known p through regarding the environment parameter p as an additional dimension of state.

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a mapping from a state to a probability distribution over actions. Furthermore, the expected return $J(\pi, p)$ is defined as the expected discounted sum of rewards when the policy π is executed under the environment parameter p, i.e.,

$$J(\pi, p) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}, p) | \pi\right]. \tag{1}$$

User-Oriented Robust PMDP

Definition In this paper, we propose *User-Oriented Robust PMDP (UOR-PMDP)*, which is represented by an 8-tuple $(\mathcal{S}, \mathcal{A}, \gamma, \mathbb{S}_0, T, R, h, W)$. The first six items of UOR-PMDP are the same as those of PMDP. Furthermore, we introduce the last two items, namely the ranking function h and the preference function W, to formulate our new *User-Oriented Robustness (UOR)* metric that allocates more weights to the environment parameters where the policy has relatively worse performance, according to user preference.

As UOR requires assessing the relative performance of the policy under different environment parameters, we define the *ranking function* $h : \Pi \times \mathcal{P} \to [0, 1]$ as

$$h(\pi, p) = \int_{\mathcal{P}} \mathbb{D}(p) \cdot \mathbb{1}\left[J(\pi, p') \leq J(\pi, p)\right] \mathrm{d}p', \tag{2}$$

which represents the probability that the performance of policy π under an environment parameter p' sampled from $\mathbb{D}$ is worse than that under the environment parameter p.

To represent a user's preference, we define the *preference function* $W : [0, 1] \to \mathbb{R}_+ \cup \{0\}$ which assigns weights to environment parameters with different rankings. Specifically, given π, the weight assigned to environment parameter p is set as $W(h(\pi, p))$. Moreover, we require function W to be non-increasing, since UOR essentially puts more weights on environment parameters with lower rankings.

In practice, to make it more convenient for a user to specify her preference, we could let the preference function W belong to a family of functions parameterized by as few as only one parameter. For example, by setting the preference function W as

$$W(x) = (k + 1) \cdot (1 - x)^k, \tag{3}$$

a single robustness degree parameter $k \in \mathbb{R}_+$ suffices to completely characterize the user preference.

[1]We use $\Delta(\mathcal{X})$ to denote the set of all distributions over set $\mathcal{X}$.

In terms of the objective, the UOR-PMDP aims to maximize the *UOR metric* $\mathcal{E}$ defined as

$$\mathcal{E}(\pi) = \int_{\mathcal{P}} \mathbb{D}(p) \cdot J(\pi, p) \cdot W(h(\pi, p)) \mathrm{d}p, \tag{4}$$

which is the expectation of the weighted sum of $J(\pi, p)$ over the distribution $\mathbb{D}$. That is, the optimal policy π^* of the UOR-PMDP satisfies

$$\pi^* = \arg\max_{\pi \in \Pi} \mathcal{E}(\pi). \tag{5}$$

Properties In fact, our UOR metric generalizes the worst-case performance robustness metric $\min_{p \in \mathcal{P}} J(\pi, p)$ (Wiesemann, Kuhn, and Rustem 2013; Rajeswaran et al. 2016; Tessler, Efroni, and Mannor 2019; Curi, Bogunovic, and Krause 2021), and the average-case metric $\mathbb{E}_{p \sim \mathbb{D}}[J(\pi, p)]$ without robustness consideration (Da Silva, Konidaris, and Barto 2012; Tobin et al. 2017).

As W is non-increasing, it has the following two extreme cases. For one extreme case, W concentrates on zero. That is, $W(x) = \delta(x)$, where δ denotes the Dirac function, and consequently the UOR metric becomes

$$\mathcal{E}(\pi) = \int_{\mathcal{P}} \mathbb{D}(p) \cdot J(\pi, p) \cdot \delta(h(\pi, p)) \mathrm{d}p = \min_{p \in \mathcal{P}} J(\pi, p).$$

For the other extreme case, W is uniform in $[0, 1]$. That is, $W(x) \equiv 1$, and consequently the UOR metric becomes

$$\mathcal{E}(\pi) = \int_{\mathcal{P}} \mathbb{D}(p) \cdot J(\pi, p) \mathrm{d}p = \mathbb{E}_{p \sim \mathbb{D}}[J(\pi, p)].$$

Solutions for UOR-PMDP

In this section, we present two *User-Oriented Robust RL (UOR-RL)* algorithms to solve UOR-PMDP in the scenarios with and without a priori known environment parameter distribution, respectively.

Distribution-Based UOR-RL

Algorithm Design We consider the scenario that the distribution $\mathbb{D}$ is known before training, and propose the *Distribution-Based UOR-RL (DB-UOR-RL)* training algorithm in Algorithm 1 which makes use of the distribution $\mathbb{D}$ during the training period.

Firstly, Algorithm 1 randomly sets the initial policy π_{θ^0}, and chooses the upper bound δ of the block diameter (line 1). The criteria for setting δ will be discussed in detail in Section 15. Then, by calling the Set_Division algorithm, Algorithm 1 divides $\mathcal{P}$ into n blocks $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n$, whose diameters are less than δ (line 2). That is,

$$\forall\, \mathcal{P}_j \in \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n\}, \forall\, p_x, p_y \in \mathcal{P}_j, ||p_x - p_y||_2 \leq \delta.$$

Note that the number of blocks n is decided by how the Set_Division algorithm divides $\mathcal{P}$ based on δ. Because of space limit, we put our Set_Division algorithm in Appendix **??**. In fact, Algorithm 1 works with any Set_Division algorithm that could guarantee that the diameters of the divided blocks are upper bounded by δ. Then, for each block $\mathcal{P}_j$, Algorithm 1 arbitrarily chooses an element p_j from the block

Algorithm 1: DB-UOR-RL Algorithm

```
   // Initialization.
1  Initialize policy π_{θ^0} and block diameter upper bound δ;
2  {P_1, P_2, ··· , P_n} ← Set_Division(P, δ);
3  foreach Block P_j do
4      p_j ← Arbitrarily chosen element in P_j;
5      m_j ← ∫_{P_j} D(p)dp;
   // Policy Training.
6  foreach Iteration i = 0 to max-iterations do
7      foreach Block P_j do
8          Execute policy π_{θ^i} under p_j and evaluate the
             empirical expected return Ĵ(π_{θ^i}, p_j);
9      Sort the sequence {Ĵ(π_{θ^i}, p_j)} into an increasing
         sequence {Ĵ(π_{θ^i}, p_{α_j})};
       // Metric Calculation.
10     Initialize metric Ê(π_{θ^i}) ← 0 and M ← 0;
11     foreach Block P_j do
12         w_j ← ∫_M^{M+m_{α_j}} W(x)dx;
13         Ê(π_{θ^i}) ← Ê(π_{θ^i}) + w_j · Ĵ(π_{θ^i}, p_{α_j});
14         M ← M + m_{α_j};
       // Policy Update.
15     π_{θ^{i+1}} ← Policy_Update(π_{θ^i}, Ê(π_{θ^i}));
```

to represent it (line 4), and calculates the probability that an environment parameter falls into the block $\mathcal{P}_j$ (line 5).

Next, Algorithm 1 trains the policy (line 6 to 15). In each iteration i, it evaluates the performance $\hat{J}(\pi_{\theta^i}, p_j)$ of the policy π_{θ^i} under each p_j (line 8), and sorts the sequence $\{\hat{J}(\pi_{\theta^i}, p_j)\}$ into an increasing one $\{\hat{J}(\pi_{\theta^i}, p_{\alpha_j})\}$ (line 9). Then, Algorithm 1 calculates the metric $\hat{\mathcal{E}}(\pi_{\theta^i})$, which is an approximation of the UOR metric $\mathcal{E}(\pi_{\theta^i})$ (line 10 to 14). Specifically, it initializes the metric $\hat{\mathcal{E}}(\pi_{\theta^i})$ and the lower limit M of the integral as zero (line 10). For each block $\mathcal{P}_j$, it calculates the weight w_j allocated to this block based on the ranking α_j of block $\mathcal{P}_j$ in the sorted sequence $\{J(\pi_{\theta^i}, p_{\alpha_j})\}$ and preference function W (line 12), and updates the metric $\hat{\mathcal{E}}(\pi_{\theta^i})$ (line 13) and the lower limit of the integral (line 14). Finally, based on the metric $\hat{\mathcal{E}}(\pi_{\theta^i})$, Algorithm 1 updates the policy by applying a Policy_Update algorithm (line 15). Note that Policy_Update could be any policy gradient algorithm that updates the policy based on the metric $\hat{\mathcal{E}}(\pi_{\theta^i})$.

The above Algorithm 1 essentially uses integral discretization to calculate an approximate UOR metric $\hat{\mathcal{E}}$ which is used as the optimization objective of Policy_Update. To discretize the integral for calculating the UOR metric, Algorithm 1 divides the environment parameter range into blocks. Furthermore, to get the ranking function for weight allocation, Algorithm 1 sorts the blocks according to the evaluated performance on them.

Algorithm Analysis To analyze Algorithm 1, we make the following three mild assumptions.

Assumption 1. The transition function T and reward function R are continuous to the environment parameter p.

Assumption 2. The transition function T and reward function R are Lipschitz continuous to the state space $\mathcal{S}$ and action space $\mathcal{A}$ with constants $L_{T,\mathcal{S}}$, $L_{T,\mathcal{A}}$, $L_{R,\mathcal{S}}$, and $L_{R,\mathcal{A}}$, respectively.

Assumption 3. The policy π during the training process in Algorithm 1 is Lipschitz continuous with constant L_π.

Assumption 1 is natural, because the R and T functions characterize the environment which will usually not change abruptly as the environment parameter p changes. Furthermore, Assumptions 2 and 3 are commonly used in previous works (Curi, Berkenkamp, and Krause 2020; Curi, Bogunovic, and Krause 2021). Based on Assumptions 1-3, we prove the following Theorem 1, which demonstrates the existence of the diameter upper bound δ under which Algorithm 1 can converge to a near-optimal policy.

Theorem 1. *$\forall$ optimality requirement $\epsilon = 2\epsilon_0 > 0$, $\exists$ δ_0, such that as long as Policy_Update can learn an ϵ_0-suboptimal policy for metric $\hat{\mathcal{E}}$, by running Algorithm 1 with any diameter upper bound $\delta \leq \delta_0$, we can guarantee that the output policy $\hat{\pi}$ of Algorithm 1 satisfies*

$$\mathcal{E}(\hat{\pi}) \geq \mathcal{E}(\pi^*) - \epsilon. \tag{6}$$

Because of space limit, the proofs to all of the theorems and corollary in this paper are provided in the appendix.

Theorem 1 reveals that as long as the diameter upper bound δ is sufficiently small, the output policy $\hat{\pi}$ of Algorithm 1 will be close enough to the optimal policy π^*. However, as δ decreases, the number of the blocks output by Set_Division on line 2 of Algorithm 1 will increase, leading to an increased complexity of Algorithm 1. Thus, it is of great importance to have a quantitative relationship between ϵ and δ, which could help us better choose the upper bound δ based on the optimality requirement ϵ. To obtain the quantitative relationship between δ and ϵ, we introduce the following Assumption 4, which is stronger than Assumption 1.

Assumption 4. The transition function T and reward function R are Lipschitz continuous to the environment parameter p with constants $L_{T,p}$ and $L_{R,p}$.

Based on Assumptions 2-4, we prove Theorem 2.

Theorem 2. *$\forall$ optimality requirement $\epsilon = 2\epsilon_0 > 0$, $\exists$ $\delta_0 = \upsilon\epsilon = O(\epsilon)$, such that as long as Policy_Update can learn an ϵ_0-suboptimal policy for metric $\hat{\mathcal{E}}$, by running Algorithm 1 with any diameter upper bound $\delta \leq \delta_0$, we can guarantee that the output policy $\hat{\pi}$ of Algorithm 1 satisfies*

$$\mathcal{E}(\hat{\pi}) \geq \mathcal{E}(\pi^*) - \epsilon. \tag{7}$$

*Note that the constant υ depends on the Lipschitz constants in Assumptions 2-4, whose detailed form is presented in Equation (**??**) in Appendix **??**.*

Theorem 2 indicates that when Algorithm 1 chooses $\delta = \upsilon\epsilon = O(\epsilon)$, the number of divided blocks is at most $O(\frac{1}{\epsilon^d})$. Therefore, with such choice of δ, we can guarantee that the complexity of each iteration in Algorithm 1 is at most $O(\frac{1}{\epsilon^d})$.

In practice, the user may not know the accurate distribution $\mathbb{D}$, but only has access to a biased empirical distribution $\mathbb{D}^e$. In the following Theorem 3, we prove the theoretical guarantee of Algorithm 1 when it runs with $\mathbb{D}^e$.

Theorem 3. *Define the policy π^e such that*

$$\pi^e = \arg\max_{\pi} \mathcal{E}^e(\pi), \tag{8}$$

where $\mathcal{E}^e$ denotes the UOR metric under the UOR-PMDP with the empirical environment parameter distribution $\mathbb{D}^e$.

Then, $\forall$ given $\epsilon > 0$, $\exists$ $\kappa = O(\epsilon^d)$, such that as long as $\mathbb{D}$ and $\mathbb{D}^e$ satisfies the total variation distance $D_{TV}(\mathbb{D}, \mathbb{D}^e) \leq \kappa$, then we can guarantee that

$$\mathcal{E}(\pi^e) \geq \mathcal{E}(\pi^*) - \epsilon. \tag{9}$$

Based on Theorems 2 and 3, we have Corollary 1.

Corollary 1. *$\forall$ optimality requirement $\epsilon_1 = 3\epsilon > 0$, $\exists$ $\delta_0 = O(\epsilon)$ and $\kappa = O(\epsilon^d)$, such that as long as $D_{TV}(\mathbb{D}, \mathbb{D}^e) \leq \kappa$ and Policy_Update can learn an ϵ-suboptimal policy for metric $\hat{\mathcal{E}}$, by running Algorithm 1 with diameter upper bound $\delta \leq \delta_0$ and distribution $\mathbb{D}^e$, we can guarantee that the output policy $\hat{\pi}^e$ of Algorithm 1 satisfies*

$$\mathcal{E}(\hat{\pi}^e) \geq \mathcal{E}(\pi^*) - \epsilon_1. \tag{10}$$

Corollary 1 demonstrates that even running Algorithm 1 with the biased distribution $\mathbb{D}^e$, as long as $\mathbb{D}^e$ is close enough to $\mathbb{D}$, the output policy is still near-optimal.

Distribution-Free UOR-RL

Algorithm design In practice, it is likely that the distribution function $\mathbb{D}$ is unknown, making Algorithm 1 not applicable. Therefore, we propose the *Distribution-Free UOR-RL (DF-UOR-RL)* training algorithm in Algorithm 2 that trains a satisfactory policy even without any knowledge of the distribution function $\mathbb{D}$.

At the beginning, Algorithm 2 randomly sets the initial policy π_{θ^0} and n_1 empty clusters, and chooses the size of each cluster as n_2 (line 1). We will introduce in detail how to set the number of clusters n_1 and cluster size n_2 in Section 14. Then, Algorithm 2 begins to train the policy (line 2-14). In each iteration i, it samples n_2 trajectories for each cluster $\mathcal{C}_j$, by executing the current policy π_{θ^i} under the observed environment parameters (line 5-6), and evaluates the discounted reward of these trajectories (line 7). After that, Algorithm 2 evaluates the performance $\hat{J}_j$ of each cluster $\mathcal{C}_j$ by averaging the discounted reward of the trajectories in the cluster (line 8), and sorts the sequence $\{\hat{J}_j\}$ into an increasing one $\{\hat{J}_{\alpha_j}\}$ (line 9). Then, Algorithm 2 calculates the metric $\tilde{\mathcal{E}}(\pi_{\theta^i})$, which is an approximation of the UOR metric $\mathcal{E}(\pi_{\theta^i})$ (line 10-13). Initially, it sets the metric $\tilde{\mathcal{E}}(\pi_{\theta^i})$ as zero (line 10). Then, for each cluster $\mathcal{C}_j$, Algorithm 2 allocates the weight to cluster according to its ranking α_j and the preference function (line 12) and updates the $\tilde{\mathcal{E}}(\pi_{\theta^i})$ (line 13) based on the weight and performance of the cluster. Finally, Algorithm 2 obtains the $\tilde{\mathcal{E}}(\pi_{\theta^i})$ and uses it to update the policy (line 14).

Different from Algorithm 1, due to the lack of the knowledge of the distribution $\mathbb{D}$, Algorithm 2 observes the environment parameter rather than directly sample it according to $\mathbb{D}$. Given that it is of large bias to evaluate $J(\pi, p)$ from only

Algorithm 2: DF-UOR-RL Algorithm

```
   // Initialization
1  Initialize empty trajectory clusters C_1, C_2, ..., C_{n_1}, cluster
     size n_2, and policy π_{θ^0};
   // Policy Training
2  foreach Iteration i = 0 to max-iterations do
3      foreach j = 1 to n_1 do
4          foreach k = 1 to n_2 do
5              Observe environment parameter p_{j,k};
6              Execute π_{θ^i} under p_{j,k}, get trajectory ξ_{j,k}, and
                 C_j ← C_j ∪ {ξ_{j,k}};
7              Evaluate discounted reward Ĵ(ξ_{j,k}) of ξ_{j,k};
8          Ĵ_j ← (1/|C_j|) · Σ_{ξ_{j,k} ∈ C_j} Ĵ(ξ_{j,k});
9      Sort the sequence {Ĵ_j} into an increasing sequence
         {Ĵ_{α_j}};
       // Metric Calculation
10     Initialize metric Ẽ(π_{θ^i}) ← 0;
11     foreach j = 1 to n do
12         w_j ← ∫_{(j-1)/n}^{j/n} W(x)dx;
13         Ẽ(π_{θ^i}) ← Ẽ(π_{θ^i}) + w_j · Ĵ_{α_j};
       // Policy Update
14     π_{θ^{i+1}} ← Policy_Update(π_{θ^i}, Ẽ(π_{θ^i}));
```

one trajectory, Algorithm 2 averages the discounted rewards of n_2 trajectories. The clusters in Algorithm 2 have the same functionality as the blocks in Algorithm 1, and Algorithm 2 uses them to calculate an approximate UOR metric $\tilde{\mathcal{E}}(\pi_{\theta^i})$.

Algorithm Analysis To analyze Algorithm 2, we introduce an additional mild Assumption 5 on two properties of the environment parameters, including the difference between consecutively sampled environment parameters in line 5 of Algorithm 2, and the convergence rate of the posterior distribution of the environment parameter to the distribution $\mathbb{D}$. Because of space limit, we provide the detailed description of Assumption **??** in Appendix **??**.

Based on Assumptions 2-**??**, we have the theoretical guarantee of Algorithm 2 in the following Theorem 4.

Theorem 4. *$\forall$ optimality requirement $\epsilon = 2\epsilon_0 > 0$ and confidence ρ, $\exists\ n_1 = \Theta(\frac{-\ln \rho}{\epsilon^2}), n_2 = \Theta(\frac{-\ln \rho}{\epsilon^{2d+2}})$, such that as long as Policy_Update can learn an ϵ_0-suboptimal policy for metric $\tilde{\mathcal{E}}$, by running Algorithm 2 with trajectory cluster number larger than n_1 and cluster size larger than n_2, we can guarantee that the output policy $\tilde{\pi}$ of Algorithm 2 satisfies*

$$\mathcal{E}(\pi) \geq \mathcal{E}(\tilde{\pi}) - \epsilon \tag{11}$$

with confidence more than $1 - \rho$.

Theorem 4 provides guidelines for setting the cluster size n_1 and cluster number n_2 in Algorithm 2. In fact, as n_1 and n_2 increase, the performance evaluation of the cluster and the weight allocated to the cluster will be more accurate, both of which lead to a more accurate approximation of the UOR metric. However, the increase of either n_1 or n_2 leads to an increased complexity of each iteration of Algorithm 2. To deal with such trade-off, we could set n_1 and n_2 based on the lower bounds in Theorem 4, through which Algorithm 2 can guarantee both the optimality requirement ϵ and $O(\frac{\ln^2 \rho}{\epsilon^{2d+4}})$ complexity of each iteration.

Experiments

Baseline Methods

We compare UOR-RL with the following four baselines.
Domain Randomization-Uniform (DR-U). Domain Randomization (DR) (Tobin et al. 2017) is a method that randomly samples environment parameters in a domain, and optimizes the expected return over all collected trajectories. DR-U is an instance of DR, which samples environment parameters from a uniform distribution. **Domain Randomization-Gaussian (DR-G).** DR-G is another instance of DR, which samples environment parameters from a Gaussian distribution. **Ensemble Policy Optimization (EPOpt).** EPOpt (Rajeswaran et al. 2016) is a method that aims to find a robust policy through optimizing the performance of the worst few collected trajectories. **Monotonic Robust Policy Optimization (MRPO).** MRPO (Jiang et al. 2021) is the state-of-the-art robust RL method, which is based on EPOpt and jointly optimizes the performance of the policy in both the average and worst cases.

MuJoCo Tasks and Settings

We conduct experiments in six MuJoCo (Todorov, Erez, and Tassa 2012) tasks of version-0 based on Roboschool[2], including Walker 2d, Reacher, Hopper, HalfCheetah, Ant, and Humanoid. In each of the six tasks, by setting different environment parameters, we get a series of environments with the same optimization goal but different dynamics. Besides, we take 6 different random seeds for each task, and compare the performance of our algorithms to the baselines under these seeds. Because of space limit, we put the specific environment parameter settings and the random seed settings during the training process in Appendix **??**. For testing, in each environment, we sample 100 environment parameters following the Gaussian distributions truncated over the range given in Table 1.

Task	Parameters	Range $\mathcal{P}$	Distribution $\mathbb{D}$
Reacher	Body size	[0.008,0.05]	$\mathcal{N}(0.029, 0.007^2)$
	Body length	[0.1,0.13]	$\mathcal{N}(0.015, 0.005^2)$
Hopper	Density	[750,1250]	$\mathcal{N}(1000, 83.3^2)$
	Friction	[0.5, 1.1]	$\mathcal{N}(0.8, 0.1^2)$
Half Cheetah	Density	[750,1250]	$\mathcal{N}(1000, 83.3^2)$
	Friction	[0.5, 1.1]	$\mathcal{N}(0.8, 0.1^2)$
Humanoid	Density	[750,1250]	$\mathcal{N}(1000, 83.3^2)$
	Friction	[0.5, 1.1]	$\mathcal{N}(0.8, 0.1^2)$
Ant	Density	[750,1250]	$\mathcal{N}(1000, 83.3^2)$
	Friction	[0.5, 1.1]	$\mathcal{N}(0.8, 0.1^2)$
Walker 2d	Density	[750,1250]	$\mathcal{N}(1000, 83.3^2)$
	Friction	[0.5, 1.1]	$\mathcal{N}(0.8, 0.1^2)$

Table 1: Environment Parameter Settings for Testing.

[2]https://openai.com/blog/roboschool

In the experiments, we let the preference function W take the form as given by Equation (3), which uses a robustness degree k to represent user preference. Thus, we conduct experiments on various UOR metrics, including the average and max-min robustness, by choosing different k in Equation (3), and we use $\mathcal{E}_{k_0}$ to denote the UOR metric with $k = k_0$.

Considering that the state and action spaces of MuJoCo are high-dimensional and continuous, we choose to use deep neural networks to represent the policies of UOR-RL and the baseline methods, and use PPO (Schulman et al. 2017) to implement the policy updating process.

Experimental Results and Discussions

We compare UOR-RL with the baseline methods when $k \in \{0, 1, 21\}$. Specifically, when $k = 0$, $W(x) \equiv 1$, and thus $\mathcal{E}_0$ is equivalent to the expected return $\mathbb{E}_{p\sim\mathbb{D}}[J(\pi, p)]$ over the distribution $\mathbb{D}$; when $k = 21$, $\mathcal{E}_{21}$ approximates the expected return over the worst 10% trajectories, because more than 90% weight is allocated to them according to the preference function W; when $k = 1$, $\mathcal{E}_1$ represents the UOR metric between $\mathcal{E}_0$ and $\mathcal{E}_{21}$.

Table 2 shows the test results under the UOR metric $\mathcal{E}_1$. Among all algorithms, DB-UOR-RL performs the best, and it outperforms the four baselines in each environment. Such results indicate that DB-UOR-RL is effective under metric $\mathcal{E}_1$. At the same time, although the performance of DF-UOR-RL is not as good as that of DB-UOR-RL, it is better than those of the baselines. This shows that DF-UOR-RL could output competitive policies, even when the distribution of environment parameters is unknown.

Table 3 shows the test result under the average return of all trajectories. In most environments, DR-G achieves the best performance among the baselines under such average-case metric, because it directly takes this average-case metric as its optimization objective. From Table 3, we could observe that the performance of DB-UOR-RL and DF-UOR-RL is close to or better than that of DR-G in most environments. Such observation indicates that UOR-RL can also yield acceptable results, when robustness is not considered.

Table 4 shows the test result under the average return of the worst 10% trajectories. From the table, both DB-UOR-RL and DF-UOR-RL perform no worse than the best baselines in most environments, which shows that UOR-RL also yields sufficiently good performance in terms of the traditional robustness evaluation criteria.

Apart from Tables 2-4, we also visualize the performance of UOR-RL and the baselines in the ranges of environment parameters given by Table 1 by plotting heat maps. Because of space limit, we only show the heat maps of the Half Cheetah and Hopper tasks with $k = 1$ in Figures 1 and 2, respectively. In these two figures, a darker color means a better performance. We could observe that both DB-UOR-RL and DF-UOR-RL are darker in color than baselines in most subranges, which supports the superiority of UOR-RL for most environment parameters. Moreover, we place the heat maps of the other four tasks in Appendix **??**.

To show the effect of the robustness degree parameter k on the performance of UOR-RL, we carry out experiments with four robustness degree parameters $k \in \{0, 1, 5, 21\}$ in Half Cheetah under the same environment parameters. The results are shown in Figures 3 and 4. To plot these two figures, we sort the collected trajectories by return into an increasing order, divide the trajectories under such order into 10 equal-size groups, calculate the *average return of the trajectories (ART)* in each group, and compute the normalized differences between the ARTs that correspond to each consecutive pair of k's in $\{0, 1, 5, 21\}$. We could observe that every curve in these two figures shows a decreasing trend as the group index increases. Such observation indicates that, as k increases, both DB-UOR-RL and DF-UOR-RL pay more attention to the trajectories that perform poorer, and thus the trained policies become more robust.

Additionally, we plot the training curves of the baselines and UOR-RL, and place them in Appendix **??**.

Related Work

Robust RL (Iyengar 2005; Nilim and El Ghaoui 2005; Wiesemann, Kuhn, and Rustem 2013) aims to optimize policies under the worst-case environment, traditionally by the zero-sum game formulation (Littman 1994; Littman and Szepesvari 1996). Several recent works focus on finite or linear MDPs, and propose robust RL algorithms with theoretical guarantees (Derman, Geist, and Mannor 2021; Wang and Zou 2021; Badrinath and Kalathil 2021; Zhang et al. 2021b; Grand-Clément and Kroer 2020; Kallus and Uehara 2020). However, real-world applications are usually with continuous state and action spaces, as well as complex non-linear dynamics (Kumar et al. 2020; Zhang, Hu, and Basar 2020).

A line of deep robust RL works robustify policies against different factors that generate the worst-case environment, such as agents' observations (Zhang et al. 2020, 2021a; Oikarinen, Weng, and Daniel 2020), agents' actions (Tessler, Efroni, and Mannor 2019; Kamalaruban et al. 2020), transition function (Mankowitz et al. 2020; Viano et al. 2021; Chen, Du, and Jamieson 2021), and reward function (Wang, Liu, and Li 2020). Another line of recent works (Kumar et al. 2020; Tobin et al. 2017; Jiang et al. 2021; Igl et al. 2019; Cobbe et al. 2019) aim to improve the average performance over all possible environments. Considering only the worst or average case may cause the policy to be overly conservative or aggressive, and limit (Zhang et al. 2020, 2021a; Oikarinen, Weng, and Daniel 2020; Tessler, Efroni, and Mannor 2019; Kamalaruban et al. 2020; Mankowitz et al. 2020; Viano et al. 2021; Chen, Du, and Jamieson 2021; Wang, Liu, and Li 2020; Kumar et al. 2020; Tobin et al. 2017; Jiang et al. 2021; Igl et al. 2019; Cobbe et al. 2019) for boarder applications. Hence, some researches study to use other cases to characterize the robustness of the policy. (Chow et al. 2015) optimizes the policy performance on α-percentile worst-case environments; (Derman et al. 2018) considers the robustness with a given environment distribution; (Xu and Mannor 2010; Yu and Xu 2015) aim to improve the policy performance on the worst distribution in an environment distribution set.

Each of (Zhang et al. 2020, 2021a; Oikarinen, Weng, and Daniel 2020; Tessler, Efroni, and Mannor 2019; Kamalaruban et al. 2020; Mankowitz et al. 2020; Viano et al. 2021; Chen, Du, and Jamieson 2021; Wang, Liu, and Li 2020; Kumar et al. 2020; Tobin et al. 2017; Jiang et al. 2021; Igl et al.

Algorithm	Reacher	Hopper	Half Cheetah	Humanoid	Ant	Walker 2d
DR-U	10.92 ± 1.90	1465 ± 447.78	2375 ± 70.36	54.73 ± 3.73	2247 ± 568.53	1022 ± 368.74
DR-G	10.86 ± 2.34	1557 ± 346.97	2382 ± 64.94	56.43 ± 12.77	2258 ± 538.72	838 ± 211.86
EPOpt	12.57 ± 0.59	1360 ± 631.32	2450 ± 84.62	65.37 ± 7.53	2307 ± 192.94	1058 ± 432.98
MRPO	11.88 ± 2.27	1578 ± 333.85	2665 ± 147.89	89.48 ± 5.72	2303 ± 560.06	1368 ± 519.04
DB-UOR-RL	$\mathbf{14.38 \pm 0.86}$	$\mathbf{1942 \pm 348.97}$	$\mathbf{3067 \pm 169.31}$	$\mathbf{95.57 \pm 17.64}$	3212 ± 187.34	$\mathbf{1373 \pm 567.17}$
DF-UOR-RL	13.63 ± 0.59	1772 ± 313.59	2760 ± 238.41	93.97 ± 20.36	$\mathbf{3213 \pm 399.13}$	1268 ± 348.03

Table 2: Test results ($k = 1$). Each value denotes the mean and std of $\mathcal{E}_1$ over 6 seeds.

Algorithm	Reacher	Hopper	Half Cheetah	Humanoid	Ant	Walker 2d
DR-U	16.47 ± 1.91	1768 ± 471.06	2428 ± 89.76	71.32 ± 5.10	2272 ± 572.20	1159 ± 369.08
DR-G	19.28 ± 1.21	1993 ± 159.17	2420 ± 61.21	72.38 ± 15.77	2475 ± 86.20	1145 ± 503.29
EPOpt	17.83 ± 0.49	1455 ± 675.24	2555 ± 142.51	82.30 ± 7.81	2328 ± 203.12	1160 ± 418.16
MRPO	17.55 ± 1.43	1940 ± 193.08	2652 ± 125.13	99.83 ± 6.89	2295 ± 574.06	1268 ± 548.13
DB-UOR-RL	$\mathbf{19.97 \pm 1.11}$	$\mathbf{2007 \pm 379.03}$	$\mathbf{2910 \pm 135.06}$	$\mathbf{104.53 \pm 18.68}$	3393 ± 240.89	$\mathbf{1403 \pm 234.75}$
DF-UOR-RL	19.53 ± 0.65	1805 ± 246.96	2698 ± 293.90	98.28 ± 11.51	$\mathbf{3410 \pm 106.21}$	1208 ± 200.76

Table 3: Test results ($k = 0$). Each value denotes the mean and std of the average return of all trajectories over 6 seeds.

Algorithm	Reacher	Hopper	Half Cheetah	Humanoid	Ant	Walker 2d
DR-U	-0.81 ± 1.86	442 ± 136.28	1895 ± 112.92	8.00 ± 7.19	2163 ± 564.79	461 ± 369.96
DR-G	-1.90 ± 3.73	593 ± 367.47	1957 ± 225.45	8.55 ± 7.37	2133 ± 531.10	380 ± 388.40
EPOpt	-0.14 ± 1.79	664 ± 495.87	2132 ± 418.40	10.47 ± 4.20	2192 ± 153.55	360 ± 222.00
MRPO	-3.66 ± 7.54	678 ± 640.70	$\mathbf{2523 \pm 195.72}$	34.92 ± 5.70	2187 ± 555.22	515 ± 419.33
DB-UOR-RL	$\mathbf{3.82 \pm 0.85}$	$\mathbf{824 \pm 583.20}$	2497 ± 404.51	$\mathbf{35.45 \pm 7.97}$	3132 ± 119.07	$\mathbf{624 \pm 283.99}$
DF-UOR-RL	1.75 ± 1.66	452 ± 641.92	2383 ± 581.09	33.18 ± 44.76	$\mathbf{3265 \pm 196.34}$	497 ± 278.93

Table 4: Test results($k = 21$). Each value denotes the mean and std of the average return of worst 10% trajectories over 6 seeds.

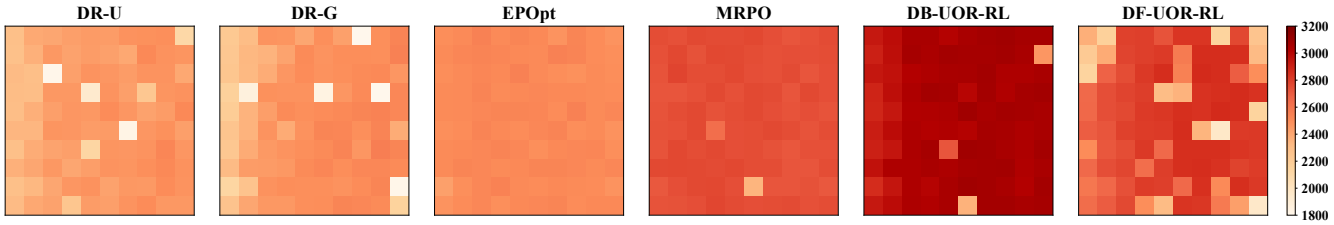

Figure 1: Heat map of $\mathcal{E}_1$ in sub-ranges (Half Cheetah). The x-axis and y-axis denote friction and density, respectively. The ranges of these two parameters are chosen as in Table 1, and are evenly divided into 10 sub-ranges.

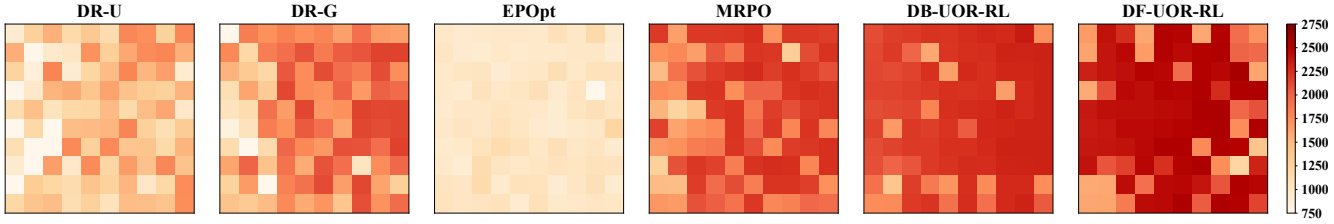

Figure 2: Heat map of $\mathcal{E}_1$ in sub-ranges (Hopper). The x-axis and y-axis denote friction and density, respectively. The ranges of these two parameters are chosen as in Table 1, and are evenly divided into 10 sub-ranges.

2019; Cobbe et al. 2019; Chow et al. 2015; Derman et al. 2018; Xu and Mannor 2010; Yu and Xu 2015) optimizes a specific type of robustness, and is only suitable to a specific preference to robustness (e.g. methods focusing on worst case suit the most conservative preference to robustness). However, user preference varies in different scenarios, and an RL method that optimizes a specific type of robustness will be no more suitable when user preference changes. In real applications, it is significant to take user preference into consideration and design a general framework suitable to

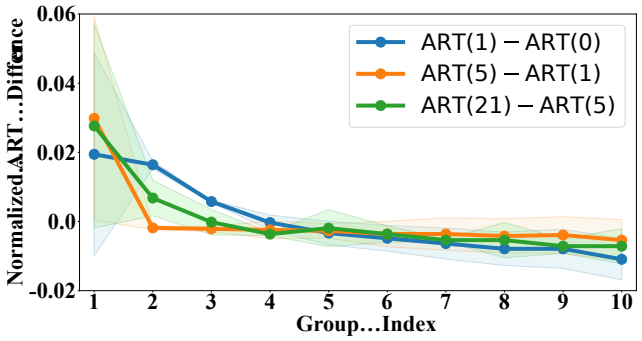

Figure 3: Normalized ART difference for each sorted group (DB-UOR-RL). ART(k_0) denotes the ART with $k = k_0$.

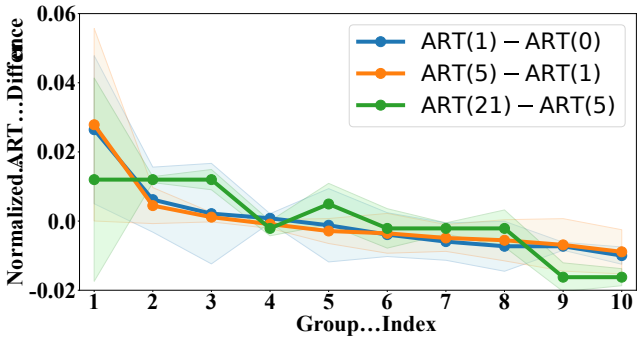

Figure 4: Normalized ART difference for each sorted group (DF-UOR-RL). ART(k_0) denotes the ART with $k = k_0$.

various types of robustness. Therefore, we design UOR-RL as a general framework, which can be applied to satisfy a variety of preference to robustness. As far as we know, UOR is the first RL framework to take user preference into consideration.

Conclusion

In this paper, we propose the UOR metric, which integrates user preference into the measurement of robustness. Aiming at optimizing such metric, we design two UOR-RL training algorithms, which work in the scenarios with or without a priori known environment distribution, respectively. Theoretically, we prove that the output policies of the UOR-RL training algorithms, in the scenarios with accurate, inaccurate or even completely no knowledge of the environment distribution, are all ϵ-suboptimal to the optimal policy. Also, we conduct extensive experiments in 6 MuJoCo tasks, and the results validate that UOR-RL is comparable to the state-of-the-art baselines under traditional metrics and establishes new state-of-the-art performance under the UOR metric.

Acknowledgements

This work was supported by NSF China (No. U21A20519, U20A20181, 61902244).

References

Badrinath, K. P.; and Kalathil, D. 2021. Robust reinforcement learning using least squares policy iteration with provable performance guarantees. In *International Conference on Machine Learning*, 511–520. PMLR.

Chen, Y.; Du, S. S.; and Jamieson, K. 2021. Improved corruption robust algorithms for episodic reinforcement learning. In *International Conference on Machine Learning*. PMLR.

Chow, Y.; Tamar, A.; Mannor, S.; and Pavone, M. 2015. Risk-sensitive and robust decision-making: a cvar optimization approach. *Advances in neural information processing systems*, 28.

Cobbe, K.; Klimov, O.; Hesse, C.; Kim, T.; and Schulman, J. 2019. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, 1282–1289. PMLR.

Curi, S.; Berkenkamp, F.; and Krause, A. 2020. Efficient model-based reinforcement learning through optimistic policy search and planning. *arXiv preprint arXiv:2006.08684*.

Curi, S.; Bogunovic, I.; and Krause, A. 2021. Combining Pessimism with Optimism for Robust and Efficient Model-Based Deep Reinforcement Learning. *arXiv preprint arXiv:2103.10369*.

Da Silva, B.; Konidaris, G.; and Barto, A. 2012. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*.

Derman, E.; Geist, M.; and Mannor, S. 2021. Twice regularized MDPs and the equivalence between robustness and regularization. In *Advances in Neural Information Processing Systems*.

Derman, E.; Mankowitz, D. J.; Mann, T. A.; and Mannor, S. 2018. Soft-robust actor-critic policy-gradient. *arXiv preprint arXiv:1803.04848*.

Grand-Clément, J.; and Kroer, C. 2020. First-Order Methods for Wasserstein Distributionally Robust MDP. *arXiv preprint arXiv:2009.06790*.

Igl, M.; Ciosek, K.; Li, Y.; Tschiatschek, S.; Zhang, C.; Devlin, S.; and Hofmann, K. 2019. Generalization in reinforcement learning with selective noise injection and information bottleneck. *arXiv preprint arXiv:1910.12911*.

Iyengar, G. N. 2005. Robust dynamic programming. *Mathematics of Operations Research*, 30(2): 257–280.

Jiang, Y.; Li, C.; Dai, W.; Zou, J.; and Xiong, H. 2021. Monotonic robust policy optimization with model discrepancy. In *International Conference on Machine Learning*, 4951–4960. PMLR.

Kallus, N.; and Uehara, M. 2020. Double reinforcement learning for efficient and robust off-policy evaluation. In *International Conference on Machine Learning*, 5078–5088. PMLR.

Kamalaruban, P.; Huang, Y.-T.; Hsieh, Y.-P.; Rolland, P.; Shi, C.; and Cevher, V. 2020. Robust reinforcement learning via adversarial training with langevin dynamics. *arXiv preprint arXiv:2002.06063*.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Sallab, A. A. A.; Yogamani, S.; and Pérez, P. 2021. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems*.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274.

Kumar, S.; Kumar, A.; Levine, S.; and Finn, C. 2020. One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL. In *Advances in Neural Information Processing Systems*, 8198–8210.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, 157–163.

Littman, M. L.; and Szepesvari, C. 1996. A Generalized Reinforcement-Learning Model: Convergence and Applications. In *International Conference on Machine Learning*, 310–318.

Mankowitz, D. J.; Levine, N.; Jeong, R.; Shi, Y.; Kay, J.; Abdolmaleki, A.; Springenberg, J. T.; Mann, T.; Hester, T.; and Riedmiller, M. 2020. Robust reinforcement learning for continuous control with model misspecification. In *International Conference on Learning Representations*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nilim, A.; and El Ghaoui, L. 2005. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5): 780–798.

Oikarinen, T.; Weng, T.-W.; and Daniel, L. 2020. Robust deep reinforcement learning through adversarial loss. *arXiv preprint arXiv:2008.01976*.

Rajeswaran, A.; Ghotra, S.; Ravindran, B.; and Levine, S. 2016. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*.

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

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 387–395. PMLR.

Tessler, C.; Efroni, Y.; and Mannor, S. 2019. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, 6215–6224. PMLR.

Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; and Abbeel, P. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems*, 23–30. IEEE.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Viano, L.; Huang, Y.-T.; Kamalaruban, P.; Weller, A.; and Cevher, V. 2021. Robust Inverse Reinforcement Learning under Transition Dynamics Mismatch. In *Advances in Neural Information Processing Systems*.

Wang, J.; Liu, Y.; and Li, B. 2020. Reinforcement learning with perturbed rewards. In *AAAI Conference on Artificial Intelligence*, 6202–6209.

Wang, Y.; and Zou, S. 2021. Online Robust Reinforcement Learning with Model Uncertainty. In *Advances in Neural Information Processing Systems*.

Wiesemann, W.; Kuhn, D.; and Rustem, B. 2013. Robust Markov decision processes. *Mathematics of Operations Research*, 38(1): 153–183.

Xu, H.; and Mannor, S. 2010. Distributionally Robust Markov Decision Processes. In *NIPS*, 2505–2513.

Yu, P.; and Xu, H. 2015. Distributionally robust counterpart in Markov decision processes. *IEEE Transactions on Automatic Control*, 61(9): 2538–2543.

Zhang, H.; Chen, H.; Boning, D.; and Hsieh, C.-J. 2021a. Robust reinforcement learning on state observations with learned optimal adversary. *arXiv preprint arXiv:2101.08452*.

Zhang, H.; Chen, H.; Xiao, C.; Li, B.; Liu, M.; Boning, D.; and Hsieh, C.-J. 2020. Robust deep reinforcement learning against adversarial perturbations on state observations. *arXiv preprint arXiv:2003.08938*.

Zhang, K.; Hu, B.; and Basar, T. 2020. On the stability and convergence of robust adversarial reinforcement learning: A case study on linear quadratic systems. In *Advances in Neural Information Processing Systems*.

Zhang, X.; Chen, Y.; Zhu, X.; and Sun, W. 2021b. Robust policy gradient against strong data corruption. *arXiv preprint arXiv:2102.05800*.