Robust Reinforcement Learning via Progressive Task Sequence

Yike Li, **Yunzhe Tian**, **Endong Tong**[*], **Wenjia Niu**[*] and **Jiqiang Liu**
Beijing Key Laboratory of Security and Privacy in Intelligent Transportation,
Beijing Jiaotong University
{yikeli, tianyunzhe, edtong, niuwj, jqliu}@bjtu.edu.cn

Abstract

Robust reinforcement learning (RL) has been a challenging problem due to the gap between simulation and the real world. Existing efforts typically address the robust RL problem by solving a max-min problem. The main idea is to maximize the cumulative reward under the worst-possible perturbations. However, the worst-case optimization either leads to overly conservative solutions or unstable training process, which further affects the policy robustness and generalization performance. In this paper, we tackle this problem from both formulation definition and algorithm design. First, we formulate the robust RL as a max-expectation optimization problem, where the goal is to find an optimal policy under both the worst cases and the non-worst cases. Then, we propose a novel framework DRRL to solve the max-expectation optimization. Given our definition of the feasible tasks, a task generation and sequencing mechanism is introduced to dynamically output tasks at appropriate difficulty level for the current policy. With these progressive tasks, DRRL realizes dynamic multi-task learning to improve the policy robustness and the training stability. Finally, extensive experiments demonstrate that the proposed method exhibits significant performance on the unmanned CarRacing game and multiple high-dimensional MuJoCo environments.

1 Introduction

Reinforcement learning (RL) has recently achieved beyond-human performance on a wide range of decision-making tasks like games [Silver *et al.*, 2017], continuous control [Lyu *et al.*, 2022] and robotics [Dalal *et al.*, 2021]. In order to reduce risk of costly failure, RL agents are typically trained in a high-precision simulator. However, due to the inevitable reality gap, these algorithms are often overfitted to simulation environments and fail to generalize to the real world. In some safety-sensitive scenarios such as autonomous driving and medical treatment, using a bad policy can be costly and

[*]Corresponding author.

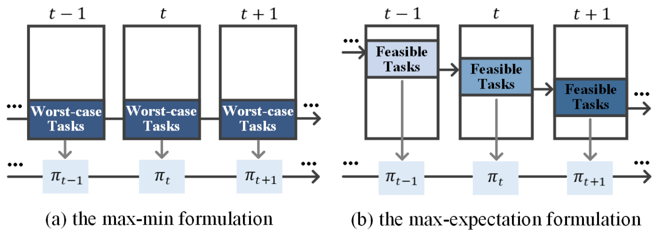


Figure 1: The max-min formulation updates the current policy only with the worst-case perturbations, while the max-expectation formulation selects a set of perturbations with appropriate difficulties (i.e., the feasible tasks) to update the current policy.

dangerous [Thomas and Brunskill, 2016]. To this end, learning policies that are robust to environmental shifts or mismatched configurations are becoming increasingly important.

Recent efforts in robust RL focus on performing perturbations at training time to achieve robustness. There are three main branches of the prior works that diverse in how the perturbations are introduced: 1) the Robust Markov Decision Process (RMDP)-based methods [Nilim and El Ghaoui, 2005; Iyengar, 2005; Wiesemann *et al.*, 2013], where the perturbations are modeled as an uncertainty set to be introduced over different elements of the MDP tuple; 2) the Domain Randomization (DR) [Jakobi, 1997; Peng *et al.*, 2018], a method that manipulates environmental perturbations by randomizing a set of parameters, which requires a delicate definition of the aspects that the agent should be robust to; and 3) the adversarial methods [Pinto *et al.*, 2017; Vinitsky *et al.*, 2020; Tessler *et al.*, 2019; Dennis *et al.*, 2020; Jiang *et al.*, 2021], which introduces adversarial agents to automatically generate perturbations by applying external forces to the protagonist agent, formulating the robust RL as a max-min game. More recently, intrinsic motivation methods are proposed to encourage agents to learn robust policies, using the intrinsically motivated reward signal from evolutionary techniques [Song and Schneider, 2022; Niekum, 2010], distance constrains [Abdullah *et al.*, 2019] and so on.

Despite the impressive progress, existing studies typically tackle the robust RL problem under the max-min settings, i.e., to maximize the return of the agent under worst-case perturbations. However, the max-min formulation often leads to overly conservative solutions and unstable training process

[Zhang *et al.*, 2020; Yu *et al.*, 2021]. In order to prevent the performance under the worst cases from being too bad, the max-min methods scarify the reward under non-worst cases. Moreover, as existing robust RL framework lacks the effective perturbation organization and programming during training, the worst-case perturbations may break the training stability and even bring a failure of policy learning. The same occurs in the field of classification problem, where the optimization technique is used to find the optimal hyperplane with the largest margin, i.e., to maximize the margin of the closest pair of classes. However, Gao et. al [Gao and Zhou, 2013] disclosed that maximizing the minimum margin does not necessarily lead to better performance, and instead, it is more crucial to optimize the margin distribution. Thus, the optimal margin distribution machine [Zhang and Zhou, 2018] is proposed, which is proved to have better generalization performance than large margin-based methods.

Inspired by the above works, an important insight brought by this work is to formulate the robust RL as a max-expectation problem, where the agent is trained to maximize the expected reward simultaneously under as many perturbations as possible including the worst cases and the non-worst cases. We argue that for the max-min methods, the conservative solutions are caused by the limited knowledge space for the agent, which is only composed of the environments under worst-case perturbations. By introducing the "expectation", we aim to extend the knowledge space for the agent to explore the optimal policy, and thus to avoid conservative policies. Figure 1 shows the difference between the two robust formulation. However, the max-expectation formulation inevitably makes finding of the robust policy a more complex optimization problem. This presents us with a new question: how to organize the learning process to effectively solve the max-expectation optimization problem?

To solve the question, in this paper, we propose a novel approach, i.e., **D**ynamic **R**obust **RL** named **DRRL**, where we model the robust RL problem as a progressive multi-task learning problem. The core of DRRL is an effective perturbation (i.e., task) programming. Through the genetic algorithm (GA) based task generation and task evaluation, a group of feasible tasks are generated iteratively, which satisfies the appropriate difficulty level for the current policy. With the progressive task sequence, DRRL improves the policy robustness and the training stability via progressive task-oriented learning. In summary, our contributions are highlighted as follows:

- We make the first attempt to formulate the robust RL as a max-expectation optimization problem, which extends the agent's knowledge space about the RL task, and thus to avoid conservative solutions.
- We propose a novel approach DRRL to effectively solve the max-expectation optimization problem, which introduces progressive multi-task learning to improve the policy robustness and training stability.
- We conduct extensive experiments on high-dimensional continuous environments (i.e., Hopper, HalfCheetah, Walker2d and CarRacing) to validate the effectiveness of DRRL compared with state-of-the-art methods.

2 Problem Formulation

Reinforcement learning is typically formalized as Markov Decision Processes (MDPs): $\mathcal{M} = (\mathcal{S}; \mathcal{A}; \mathcal{P}; r; \gamma; s_0)$, where $\mathcal{S}$ is a set of states, and $\mathcal{A}$ is a set of actions, $\mathcal{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state transition probability describing the system's dynamics, r: $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function measuring the agent's performance, $\gamma \in [0, 1)$ is the discount factor, and s_0 is the initial state distribution. The learning goal for a given MDP is to obtain an optimal policy $\pi^* : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ that maximizes the expected cumulative discounted reward:

$$\pi^* = \underset{\pi}{argmax}\, R(\pi|\mathcal{P}) = \underset{\pi}{argmax}\, \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_t, a_t)|\mathcal{P}]. \quad (1)$$

It is worth noting that the cumulative discounted reward is conditioned on the transition probability $\mathcal{P}$, which is always fixed in ordinary RL [Chen *et al.*, 2021]. Motivated by real-world applications, a good RL policy should generalize well across a range of different transition probabilities. That is, a good RL policy should be robust to variations in dynamics. Taking OpenAI gym's CarRacing scenario [Brockman *et al.*, 2016] as an example, we aim to learn a policy for autonomous vehicles that can run not only on high-grade roads (corresponding to one transition function) but also on icy roads (corresponding to another transition function).

Thus, instead of learning an optimal policy on a fixed transition probability (as shown in Eq. 1), the goal of robust RL is to learn a policy that is robust to a group of feasible transition probabilities. Generally, robust RL is achieved by solving the following max-min optimization problem [Wiesemann *et al.*, 2013; Kuang *et al.*, 2022]

$$\pi^* = \underset{\pi}{argmax}[\underset{\mathcal{P}\in\mathcal{P}(\cdot)}{min}\, \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_t, a_t)|\mathcal{P}]]. \quad (2)$$

$\mathcal{P}(\cdot)$ is the set of possible transition probabilities. In this case, the objective of robust RL is to learn best-case policies under worst-case transitions. However, in the worst-cases, the agent is trained to always take conservative actions to avoid performing catastrophic actions [Lecarpentier and Rachelson, 2019]. Thus, the expected rewards in some best cases are inevitably reduced, which is obviously not the optimal solution for real-world applications.

In our work, instead of the max-min game, we formulate the robust RL problem as an expectation optimization, where the learning goal is to maximize the expected reward among all the possible transition probabilities:

$$\begin{aligned} \pi^* &= \underset{\pi}{argmax}\, R(\pi|\mathcal{P}(\cdot)) \\ &= \underset{\pi}{argmax} \underset{\mathcal{P}\in\mathcal{P}(\cdot)}{\mathbb{E}}[\mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_t, a_t)|\mathcal{P}]]. \end{aligned} \quad (3)$$

To solve the expectation optimization problem in Eq. 3, we tackle the robust RL issue as a multi-task learning problem via modeling transition probabilities as the tasks, and propose a progressive robust policy optimization framework DRRL, which is detailed in the following section.

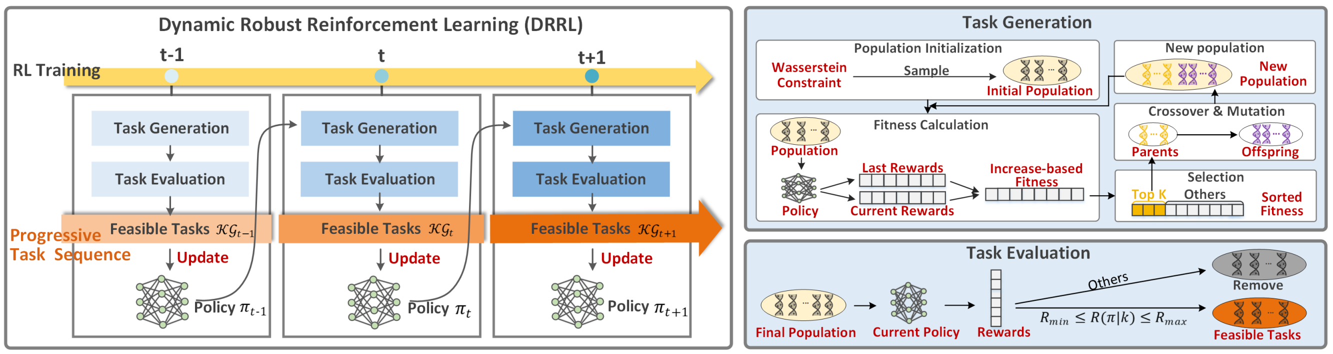


Figure 2: Overview of Dynamic Robust Reinforcement Learning (DRRL).

3 Methodology

3.1 DRRL Overview

For our defined robust RL, we have the following two assumptions:

- Modeling errors of transition probability can be viewed as a set of RL tasks, in which each task corresponds to a specific variant of $\mathcal{P}_0$. Here, $\mathcal{P}_0$ represents the original transition probability in simulation.
- If the trained optimal policy is robust to a sequence of well-designed RL tasks, then it will have high probability to be robust to modeling errors or mismatch between simulation and real world.

Based on the above assumptions, our proposed DRRL tackles robust RL as a dynamic multi-task learning process on progressive task sequence.

Definition 1 (Dynamic Progressive Tasks)**.** *Dynamic progressive tasks are a sequence of task groups over a maximum of T training steps: $\mathcal{DCT} = \{\mathcal{KG}_1|\pi_0, \mathcal{KG}_2|\pi_1, ..., \mathcal{KG}_t|\pi_{t-1}, ..., \mathcal{KG}_T|\pi_{T-1}\}$, in which $\mathcal{KG}_t|\pi_{t-1} = \{k_t^1, k_t^2, ..., k_t^i, ...\}$. Each single task k_t^i corresponds to a specific variant of the original transition probability $\mathcal{P}_0$:*

$$k_t^i \sim \omega_t^i \cdot \mathcal{P}_0 \quad \forall\, task\ k_t^i \in KG_t|\pi_{t-1}, \tag{4}$$

such that the following three conditions are satisfied.

- *The training difficulty levels (DLs) of tasks in the same group are much the same, i.e., $DL(k_t^1) \approx DL(k_t^2) \cdots$.*
- *The tasks of $\mathcal{KG}_t$ are generated dynamically according to the current policy π_{t-1}, i.e., $\mathcal{KG}_t \sim tasks(\pi_{t-1})$.*
- *The training difficulty levels of task groups gradually increase, i.e., $DL(\mathcal{KG}_t) < DL(\mathcal{KG}_{t+1})$.*

Thus, the progressive multi-task learning trains a robust policy with easier tasks first, and then gradually increases the difficulty level until all the target tasks are traversed.

As shown in Figure 2, we propose Dynamic Robust Reinforcement Learning (DRRL). At the iteration t of training, tasks are firstly generated by the GA-based task generation module, where the fitness function is defined as the increase of reward. Then, according to the tasks' expected rewards, the task evaluation module will filter out feasible tasks $\mathcal{KG}_t$, which satisfy the appropriate difficulty level for the current policy π_{t-1}. Based on $\mathcal{KG}_t$, policy π_{t-1} will be trained and updated to π_t. In this way, the feasible tasks at different iterations dynamically form an easy-to-hard tasks sequence. Finally, through task (or perturbation) organization and programming, we can accelerate the policy convergence and improve the policy generalization performance.

3.2 Task Generation and Sequencing Mechanism

Existing perturbation-based robust RL approaches perform perturbations on state observations, actions, or transition probabilities in the learning process. Unfortunately, most methods lack guarantees of the perturbation extents of the transition functions. Hence, as a prerequisite, we adapt the Wasserstein constraint [Abdullah *et al.*, 2019; He *et al.*, 2022] to restrict the admissible perturbed transition probabilities within a Wasserstein ball centered at the original transition probability $\mathcal{P}_0$ of the RL simulator:

$$\mathcal{W}_\epsilon(\mathcal{P}_0) = \{\mathcal{P}(\cdot) : \mathbb{E}_{(s,a)\sim\mathcal{P}}[\mathcal{W}_2^2(\mathcal{P}(\cdot|s,a), \mathcal{P}_0(\cdot|s,a))] \leq \epsilon\}. \tag{5}$$

Here, $\epsilon \in \mathbb{R}_+$ is a hyper-parameter used to specify the maximum Wasserstein bound. In our method, only the transition probabilities that satisfy the Wasserstein constraint (i.e., $\mathcal{P} \in \mathcal{W}_\epsilon(\mathcal{P}_0)$) will be chosen for robust RL training.

Task Generation

Sampling tasks from $\mathcal{W}_\epsilon(\mathcal{P}_0)$ directly, and filtering out feasible tasks to get $\mathcal{DCT}$ may not be the most sample efficient way. Thus, we adopt genetic algorithms (GAs) to gradually generate appropriate tasks. GA is a kind of search algorithm inspired by the process of natural selection, where a population of size NP is evolved iteratively [Pan *et al.*, 2019]. In our GA variant, the population is composed of candidate tasks k, represented by a group of simulation parameters $k = \{p_1, p_2, \cdots, p_n\}$, where the simulation parameters are restricted to obtain the variations of transition dynamics satisfying the Wasserstein constraints in Eq. 5. For each task k, we calculate the fitness score as:

$$fitness(k) = \frac{R(\pi_i|k) - R(\pi_{i-1}|k)}{R(\pi_{i-1}|k)}. \tag{6}$$

$$R(\pi|k) = \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_t, a_t)|k]. \tag{7}$$

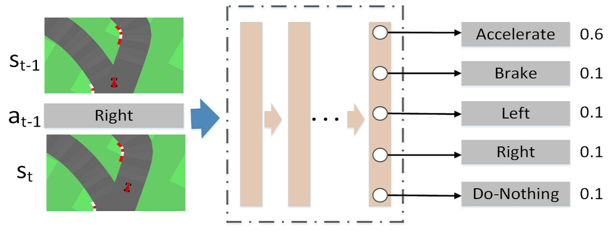


Figure 3: An example of Robust Policy Network.

Here, $R(\pi_i|k)$ and $R(\pi_{i-1}|k)$ are the task's expected reward of the current policy and the last policy, respectively. The fitness function defines the difficulty of a task to be the increase of reward, taking into account the policy change between the previous iteration and the current iteration [Xu *et al.*, 2020]. The task with high fitness indicates that the policy improves the task's reward significantly in the current iteration, and is more likely to learn better in the next iteration. On the contrary, the task with low fitness indicates that (1) the current policy does not have enough competence to handle it and may wait to be learned at a later iteration; or (2) the current policy performs enough well on this task having no space to further increase and do not need to be trained repeatedly. Therefore, the tasks with higher fitness are viewed with more appropriate difficulty for the current policy.

The goal is to find a group of tasks with higher fitness scores. At every generation, each task k is evaluated using Eq. 6, produces a fitness score, and is sorted according to the score. The top K candidates (called parents) are kept to the next generation. Then a two-point crossover of the selected parents is used to generate their child candidates (called offspring). Also, random mutation is applied to the child candidates with the mutation rate P_m. By running such a GA algorithm, we aim to gradually generate appropriate tasks for the current policy.

Task Evaluation

In task generation, a group of training tasks with high fitness scores are outputted. However, the increase-based fitness function neglects the current expected reward. For example, one task with a high fitness score may have an extremely low expected reward and is still a difficult task for the current policy. Thus, we consider the tasks with both high fitness scores and appropriate expected rewards to be feasible tasks, used for training the policy. Here, we give a definition on the feasible task:

Definition 2 (Feasible Task)**.** *For a policy π, we consider a task k to be a feasible task if π on task k is able to receive expected reward $R_{min} \leq R(\pi|k) \leq R_{max}$.*

On the one hand, $R(\pi|k) \geq R_{min}$ guarantees task k can obtain enough reward and the policy π can reach convergence more easily. On the other hand, $R(\pi|k) \leq R_{max}$ ensures we do not generate feasible tasks repeatedly from the already mastered tasks, as these mastered tasks usually have high rewards. The expected reward constraint in Definition 2 further guarantees the feasible tasks are at the appropriate level of difficulty for the current policy. After the task evaluation, only the feasible tasks $\mathcal{KG}$ are filtered out and will be used

Algorithm 1 Iterative training of DRRL

```
Input: Simulator environment $\varepsilon$; Size of the population $NP$; Size
    of the parent population $K$; Training iterations limit $N$; Gener-
    ation limit $G$; Wasserstein constraint $\mathcal{W}_\epsilon(\mathcal{P}_0)$
Output: Robust policy $\pi_N$
    $\mathcal{DCT} \leftarrow \{\mathcal{KG}_0 = \varnothing, \mathcal{KG}_1 = \varnothing, \cdots, \mathcal{KG}_N = \varnothing\}$
    $\mathcal{KG}_0 \leftarrow \{\varepsilon\}$
    $\pi_0 \leftarrow initialize\_policy(\mathcal{KG}_0)$
    $R(\pi_0) \leftarrow \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_{t-1}, a_{t-1}, s_t, a_t)|\varepsilon]$
    $R^1_{min} \leftarrow w_{min} \cdot R(\pi_0)$, $R^1_{max} \leftarrow w_{max} \cdot R(\pi_0)$
 1: for $i = 1$ to $N$ do
        // task generation
 2:     $population \leftarrow sample\_task(\mathcal{W}_\epsilon(\mathcal{P}_0), NP)$
 3:     for $m = 1$ to $M$ do
 4:         $fitness \leftarrow calculate\_fitness(population)$
 5:         $parents \leftarrow select(fitness, K)$
 6:         $population \leftarrow crossover\_and\_mutate(parents, NP)$
 7:     end for
        // task evaluation
 8:     for $k$ in $population$ do
 9:         $R(\pi_{i-1}|k) \leftarrow \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r((s_{t-1}, a_{t-1}, s_t, a_t)|k]$
10:         if $R^i_{min} \leq R(\pi_{i-1}|k) \leq R^i_{max}$ then
11:             $\mathcal{KG}_i \leftarrow add\_task(k)$
12:         end if
13:     end for
        // policy update
14:     $\pi_i \leftarrow update\_policy(\pi_{i-1}, \mathcal{KG}_i)$
15:     $R(\pi_i) \leftarrow \mathbb{E}_{k\sim\mathcal{DCT}}[\mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_{t-1}, a_{t-1}, s_t, a_t)|k]]$
16:     $R^{i+1}_{min} \leftarrow w_{min} \cdot R(\pi_i)$, $R^{i+1}_{max} \leftarrow w_{max} \cdot R(\pi_i)$
17: end for
18: return $\pi_N$
```

for updating the policy. In each training iteration, the final feasible tasks which are firstly generated by task generation and then filtered out by task evaluation, meet the difficulty requirement and guarantee the training stability.

Task Sequencing

In progressive multi-task learning, the difficulty levels for the feasible tasks should be updated as the policy is constantly updated. In other words, the values of R_{min} and R_{max} should be updated dynamically. For instance, at iteration m, we generate a group of feasible tasks $\mathcal{KG}_m$ for the current policy. Then, we update the policy to π_m on these tasks and obtain the expected reward:

$$R(\pi_m) = \mathop{\mathbb{E}}_{k\sim\mathcal{DCT}(m)}[\mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_t, a_t)|k]]. \tag{8}$$

$\mathcal{KG}_i$ represents the generated feasible tasks at iteration i, and $\mathcal{DCT}(m) = \bigcup_{i=0}^{m} \mathcal{KG}_i$ is the union of all generated feasible tasks in the first m iterations. In order to set appropriate difficulty level for the feasible tasks according to current expected reward $R(\pi_m)$, R_{min} and R_{max} will be updated as:

$$R^{m+1}_{min} = w_{min} \cdot R(\pi_m), \qquad R^{m+1}_{max} = w_{max} \cdot R(\pi_m). \tag{9}$$

Here, w_{min} and w_{max} are hyper-parameters. We suggest that $w_{min} \in (0.5, 1)$ and $w_{max} \in (1, 1.5)$. With new R_{min} and R_{max}, new task evaluation and policy training will be performed iteratively.

	Under no perturbations			**Under adversarial perturbations**			**Under test conditions**		
	Hopper	HalfCheetah	Walker2d	Hopper	HalfCheetah	Walker2d	Hopper	HalfCheetah	Walker2d
TRPO	**3118.13 ± 81.02**	5001.20 ± 83.59	2666.28 ± 48.93	568.07 ± 125.29	4541.38 ± 153.22	2120.50 ± 133.99	1788.77 ± 997.81	4767.51 ± 223.76	2608.65 ± 131.47
RARL	2914.45 ± 10.20	4428.40 ± 79.12	3171.30 ± 533.35	603.23 ± 52.79	4249.21 ± 86.91	2701.75 ± 124.31	1331.78 ± 1100.44	3877.24 ± 437.72	2808.26 ± 313.82
RAP-3	1243.71 ± 1182.23	4736.52 ± 55.96	2902.37 ± 47.11	903.54 ± 3.01	4498.53 ± 81.04	2732.50 ± 119.78	1516.11 ± 946.96	4398.36 ± 318.14	2491.56 ± 261.71
DR	2139.12 ± 0.63	4713.61 ± 89.52	3149.56 ± 104.35	309.51 ± 1.69	4283.08 ± 85.46	2825.99 ± 62.05	1680.86 ± 756.29	4579.08 ± 141.33	2995.08 ± 220.64
DRRL	3112.81 ± 4.18	**5403.69 ± 82.45**	**3552.98 ± 25.67**	**1503.39 ± 3.98**	**4888.74 ± 68.43**	**3549.95 ± 32.68**	**2190.15 ± 1103.02**	**5060.88 ± 323.45**	**3414.28 ± 268.17**
Imp.	-0.17%	8.05%	12.04%	66.39%	7.65%	25.62%	22.44%	6.15%	14.0%

Table 1: Policy performance under no perturbations, adversarial perturbations, and test conditions (Bold: best; Underline: runner-up). '*Imp.*' means the relative improvement percentage of our method against the best baseline.

3.3 Robust Policy Network

Traditional RL policy network takes the current state s as input and outputs a policy $\pi(s,a) = Pr(a_t = a|s_t = s)$, that gives an agent the probability of taking action a in state s.

In real-world applications, there are always several latent parameters that fail to be modeled in state space $\mathcal{S}$, such as friction in autonomous driving scenarios. Traditional policy networks just take the current state s_t as input and cannot perceive environmental setting variants. Consequently, an agent will tend to take conservative actions to avoid catastrophic events in some worst-cases. To solve this problem, we improve the policy network as follows:

$$\begin{aligned}&\pi^*(a_t|s_t, s_{t-1}, a_{t-1}) = \\ &\underset{\pi}{argmax}[\mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot r(s_{t-1}, a_{t-1}, s_t, a_t)|\mathcal{P}(\cdot)]].\end{aligned} \tag{10}$$

The agent would take action a_t according to not only the current state s_t but also the previous state s_{t-1} and previous action a_{t-1}. Taking the CarRacing scenario as an example, although the friction value of the testing scenario is difficult to obtain, we can infer the friction value indirectly. Generally, an autonomous vehicle may show different driving statuses given the same state and action due to different friction coefficients. Hence, in principle, if an autonomous vehicle takes action a_{t-1} at state s_{t-1}, making the state transfer to a new state s_t, we can extrapolate the current friction correspondingly. By using s_{t-1}, a_{t-1}, and s_t as pre-conditions of policy π, the improved policy network can represent possible latent environmental parameters in the training/test scenario.

As shown in Figure 3, if an autonomous vehicle takes action *turn right* at state s_{t-1}, and transfer to new state s_t, we can infer that the current friction is suitable. Then, according to policy $\pi^*(a_t|s_t, s_{t-1}, a_{t-1})$, the autonomous vehicle should take action $accelerate$. Accordingly, the final convergent policy will perform uniformly well in all cases.

The input space of the proposed policy network will be intuitively increased from $|s|$ to $|s|^2|a|$. Indeed, there are high temporal correlations between s_{t-1} and s_t. An agent takes action a_{t-1} at state s_{t-1} and transfers to state s_t, and meanwhile the number of possible actions is small. Hence, the input space can be reduced to $|s||a|^2$. In particular, for a given environmental parameter setting where the transition probability is fixed, $\pi^*(a_t|s_t, s_{t-1}, a_{t-1}) = \pi^*(a_t|s_t)$.

In summary, the proposed policy network will not bring about a serious increase in input space. Moreover, it can generate a separate policy for each environment parameter setting, while improving the overall performance on different transition probability variations.

3.4 Progressive Robust Policy Optimization

As mentioned earlier, we attempt to train a robust policy $\pi^*(a_t|s_t, s_{t-1}, a_{t-1})$ to maximize the expected reward in Eq. 10. Algorithm 1 outlines our approach in detail.

Initially, we train a policy based on the initial environment parameters of simulator. For the initial policy π_0, we calculate the expected reward $R(\pi_0)$ used to specify the appropriate level of difficulty for the feasible tasks at the first iteration (i.e., R^1_{min} and R^1_{max}). At each iteration i, we firstly use the GA to generate a group of appropriate tasks. The initial population of size NP is sampled from $\mathcal{W}_\epsilon(\mathcal{P}_0)$. At each generation m, the increase-based fitness scores are calculated based on Eq. 6, and the top K candidates sorted by the fitness scores are selected as parents. After the crossover and mutation, a new population is generated, where all the candidates satisfy the Wasserstein constraint $\mathcal{W}_\epsilon(\mathcal{P}_0)$. By iteratively running GA, a group of tasks with high fitness scores are outputted. Next in the task evaluation, we calculate each task's expected reward. The tasks satisfying the current difficulty level (see Definition 2) will be labeled as feasible tasks for the current policy and added to the $\mathcal{KG}_i$. Then, we use feasible tasks $\mathcal{KG}_i$ to update policy π_{i-1}, with the objective function given by Eq. 10. Noting that, in our DRRL, any RL algorithm can be used as the optimizer to train the policy. At last, we update R^{i+1}_{min} and R^{i+1}_{max} according to the new policy π_i, making the entire training process from easy to hard. The above process is repeated until convergence or the maximum number of iterations N is reached, and the current policy will be outputted as the final robust policy.

4 Experimental Results and Analysis

In our experiments[1], we validate the following hypotheses:

H1. Based on the max-expectation formulation, DRRL is able to improve the training stability and policy robustness compared to state-of-the-art methods.

H2. The proposed robust policy network enables the agent to learn from the history, leading to improved generalisation across environments with varying dynamics.

H3. The task generation and sequencing mechanism in DRRL provides better control over the perturbation difficulty, with the progressively hard task sequence.

[1]The code is available at https://github.com/li-yike/DRRL.

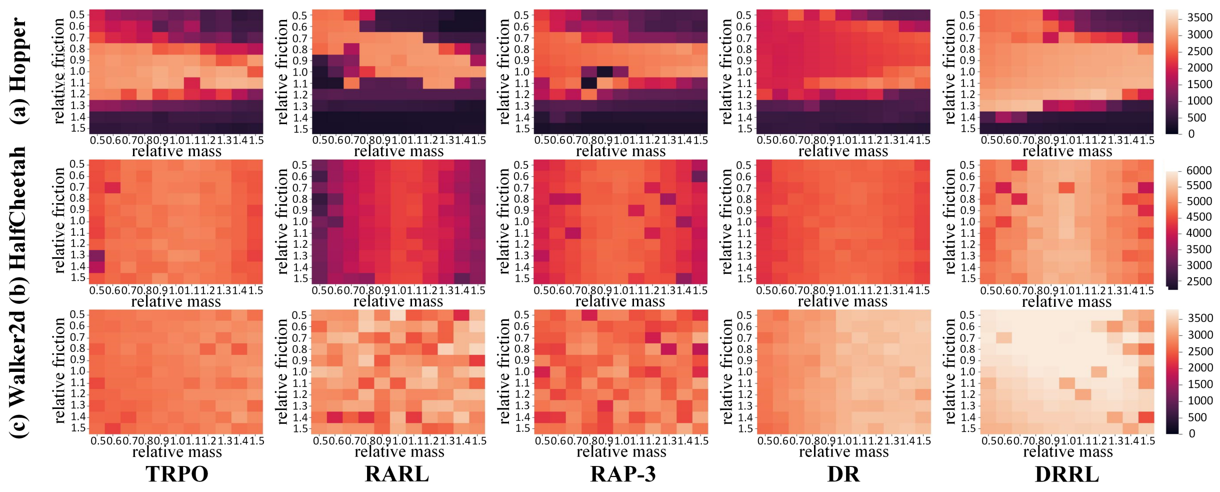


Figure 4: Mean reward visualization of heatmap w.r.t. different methods under test conditions.

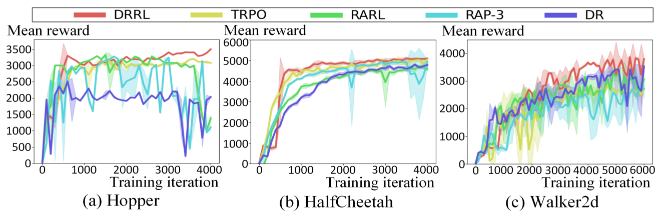


Figure 5: Training curves for DRRL and the baselines.

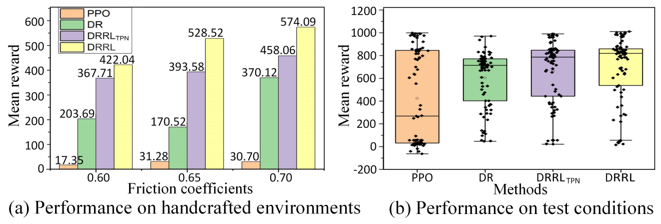


Figure 6: Policy Performance in the CarRacing environment.

4.1 H1. Robustness and Training Stability Analysis

In this evaluation, we implement the Hopper, HalfCheetah, and Walker2d benchmarks using OpenAI gym [Brockman *et al.*, 2016] with the MuJoCo simulator [Todorov *et al.*, 2012].

We compare our method with four state-of-the-art baselines. (1) TRPO [Schulman *et al.*, 2015], that realizes a strong RL without perturbations. (2) RARL [Pinto *et al.*, 2017], that trains the protagonist agent in the presence of an adversary agent's perturbations. (3) RAP [Vinitsky *et al.*, 2020], that uses a population of adversary agents for perturbations. The size of the adversary population is 3 (called RAP-3), in our implementation. (4) DR [Mehta *et al.*, 2020], that trains the agent under a set of random perturbations.

In our experiments, we use TRPO as the policy optimizer implemented by a neural network consisting of three hidden layers with 100 neurons each. We set the learning rate as 0.01 and other hyper-parameters (e.g., batchsize, discount factor) are tuned by grid search. For the GA implementation, the population size NP, the parent population size K, and the mutation rate P_m are set as 250, 50 and 0.9, respectively.

Training Stability

We analyze the training stability by comparing the mean and standard deviation of rewards at each training iteration. The comparison is shown in Figure 5, from which we can observe that for all the three environments, DRRL leads to a more stable increase of reward and a smaller standard deviation. These empirically highlight that by controlling perturbation difficulty and implementing training in a scheduled way, our DRRL gains better training stability.

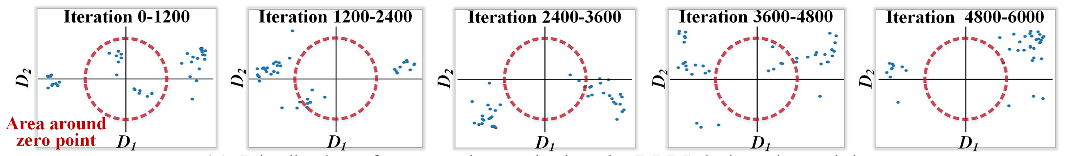
We then evaluate the learned policy in original environments without perturbations, and the results are shown in the first part of Table 1. We can observe that across the three environments, generally DRRL matches or outperforms all the baselines. The DRRL policy reaches the highest mean reward in HalfCheetah and Walker2d, with 8.05%, and 12.04% improvements compared to the second best results, respectively. This indicates that, although trained under perturbations, our DRRL still achieves high rewards under no perturbations.

Robustness under Adversarial Perturbations

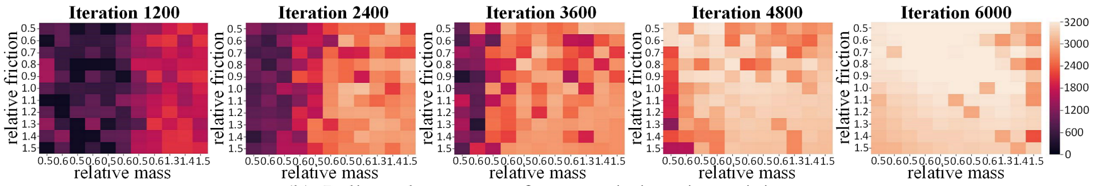
To simulate the uncertain effects of the latent parameters in the real world, following the idea of RARL, we train another adversarial agent to add adversarial forces to destabilize the protagonist agent. Under the well-trained adversary, the performance of the protagonist agent is shown in the second part of Table 1, from which we can observe that our method consistently outperforms all the baselines. The average improvements of our DRRL to the second best results are 66.39% in Hopper, 7.65% in HalfCheetah, and 25.62% in Walker2d. It proves that our method improves the robustness in the presence of adversarial perturbations, which enables the agent to better adapt to the latent environmental effects.

Robustness under Test Conditions

Further, we measure the robustness under a variety of test conditions, where different torso mass and friction coefficients are set for the three environments. The agent's rewards



(a) Distribution of generated perturbations by DRRL in iterative training



(b) Policy robustness performance in iterative training

Figure 7: Visualization of generated perturbations by DRRL and policy robustness performance in iterative training for Walker2d.

across all combinations are visualized in Figure 4, where each square denotes the mean reward of five trials. We observe that for all three environments, DRRL shows a significant outperformance on robustness against various test conditions. Note that, for Walker2d, the robust policy learned by DRRL is generalized to nearly all environmental settings instead of just earning high rewards locally. In addition, the quantitative experimental results are shown in the third part of Table 1 which reports the mean reward with standard deviations under all test conditions. Clearly, the performance of our DRRL is significantly higher than all baselines in these benchmarks.

4.2 H2. Effectiveness of Robust Policy Network

We conduct the following ablation study to demonstrate the effectiveness of the robust policy network in DRRL. Here, we compare DRRL with a variant DRRL$_{TPN}$, where the robust policy network (RPN) $\pi^*(a_t|s_t, s_{t-1}, a_{t-1})$ is replaced by the traditional policy network (TPN) $\pi^*(a_t|s_t)$ while other parts are kept unchanged. The ablation study is implemented in a new benchmark CarRacing, a top-down car racing from pixels environment with a three-dimensional continuous action space (i.e., steer, gas, brake). We introduce perturbations by setting friction coefficient $f \in [0.6, 1]$, gas error $E_{gas} \in [0, 0.4]$, steer error $E_{steer} \in [0, 0.4]$, and brake error $E_{brake} \in [0, 0.4]$.

Figure 6(a) shows the performance of the racing car under the handcrafted perturbed environments, where $E_{gas} = 0.4$, $E_{brake} = 0.4$, $E_{steer} = 0.4$, and $f \in \{0.6, 0.65, 0.7\}$. Besides, the performance under the test conditions is shown in Figure 6(b). To generate the test conditions, we take $E_{gas} \in \{0, 0.2, 0.4\}$, $E_{steer} \in \{0, 0.2, 0.4\}$, $E_{brake} \in \{0, 0.2, 0.4\}$ and $f \in \{1, 0.8, 0.6\}$, and apply them to different combinations (81 in total). As presented in both figures, overall, our DRRL is consistently superior to other methods. PPO is least competitive, illustrating that PPO lacks robustness when variations are introduced in the dynamics. DRRL$_{TPN}$ achieves better performance, which demonstrates the benefits of progressive multi-task learning with appropriate task difficulty. However, it still underperforms the DRRL, illustrating our proposed robust policy network is vital to improve generalisation across varying dynamics.

4.3 H3. Stepwise Analysis

Here, we examine the effectiveness of the GA-based progressive perturbation generation. Figure 7(a) shows the perturbations generated by DRRL in Walker2d, where each point denotes a two-dimensional adversarial force (D_1, D_2). We observe that during the training, the point distributions shift from the area around the zero point to the boundary. It indicates that along with the improvement of the policy, the generated tasks for the agent are increasingly difficult. This easy-to-hard task sequence enables the agent to generalize increasingly well under the test conditions, as shown in Figure 7(b).

5 Related Work

Robust RL problem has been studied extensively from different perspectives. DR-based methods [Peng *et al.*, 2018; Andrychowicz *et al.*, 2020; Loquercio *et al.*, 2019] introduce random perturbations by changing simulation parameters. Another line of works [Pinto *et al.*, 2017; Zhang *et al.*, 2020; Curi *et al.*, 2021; Jiang *et al.*, 2021] consider the adversarial multi-agent setting. Other works [Wang and Zou, 2021; Badrinath and Kalathil, 2021] inspired by Robust Markov Decision Process theory consider the worst-case perturbations from transition probabilities. More recently, there are some intrinsic motivation methods that train with generated curriculum [Sheng *et al.*, 2022; Dennis *et al.*, 2020], historical failures [Song and Schneider, 2022], and gradually changing bounds [Wu *et al.*, 2021; Liang *et al.*, 2022].

Different from existing works, we reformulate the robust RL as a max-expectation optimization, and solve it via dynamic multi-task learning under a progressive task sequence.

6 Conclusions

In this paper, we make the first attempt to formulate the robust RL as a max-expectation optimization problem to avoid conservative solutions. Then, we propose DRRL, which realizes dynamic multi-task learning and achieves better organization of robust RL training. In particular, an effective task generation and sequencing mechanism with increase-based task fitness is introduced to gradually generate appropriate tasks for iterative training. Experimental results including stability analysis, robustness analysis, stepwise analysis, and ablation analysis demonstrate the effectiveness of DRRL.

Acknowledgements

The work was supported by the Fundamental Research Funds for the Central Universities of China (2022YJS023), the National Natural Science Foundation of China (61972025, 61802389), the National Key Research and Development Program of China under Grant 2020YFB1005604 and Grant 2020YFB2103802.

Contribution Statement

Yike Li and Yunzhe Tian contributed equally to this work.

References

[Abdullah *et al.*, 2019] Mohammed Amin Abdullah, Hang Ren, Haitham Bou Ammar, Vladimir Milenkovic, Rui Luo, Mingtian Zhang, and Jun Wang. Wasserstein robust reinforcement learning. *CoRR*, abs/1907.13196, 2019.

[Andrychowicz *et al.*, 2020] 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.

[Badrinath and Kalathil, 2021] Kishan Panaganti Badrinath and Dileep Kalathil. Robust reinforcement learning using least squares policy iteration with provable performance guarantees. In *ICML*, pages 511–520, 2021.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[Chen *et al.*, 2021] Yifang Chen, Simon Du, and Kevin Jamieson. Improved corruption robust algorithms for episodic reinforcement learning. In *ICML*, pages 1561–1570, 2021.

[Curi *et al.*, 2021] Sebastian Curi, Ilija Bogunovic, and Andreas Krause. Combining pessimism with optimism for robust and efficient model-based deep reinforcement learning. In *ICML*, pages 2254–2264, 2021.

[Dalal *et al.*, 2021] Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. In *NeurIPS*, volume 34, pages 21847–21859, 2021.

[Dennis *et al.*, 2020] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *NeurIPS*, pages 13049–13061, 2020.

[Gao and Zhou, 2013] Wei Gao and Zhi-Hua Zhou. On the doubt about margin explanation of boosting. *Artificial Intelligence*, 203:1–18, 2013.

[He *et al.*, 2022] Shuncheng He, Yuhang Jiang, Hongchang Zhang, Jianzhun Shao, and Xiangyang Ji. Wasserstein unsupervised reinforcement learning. In *AAAI*, volume 36, pages 6884–6892, 2022.

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

[Jakobi, 1997] Nick Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

[Jiang *et al.*, 2021] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. In *NeurIPS*, pages 1884–1897, 2021.

[Kuang *et al.*, 2022] Yufei Kuang, Miao Lu, Jie Wang, Qi Zhou, Bin Li, and Houqiang Li. Learning robust policy against disturbance in transition dynamics via state-conservative policy optimization. In *AAAI*, volume 36, pages 7247–7254, 2022.

[Lecarpentier and Rachelson, 2019] Erwan Lecarpentier and Emmanuel Rachelson. Non-stationary markov decision processes, a worst-case approach using model-based reinforcement learning. In *NeurIPS*, volume 32, pages 7216–7225, 2019.

[Liang *et al.*, 2022] Yongyuan Liang, Yanchao Sun, Ruijie Zheng, and Furong Huang. Efficient adversarial training without attacking: Worst-case-aware robust reinforcement learning. In *NeurIPS*, pages 1–28, 2022.

[Loquercio *et al.*, 2019] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[Lyu *et al.*, 2022] Jiafei Lyu, Xiaoteng Ma, Jiangpeng Yan, and Xiu Li. Efficient continuous control with double actors and regularized critics. In *AAAI*, volume 36, pages 7655–7663, 2022.

[Mehta *et al.*, 2020] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *CoRL*, pages 1162–1176, 2020.

[Niekum, 2010] Scott Niekum. Evolved intrinsic reward functions for reinforcement learning. In *AAAI*, volume 24, pages 1955–1956, 2010.

[Nilim and El Ghaoui, 2005] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[Pan *et al.*, 2019] Xinlei Pan, Weiyao Wang, Xiaoshuai Zhang, Bo Li, Jinfeng Yi, and Dawn Song. How you act tells a lot: Privacy-leaking attack on deep reinforcement learning. In *AAMAS*, page 368–376, 2019.

[Peng *et al.*, 2018] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *ICRA*, pages 3803–3810, 2018.

[Pinto *et al.*, 2017] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *ICML*, pages 2817–2826, 2017.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[Sheng *et al.*, 2022] Junru Sheng, Peng Zhai, Zhiyan Dong, Xiaoyang Kang, Chixiao Chen, and Lihua Zhang. Curriculum adversarial training for robust reinforcement learning. In *IJCNN*, pages 1–8, 2022.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Song and Schneider, 2022] Yeeho Song and Jeff Schneider. Robust reinforcement learning via genetic curriculum. *CoRR*, abs/2202.08393, 2022.

[Tessler *et al.*, 2019] Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. In *ICML*, pages 6215–6224, 2019.

[Thomas and Brunskill, 2016] Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*, pages 2139–2148, 2016.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

[Vinitsky *et al.*, 2020] Eugene Vinitsky, Yuqing Du, Kanaad Parvate, Kathy Jang, Pieter Abbeel, and Alexandre Bayen. Robust reinforcement learning using adversarial populations. *CoRR*, abs/2008.01825, 2020.

[Wang and Zou, 2021] Yue Wang and Shaofeng Zou. Online robust reinforcement learning with model uncertainty. In *NeurIPS*, volume 34, pages 7193–7206, 2021.

[Wiesemann *et al.*, 2013] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust markov decision processes. *Mathematics of Operations Research*, 38(1):153–183, 2013.

[Wu *et al.*, 2021] Fan Wu, Linyi Li, Zijian Huang, Yevgeniy Vorobeychik, Ding Zhao, and Bo Li. Crop: Certifying robust policies for reinforcement learning through functional smoothing. In *ICLR*, pages 1–34, 2021.

[Xu *et al.*, 2020] Chen Xu, Bojie Hu, Yufan Jiang, Kai Feng, Zeyang Wang, Shen Huang, Qi Ju, Tong Xiao, and Jingbo Zhu. Dynamic curriculum learning for low-resource neural machine translation. In *COLING*, pages 3977–3989, 2020.

[Yu *et al.*, 2021] Jing Yu, Clement Gehring, Florian Schäfer, and Animashree Anandkumar. Robust reinforcement learning: A constrained game-theoretic approach. In *Learning for Dynamics and Control*, pages 1242–1254, 2021.

[Zhang and Zhou, 2018] Teng Zhang and Zhi-Hua Zhou. Optimal margin distribution clustering. In *AAAI*, volume 32, 2018.

[Zhang *et al.*, 2020] Kaiqing Zhang, Bin Hu, and Tamer Basar. On the stability and convergence of robust adversarial reinforcement learning: A case study on linear quadratic systems. In *NeurIPS*, volume 33, pages 22056–22068, 2020.