Wasserstein Robust Reinforcement Learning

Mohammed Amin Abdullah*
Huawei R&D UK
mohammed.abdullah@huawei.com

Hang Ren ‡
Huawei R&D UK
Imperial College London
hang.ren@huawei.com

Haitham Bou-Ammar *†
Huawei R&D UK
University College London
haitham.bouammar@huawei.com

Vladimir Milenković‡
Huawei R&D UK
University of Cambridge

Rui Luo †
Huawei R&D UK
University College London

Mingtian Zhang†
Huawei R&D UK
University College London

Jun Wang
Huawei R&D UK
University College London

Abstract

Reinforcement learning algorithms, though successful, tend to over-fit to training environments hampering their application to the real-world. This paper proposes WR^{2}L – a robust reinforcement learning algorithm with significant robust performance on low and high-dimensional control tasks. Our method formalises robust reinforcement learning as a novel min-max game with a Wasserstein constraint for a correct and convergent solver. Apart from the formulation, we also propose an efficient and scalable solver following a novel zero-order optimisation method that we believe can be useful to numerical optimisation in general. We empirically demonstrate significant gains compared to standard and robust state-of-the-art algorithms on high-dimensional MuJuCo environments.

1   Introduction

Reinforcement learning (RL) has become a standard tool for solving decision-making problems with minimal feedback. Applications with these characteristics are ubiquitous, including, but not limited to, computer games (Mnih et al., 2013), robotics (Kober and Peters, 2012; Deisenroth et al., 2013; Bou-Ammar et al., 2014), finance (Fischer, 2018), and personalised medicine (Emmert-Streib and Dehmer, 2018). Although significant progress has been made on developing algorithms for learning large-scale and high-dimensional reinforcement learning tasks, these algorithms often over-fit to training environments and fail to generalise across even slight variations of transition dynamics (Packer et al., 2018; Zhao et al., 2019).

---

*The first three authors are to be considered as joint first co-authors

†Honorary Lecturer Position at University College London.

‡Work done while interning at the Reinforcement Learning Team in Huawei Technologies Research and Development in London.

Robustness to changes in transition dynamics, however, is a crucial component for adaptive and safe RL in real-world environments. To illustrate, consider a self-driving car scenario in which we attempt to design an agent capable of driving a vehicle smoothly, safely, and autonomously. A typical reinforcement learning work-flow to solving such a problem consists of building a simulator to emulate real-world scenarios, training in simulation, and then transferring resultant policies to physical systems for control. Unfortunately, such a strategy is easily prone to failure as designing accurate simulators that capture intricate complexities of large cities is extremely challenging. Rather than learning in simulation, another work-flow might consist of constructing a pipeline to directly learn on the hardware system itself. Apart from memory constraints, state-of-the-art reinforcement learning algorithms exhaust hundreds to millions of agent-environment interactions before acquiring successful behaviour. Of course, such high demands on sample complexities prohibit the direct application of learning algorithms on real-systems, leaving robustness to misspecified simulators a largely unresolved problem.

Motivated by real-world applications, recent literature in reinforcement learning has focused on the above problems, proposing a plethora of algorithms for robust decision-making (Morimoto and Doya, 2005; Pinto et al., 2017; Tessler et al., 2019). Most of these techniques borrow from game theory to analyse, typically in a discrete state and actions spaces, worst-case deviations of agents' policies and/or environments, see Sargent and Hansen (2001); Nilim and El Ghaoui (2005); Iyengar (2005); Namkoong and Duchi (2016) and references therein. These methods have also been extended to linear function approximators (Chow et al., 2015), and deep neural networks (Peng et al., 2017) showing (modest) improvements in performance gain across a variety of disturbances, e.g., action uncertainties, or dynamical model variations.

Though successful in practice, current techniques to robust decision making remain task-specific, and there are two major lingering drawbacks. First, these algorithms fail to provide a general robustness framework due to their specialised heuristics. In particular, algorithms designed for discrete state-action spaces fail to generalise to continuous domains and vice versa. This is due to their exploiting mathematical properties valid only for discrete state and action spaces, or for convex-concave functions, e.g., $\min\max f(\cdot) = \max\min f(\cdot)$. Clearly, such mathematical derivations are only loosely related to implementation, and further insights can be gathered with more rigorous attempts tackling the continuous problem itself. Apart from theoretical limitations, the second drawback of current approaches can be traced back to the process by which empirical validation is conducted. Rarely do the papers presenting these techniques compare gains against other robust algorithms in literature. In fact, focus is mainly diverted to outperforming state-of-the-art reinforcement learning – a criterion easy to satisfy as algorithms from RL were never trained for robustness. Interestingly, upon careful evaluation, we came to realise that robust adversarial reinforcement learning of Pinto et al. (2017), for example, is superior to some of the more recently published works attempting to solve the same problem, see Section 6.

In this paper, we contribute to the above endeavour to solve the robustness problem in RL by proposing a generic framework for robust reinforcement learning that can cope with both discrete and continuous state and actions spaces. The algorithm we introduce, which we call *Wasserstein Robust Reinforcement Learning* (WR^{2}L), Algorithm 1, aims to find the best policy, where any given policy is judged by the worst-case dynamics amongst all candidate dynamics in a certain set. This set is essentially the average Wasserstein ball around a reference dynamics $\mathcal{P}_0$. The constraints makes the problem well-defined, as searching over arbitrary dynamics can only result in non-performing system. The measure of performance is the standard RL objective form, the expected return. Both the policy and the dynamics are parameterised; the policy parameters $\boldsymbol{\theta}_k$ may be the weights of a deep neural network, and the dynamics parameters $\boldsymbol{\phi}_j$ the parameters of a simulator or differential equation solver. The algorithm performs estimated descent steps in $\boldsymbol{\phi}$ space and - after (almost) convergence - performs an update of policy parameters, i.e., in $\boldsymbol{\theta}$ space. Since $\boldsymbol{\phi}_j$ may be high-dimensional, we adapt a zero'th order sampling method based extending Salimans et al. (2017) to make estimations of gradients, and in order to define the constraint set which $\boldsymbol{\phi}_j$ is bounded by, we generalise the technique to estimate Hessians (Proposition 2).

We emphasise that although access to a simulator with parameterisable dynamics are required, the actual reference dynamics $\mathcal{P}_0$ need not be known explicitly nor learnt by our algorithm. Put another way, we are in the "RL setting", not the "MDP setting" where the transition probability matrix is known *a priori*. The difference is made obvious, for example, in the fact that we cannot perform dynamic programming, and the determination of a particular probability transition can only be es-

timated from sampling, not retrieved explicitly. Hence, our algorithm is not model-based in the traditional sense of learning a model to perform planning.

We believe our contribution is useful and novel for two main reasons. Firstly, our framing of the robust learning problem is in terms of dynamics uncertainty sets defined by Wasserstein distance. Whilst we are not the first to introduce the Wasserstein distance into the context of MDPs (see, e.g., Yang (2017) or Lecarpentier and Rachelson (2019)), we believe our formulation is amongst the first suitable for application to the demanding application-space we desire, that being, high-dimensional, continuous state and action spaces. Secondly, we believe our solution approach is both novel and effective (as evidenced by experiments below, see Section 6), and does not place a great demand on model or domain knowledge, merely access to a simulator or differentiable equation solver that allows for the parameterisation of dynamics. Furthermore, it is not computationally demanding, in particular, because it does not attempt to build a model of the dynamics, and operations involving matrices are efficiently executable using the Jacobian-vector product facility of automatic differentiation engines.

The rest of the paper is organised as follows. In the next section, we provide a background on notation and an overview of Wasserstein distance in its various forms. The problem formulation and main algorithm is then presented in Section 3. In Section 4, we describe the zero'th order method we use to estimate the Hessian matrix used by the algorithm, and the proof of its correctness is given. In Section 5 we survey related literature. Experiments and results are given in Section 6, and finally, conclusions and future work are discussed in Section 7.

2 Background

This section provides background material needed for the remainder of the paper. We first describe the reinforcement learning framework adopted in this paper, and then proceed to detail notions from Wasserstein distances needed to constrain our learning objective.

2.1 Reinforcement Learning

In reinforcement learning, an agent interacts with an unknown and stochastic environment with the goal of maximising a notion of return (Sutton and Barto, 1998; Peters and Schaal, 2008b; Busoniu et al., 2010). These problems are typically formalised as Markov decision processes (MDPs)[4] $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S} \subseteq \mathbb{R}^d$ denotes the state space, $\mathcal{A} \subseteq \mathbb{R}^n$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a state transition probability describing the system's dynamics, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function measuring the agent's performance, and $\gamma \in [0, 1)$ specifies the degree to which rewards are discounted over time.

At each time step t, the agent is in state $\boldsymbol{s}_t \in \mathcal{S}$ and must choose an action $\boldsymbol{a}_t \in \mathcal{A}$, transitioning it to a new state $\boldsymbol{s}_{t+1} \sim \mathcal{P}\left(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t\right)$, and yielding a reward $\mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)$. A policy $\pi : \mathcal{S} \times A \to [0, 1]$ is defined as a probability distribution over state-action pairs, where $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ represents the density of selecting action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$. Upon consequent interactions with the environment, the agent collects a trajectory $\boldsymbol{\tau}$ of state-action pairs. The goal is to determine an optimal policy $\pi^{\star}$ by solving:

$$\pi^{\star} = \arg \max_{\pi} \mathbb{E}_{\boldsymbol{\tau} \sim p_{\pi}(\boldsymbol{\tau})} \left[\mathcal{R}_{\text{Total}}(\boldsymbol{\tau}) \right], \tag{1}$$

where $p_{\pi}(\boldsymbol{\tau})$ denotes the trajectory density function, and $\mathcal{R}_{\text{Total}}(\boldsymbol{\tau})$ the return, that is, the total accumulated reward:

$$p_{\pi}(\boldsymbol{\tau}) = \mu_0(\boldsymbol{s}_0)\pi(\boldsymbol{a}_0|\boldsymbol{s}_0) \prod_{t=1}^{T-1} \mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \ \ \text{and} \ \ \mathcal{R}_{\text{Total}}(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t), \tag{2}$$

with $\mu_0(\cdot)$ denoting the initial state distribution.

---

[4]Please note that we present reinforcement learning with continuous states and actions. This allows us to easily draw similarities to optimal control as detailed later. Extending these notions to discrete settings is relatively straight-forward.

2.2 Wasserstein Distance

In our problem definition, we make use of a distance measure to bound allowed variations from a reference transition density $\mathcal{P}_0(\cdot)$. In general, a number of common metrics for measuring closeness between two probability distributions exist. Examples of which are total variation distance and Kullback-Leibler divergence. In this paper, however, we measure distance between two different dynamics by the *Wasserstein distance*. This has a number of desirable properties; firstly, it is a genuine distance, exhibiting symmetry, which is a property that K-L divergence lacks. Secondly, it is very flexible in the forms of the distributions that can be compared; it can measure the distance between two discrete distributions, two continuous distributions, and a discrete and continuous distribution (this latter case implying another valuable advantage - that the supports of the distributions can be different). In all cases, the Wasserstein distance is well-defined. Finally, and perhaps most importantly, the Wasserstein distance takes into account the underlying geometry of the space the distributions are defined on, which could be information that is fruitful to exploit in learning optimal control. Indeed this last point is the core motivator for our choosing Wasserstein distance for our algorithm, as shall be explained later.

Definition: Given a measurable space $(\mathcal{X}, \mathcal{F})$ with $\mathcal{X}$ being a metric space, a pair of discrete measures μ, ν defined on this measurable space can be written as $\mu = \sum_{i=1}^{n} \mu_i \delta_{x_i}$ and $\nu = \sum_{j=1}^{m} \nu_j \delta_{y_j}$ where all $x_i, y_j \in \mathcal{X}$. A coupling $\kappa(\cdot, \cdot)$ of μ and ν is a measure over $\{x_1, \ldots, x_n\} \times \{y_1, \ldots y_m\}$ that preserves marginals, i.e, $\mu_i = \sum_j \kappa(\mu_i, \nu_j) \; \forall i$ and $\nu_j = \sum_i \kappa(\mu_i, \nu_j) \; \forall j$. This then induces a cost of "moving" the mass of μ to ν, given as the (Frobenius) inner product $\langle \kappa, C \rangle$ where the matrix $C \in \mathbb{R}^{n \times m}$ has $[C]_{ij} = c_{ij} = d(x_i, y_j)$, i.e., the cost of moving a unit of measure from x_i to y_j. Minimised over the space of all couplings $\mathbf{K}(\mu, \nu)$, we get the Wasserstein distance, also known as the *Earth-Mover Distance* (EMD).

More generally, let $\mathcal{X}$ be a metric space with metric $d(\cdot, \cdot)$. Let $\mathcal{C}(\mathcal{X})$ be the space of continuous functions on $\mathcal{X}$ and let $\mathcal{M}(\mathcal{X})$ be the set of probability measures on $\mathcal{X}$. Let $\mu, \nu \in \mathcal{M}(\mathcal{X})$. Let $\mathbf{K}(\mu, \nu)$ be the set of couplings between μ, ν:

$$\mathbf{K}(\mu, \nu) := \{\kappa \in \mathcal{M}(\mathcal{X} \times \mathcal{X}) ; \forall (A, B) \subset \mathcal{X} \times \mathcal{X}, \kappa(A \times \mathcal{X}) = \mu(A), \kappa(\mathcal{X} \times B) = \nu(B)\} \quad (3)$$

That is, the set of joint distributions $\kappa \in \mathcal{M}(\mathcal{X} \times \mathcal{X})$ whose marginals agree with μ and ν respectively. Given a metric (serving as a cost function) $d(\cdot, \cdot)$ for $\mathcal{X}$, the p'th Wasserstein distance $W_p(\mu, \nu)$ for $p \geq 1$ between μ and ν is defined as:

$$W_p(\mu, \nu) := \left(\min_{\kappa \in \mathbf{K}(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} d(x, y)^p d\kappa(x, y) \right)^{1/p} \quad (4)$$

3 Wasserstein Robust Reinforcement Learning

This section formalises robust reinforcement learning by equipping agents with capabilities of determining well-behaved policies under worst-case models which are bounded in ϵ-Wasserstein balls. Our motivation for formalising WR^{2}L is rooted in robust optimal control – a field dedicated to determining optimal action-selection rules under uncertainties induced by modelling assumptions. Here an agent controlling a plant/system faces an adversary that optimises for a disturbance controller while aiming at minimising rewards received by the agent. Interestingly, these problems relate to two-player min-max games and provide a rich literature with efficient solutions in certain specific scenarios, e.g., discrete states and actions, robust linear quadratic regulators, among others. Though generic min-max objectives can lead to robustness (see Pinto et al. (2017)), typical algorithms rooted in game-theory optimise well-behaved objectives by introducing additional structural assumptions to adversaries. For example, it is not uncommon in robust optimal control to assume the process by which disturbances are applied (e.g., additive, multiplicative), or to only consider a subset adhering to maximally bounded norms [5].

Starting from robust optimal control, we derive WR^{2}L's objective by introducing two major refinements to standard reinforcement learning. Similar to robust control, we enable agents to perturb

---

[5] Please note that this is not to say that robust optimal control is restricted to the above settings, see Doyle et al. (2013).

transition models from a reference simulator with the goal of determining worst-case scenarios. We do not, however, pose additional structural assumptions on the process by which adversaries apply these perturbations. Rather, we posit a parameterised class of disturbances and adopt zero'th order optimisation (see Section 4) for flexibility, thus broadening our application spectrum to high-dimensional and stochastic dynamical systems. Optimisation problems of this nature, on the other hand, tend to be ill-specified due to the unconstrained process by which models are fit. To bound allowed variations in transition models, and ensure correctness and tractability, we then introduce an ϵ-ball Wasserstein constraint around a simulator $\mathcal{P}_0(\cdot)$ to guarantee convergence.

Before continuing, however, it is worth revisiting the motivations for choosing such a distance. Per the definition, constraining the possible dynamics to be within an ϵ-Wasserstein ball of a reference dynamics $\mathcal{P}_0(\cdot)$ means constraining it in a certain way. Wasserstein distance has the form mass $\times$ distance. If this quantity is constrained to be less than a constant ϵ, then if the mass is large, the distance is small, and if the distance is large, the mass is small. Intuitively, when modelling the dynamics of a system, it may be reasonable to concede that there could be a systemic error - or bias - in the model, but that bias should not be too large. It is also reasonable to suppose that occasionally, the behaviour of the system may be wildly different to the model, but that this should be a low-probability event. If the model is frequently wrong by a large amount, then there is no use in it. In a sense, the Wasserstein ball formalises these assumptions.

In what comes next, we detail the problem definition as a novel min-max game with Wasserstein constrained dynamics. In Section 3.2, we elaborate a generic algorithm capable of robustly updating both model and policy parameters.

3.1 Problem Definition: Robust Objectives and Constraints

As mentioned earlier, the problem definition we introduce in this paper extends reinforcement learning in two directions. In the first, we introduce a min-max objective with parameterised transition models, while in the second, we incorporate Wasserstein constraints to bound allowed perturbations.

Parameterising Policies and Transition Models: Due to the continuous nature of the state and action spaces considered in this work, we resort to deep neural networks to parameterise policies, which we write as $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t)$, where $\boldsymbol{\theta} \in \mathbb{R}^{d_1}$ is a set of tunable hyper-parameters to optimise. For instance, these policies can correspond to multi-layer perceptrons for MuJoCo environments, or to convolutional neural networks in case of high-dimensional states depicted as images. Exact policy details are ultimately application dependent and, consequently, provided in the relevant experiment sections.

In principle, one can similarly parameterise transition models using deep networks (e.g., LSTM-type models) to provide one or more action-conditioned future state predictions. Though appealing, going down this path led us to agents that discover worst-case transition models which minimise training data but lack any valid physical meaning. For instance, original experiments we conducted on CartPole ended up involving transitions that alter angles without any change in angular velocities. More importantly, these effects became more apparent in high-dimensional settings where the number of potential minimisers increases significantly. It is worth noting that we are not the first to realise such an artifact when attempting to model physics-based dynamics using deep networks. Authors in (Lutter et al., 2019) remedy these problems by introducing Lagrangian mechanics to deep networks, while others (Koller et al., 2018; Chen et al., 2018) argue the need to model dynamics given by differential equation structures directly.

Though incorporating physics-based priors to deep networks is an important and challenging task that holds the promise of scaling model-based reinforcement learning for efficient solvers, in this paper we rather study an alternative direction focusing on perturbing differential equation solvers and/or simulators with respect to the dynamic specification parameters $\boldsymbol{\phi} \in \mathbb{R}^{d_2}$. Not only would such a consideration reduce the dimensionality of parameter spaces representing transition models, but would also guarantee valid dynamics due to the nature of the discrete differential equation solver. Though tackling some of the above problems, such a direction arrives with a new set of challenges related to computing gradients and Hessians of black-box solvers. In Section 4, we develop an efficient and scalable zero-order method for valid and accurate model updates.

Unconstrained Loss Function: Having equipped agents with the capability of representing policies and perturbing transition models, we are now ready to present an unconstrained version of WR^{2}L's loss function. Borrowing from robust optimal control, we define robust reinforcement learning as an algorithm that learns best-case policies under worst-case transitions:

$$\max_{\boldsymbol{\theta}} \left[\min_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}^{\phi}(\boldsymbol{\tau})} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right] \right], \tag{5}$$

where $p_{\boldsymbol{\theta}}^{\phi}(\boldsymbol{\tau})$ is a trajectory density function parameterised by both policies and transition models, i.e., $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, respectively:

$$p_{\boldsymbol{\theta}}^{\phi}(\boldsymbol{\tau}) = \mu_0(\boldsymbol{s}_0)\pi(\boldsymbol{a}_0|\boldsymbol{s}_0) \prod_{t=1}^{T-1} \underbrace{\mathcal{P}_{\phi}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)}_{\text{specs vector and diff. solver}} \underbrace{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)}_{\text{deep network}}.$$

At this stage, it should be clear that our formulation, though inspired from robust optimal control, is, truthfully, more generic as it allows for parameterised classes of transition models without incorporating additional restrictions on the structure or the scope by which variations are executed[6].

Constraints & Complete Problem Definition: Clearly, the problem in Equation 5 is ill-defined due to the arbitrary class of parameterised transitions. To ensure well-behaved optimisation objectives, we next introduce constraints to bound search spaces and ensure convergence to feasible transition models. For a valid constraint set, our method assumes access to samples from a *reference dynamics model* $\mathcal{P}_0(\cdot|\boldsymbol{s}, \boldsymbol{a})$, and bounds learnt transitions in an ϵ-Wasserstein ball around $\mathcal{P}_0(\cdot|\boldsymbol{s}, \boldsymbol{a})$, i.e., the set defined as:

$$\mathcal{W}_\epsilon\left(\mathcal{P}_{\phi}(\cdot), \mathcal{P}_0(\cdot)\right) = \left\{ \mathcal{P}_{\phi}(\cdot) : \mathcal{W}_2^2\left(\mathcal{P}_{\phi}(\cdot|\boldsymbol{s}, \boldsymbol{a}), \mathcal{P}_0(\cdot|\boldsymbol{s}, \boldsymbol{a})\right) \leq \epsilon, \ \forall(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A} \right\}, \tag{6}$$

where $\epsilon \in \mathbb{R}_+$ is a hyperparameter used to specify the "degree of robustness" in a similar spirit to maximum norm bounds in robust optimal control. It is worth noting, that though we have access to samples from a reference simulator, our setting is by no means restricted to model-based reinforcement learning in an MDP setting. That is, our algorithm operates successfully given only traces from $\mathcal{P}_0$ accompanied with its specification parameters, e.g., pole lengths, torso masses, etc. – a more flexible framework that does not require full model learners.

Though defining a better behaved optimisation objective, the set in Equation 6 introduces infinite number of constraints when considering continuous state and/or actions spaces. To remedy this problem, we target a relaxed version that considers average Wasserstein constraints instead:

$$\hat{\mathcal{W}}_\epsilon^{(\text{average})}\left(\mathcal{P}_{\phi}(\cdot), \mathcal{P}_0(\cdot)\right) = \left\{ \mathcal{P}_{\phi}(\cdot) : \int_{(\boldsymbol{s}, \boldsymbol{a})} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}) \mathcal{W}_2^2\left(\mathcal{P}_{\phi}(\cdot|\boldsymbol{s}, \boldsymbol{a}), \mathcal{P}_0(\cdot|\boldsymbol{s}, \boldsymbol{a})\right) d(\boldsymbol{s}, \boldsymbol{a}) \leq \epsilon \right\} \tag{7}$$

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

In words, Equation 7 defines a set where *expected* Wasserstein distance is bounded by ϵ. Expectation in the above equation is evaluated over state-action pairs sampled according to $\mathcal{P}(\boldsymbol{s}, \boldsymbol{a})$ which is policy and transition-model dependent. Precisely, one can factor $\mathcal{P}(\boldsymbol{s}, \boldsymbol{a})$ as:

$$\mathcal{P}(\boldsymbol{s} \in \mathcal{S}, \boldsymbol{a} \in \mathcal{A}) = \mathcal{P}(\boldsymbol{a} \in \mathcal{A}|\boldsymbol{s} \in \mathcal{S})\mathcal{P}(\boldsymbol{s} \in \mathcal{S}) = \pi(\boldsymbol{a} \in \mathcal{A}|\boldsymbol{s} \in \mathcal{S})\rho_{\pi}^{\phi_0}(\boldsymbol{s} \in \mathcal{S}),$$

where $\boldsymbol{s} \in \mathcal{S}$ and $\boldsymbol{a} \in \mathcal{A}$ are to be interpreted as events being elements of the state and action sets – a notation better suites for the continuous nature of the considered random variables. Moreover, $\rho_{\pi}^{\phi_0}(\boldsymbol{s} \in \mathcal{S})$ is a uniform distribution over state-actions pairs sampled from a trajectory. Precisely, the way we compute the expected Wasserstein distance is two steps. In the first, given a batch of trajectories sampled according to any policy π, potentially of varying lengths, we create a bucket of state-action pairs[7]. Given such data, we then compute expected Wasserstein distance over the pairs using Monte-Carlo estimation. The π we use is one which samples actions uniformly at random.

---

[6]Of course, allowed perturbations are ultimately constrained by the hypothesis space. Even then, our model is more general compared to robust optimal control that assumes additive, multiplicative, or other forms of disturbances.

[7]Since we do not require access to current policies in order to compute the average Wasserstein distance, an argument can be made that WR^{2}L support off-policy constraint evaluation. This is an important link that we plan to further exploit in the future to improve efficiency, scalablity, and enable transfer between various tasks.

With this in mind, we arrive at WR^{2}L's optimisation problem allowing for best policies under worst-case yet bounded transition models:

Wasserstein Robust Reinforcement Learning Objective:

$$\max_{\boldsymbol{\theta}} \left[\min_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}^{\boldsymbol{\phi}}(\boldsymbol{\tau})} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right] \right] \quad \text{s.t.} \quad \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim \pi(\cdot) \rho_{\pi}^{\phi_0}(\cdot)} \left[\mathcal{W}_2^2 \left(\mathcal{P}_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s},\boldsymbol{a}), \mathcal{P}_0(\cdot|\boldsymbol{s},\boldsymbol{a}) \right) \right] \leq \epsilon$$

(8)

3.2 Solution Methodology

Having derived our formal problem definition, this section presents our approach to solving for $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ in the objective of Equation 8. On a high level, our solution methodology follows an alternating procedure interchangeably updating one variable given the other fixed.

Updating Policy Parameters: It is clear from Equation 8 that the average Wasserstein distance constraint is independent from $\boldsymbol{\theta}$ and can, in fact, use any other policy π to estimate the expectation. Hence, given a fixed set of model parameters, $\boldsymbol{\theta}$ can be updated by solving the relevant sub-problem of Equation 8 written as:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}^{\boldsymbol{\phi}}(\boldsymbol{\tau})} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right].$$

Interestingly, this problem is a standard reinforcement learning one with a crucial difference in that traces are sampled according to the transition model given by fixed model parameters, $\boldsymbol{\phi}$ that ultimately differ from these of the original simulator ϕ_0. Consequently, one can easily adapt any policy search method for updating policies under fixed dynamical models. As described later in Section 4, we make use of proximal policy optimisation (Schulman et al., 2017), for instance, to update such action-selection-rules.

Updating Model Parameters: Now, we turn our attention to solving the average constraint optimisation problem needed for updating $\boldsymbol{\phi}$ given a set of fixed policy parameters $\boldsymbol{\theta}$. Contrary to the previous step, here, the Wasserstein constraints play an important role due to their dependence on $\boldsymbol{\phi}$. Unfortunately, even with the simplification introduced in Section 3.1 the resultant constraint is still difficult to computer in general, the difficulty being the evaluation of the Wasserstein term[8].

To alleviate this problem, we propose to approximate the constraint in (8) by its Taylor expansion up to second order. That is, defining

$$W(\boldsymbol{\phi}) := \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim \pi(\cdot) \rho_{\pi}^{\phi_0}(\cdot)} \left[\mathcal{W}_2^2 \left(\mathcal{P}_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s},\boldsymbol{a}), \mathcal{P}_0(\cdot|\boldsymbol{s},\boldsymbol{a}) \right) \right]$$

The above can be approximated around ϕ_0 by a second-order Taylor as:

$$W(\boldsymbol{\phi}) \approx W(\phi_0) + \nabla_{\boldsymbol{\phi}} W(\phi_0)^{\mathsf{T}} (\boldsymbol{\phi} - \phi_0) + \frac{1}{2} (\boldsymbol{\phi} - \phi_0)^{\mathsf{T}} \nabla_{\boldsymbol{\phi}}^2 W(\phi_0) (\boldsymbol{\phi} - \phi_0).$$

Recognising that $W(\phi_0) = 0$ (the distance between the same probability densities), and $\nabla_{\boldsymbol{\phi}} W(\phi_0) = 0$ since ϕ_0 minimises $W(\boldsymbol{\phi})$, we can simplify the Hessian approximation by writing:

$$W(\boldsymbol{\phi}) \approx \frac{1}{2} (\boldsymbol{\phi} - \phi_0)^{\mathsf{T}} \nabla_{\boldsymbol{\phi}}^2 W(\phi_0) (\boldsymbol{\phi} - \phi_0).$$

Substituting our approximation back in the original problem in Equation 8, we reach the following optimisation problem for determining model parameter given fixed policies:

$$\min_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}^{\boldsymbol{\phi}}(\boldsymbol{\tau})} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right] \quad \text{s.t.} \quad \frac{1}{2} (\boldsymbol{\phi} - \phi_0)^{\mathsf{T}} \boldsymbol{H}_0 (\boldsymbol{\phi} - \phi_0) \leq \epsilon, \tag{9}$$

---

[8]The situation is easier in case of two Gaussian densities. Here, however, we keep the treatment general by proposing an alternative direction using Taylor expansions.

where $\boldsymbol{H}_0 = \nabla_{\boldsymbol{\phi}}^2 \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}(\cdot)} \left[\mathcal{W}_2^2 \left(\mathcal{P}_\phi(\cdot|\boldsymbol{s},\boldsymbol{a}), \mathcal{P}_0(\cdot|\boldsymbol{s},\boldsymbol{a}) \right) \right] \Big|_{\phi=\phi_0}$ is the Hessian of the expected squared 2-Wasserstein distance evaluated at ϕ_0.

Optimisation problems with quadratic constraints can be efficiently solved using interior-point methods. To do so, one typically approximates the loss with a first-order expansion and determines a closed-form solution. Consider a pair of parameters $\boldsymbol{\theta}^{[k]}$ and $\boldsymbol{\phi}^{[j]}$ (which will correspond to parameters of the j'th inner loop of the k'th outer loop in the algorithm we present). To find $\boldsymbol{\phi}^{[j+1]}$, we solve:

$$\min_{\boldsymbol{\phi}} \nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\tau}\sim p_{\boldsymbol{\theta}}^\phi(\boldsymbol{\tau})} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right] \Big|_{\boldsymbol{\theta}^{[k]},\boldsymbol{\phi}^{[j]}}^{\mathsf{T}} (\boldsymbol{\phi} - \boldsymbol{\phi}^{[j]}) \text{ s.t. } \frac{1}{2}(\boldsymbol{\phi} - \boldsymbol{\phi}_0)^{\mathsf{T}} \boldsymbol{H}_0 (\boldsymbol{\phi} - \boldsymbol{\phi}_0) \leq \epsilon.$$

It is easy to show that a minimiser to the above equation can derived in a closed-form as:

$$\boldsymbol{\phi}^{[j+1]} = \boldsymbol{\phi}_0 - \sqrt{\frac{2\epsilon}{\boldsymbol{g}^{[k,j]\mathsf{T}} \boldsymbol{H}_0^{-1} \boldsymbol{g}^{[k,j]}}} \boldsymbol{H}_0^{-1} \boldsymbol{g}^{[k,j]}, \tag{10}$$

with $\boldsymbol{g}^{[k,j]}$ denoting the gradient[9] evaluated at $\boldsymbol{\theta}^{[k]}$ and $\boldsymbol{\phi}^{[j]}$, i.e., $\boldsymbol{g}^{[k,j]} = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\tau}\sim p_{\boldsymbol{\theta}}^\phi(\boldsymbol{\tau})} \mathbb{E} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right] \Big|_{\boldsymbol{\theta}^{[k]},\boldsymbol{\phi}^{[j]}}$.

Generic Algorithm: Having described the two main steps needed for updating policies and models, we now summarise these findings in the pseudo-code in Algorithm 1. As the Hessian[10] of the Wasserstein distance is evaluated based on reference dynamics and any policy π, we pass it, along with ϵ and ϕ_0 as inputs. Then Algorithms 1 operates in a descent-ascent fashion in two main phases. In the first, lines 5 to 10 in Algorithm 1, dynamics parameters are updated using the closed-form solution in Equation 10, while ensuring that learning rates abide by a step size condition (we used the Wolfe conditions (Wolfe, 1969), though it can be some other method). With this, the second phase (line 11) utilises any state-of-the-art reinforcement learning method to adapt policy parameters generating $\boldsymbol{\theta}^{[k+1]}$.

Regarding the termination condition for the inner loop, we leave this as a decision for the user. It could be, for example, a large finite time-out, or the norm of the gradient $\boldsymbol{g}^{[k,j]}$ being below a threshold, or whichever happens first.

4 Zero'th Order Wasserstein Robust Reinforcement Learning

So far, we have presented an algorithm for robust reinforcement learning assuming accessibility to first and second-order information of the loss and constraint. It is relatively easy to attain such information if we were to follow a model-based setting that parameterises transitions with deep networks. As mentioned earlier, however, we follow another route that utilises a black-box optimisation scheme as deep neural networks are not always suitable for dynamical systems grounded in Lagrangian mechanics and physics (Lutter et al., 2019; Lutter and Peters, 2019).

This section details our zero-order robust solver, where we present an implementation of our approach for a scenario where training can be done on a simulator for which the dynamics are parameterised and can be altered at will. Whilst we refer to simulators (since much of RL training is done on such), almost exactly the same applies to differential equation solvers or other software based techniques for training a policy before deployment into the real world.

To elaborate, consider a simulator $\mathbb{S}_\phi$ for which the dynamics are parameterised by a real vector ϕ, and for which we can execute steps of a trajectory (i.e., the simulator takes as input an action

---

[9]**Remark:** Although this looks superficially similar to an approximation made in TRPO (Schulman et al., 2015a), the latter aims to optimise the *policy parameter* rather than dynamics. Furthermore, the constraint is based on the Kullback-Leibler divergence rather than the Wasserstein distance

[10]Please note our algorithm does not need to store the Hessian matrix. In line 7 of the algorithm, it is clear that we require Hessian-vector products. These can be easily computed using computational graphs without having access to the full Hessian matrix.

Algorithm 1: Wasserstein Robust Reinforcement Learning

1: **Inputs:** Wasserstein distance Hessian, H_0 evaluated at ϕ_0 under any policy π, radius of the Wasserstein ball ϵ, and the reference simulator specification parameters ϕ_0
2: Initialise $\phi^{[0]}$ with ϕ_0 and policy parameters $\theta^{[0]}$ arbitrarily
3: **for** $k = 0, 1, \ldots$ **do**
4:      $x^{[0]} \leftarrow \phi^{[0]}$
5:      $j \leftarrow 0$
6:      **Phase I: Update model parameter while fixing the policy:**
7:      **while** termination condition not met **do**
8:          Compute descent direction for the model parameters as given by Equation 10:

$$p^{[j]} \leftarrow \phi_0 - \sqrt{\frac{2\epsilon}{g^{[k,j]\top}H_0^{-1}g^{[k,j]}}}H_0^{-1}g^{[k,j]} - x^{[j]}$$

9:          Update candidate solution, while satisfying step size conditions (see discussion below) on the learning rate α:

$$x^{[j+1]} \leftarrow x^{[j]} + \alpha p^{[j]}$$

10:          $j \leftarrow j + 1$
11:      **end while**
12:      Perform model update setting $\phi^{[k+1]} \leftarrow x^{[j]}$
13:      **Phase II: Update policy given new model parameters:**
14:      Use any standard reinforcement learning algorithm for *ascending* in the gradient direction,

     e.g., $\theta^{[k+1]} \leftarrow \theta^{[k]} + \beta^{[k]}\nabla_\theta\mathbb{E}_{\tau \sim p_\theta^\phi(\tau)}\left[\mathcal{R}_{\text{total}}(\tau)\right]\Big|_{\theta^{[k]}, \phi^{[k+1]}}$, with $\beta^{[k]}$ is a policy learning rate.

15: **end for**
---

a and gives back a successor state and reward). For generating novel physics-grounded transitions, one can simply alter ϕ and execute the instruction in $\mathbb{S}_\phi$ from some a state $s \in \mathcal{S}$, while applying an action $a \in \mathcal{A}$. Not only does this ensure valid (under mechanics) transitions, but also promises scalability as specification parameters typically reside in lower dimensional spaces compared to the number of tuneable weights when using deep networks as transition models.

As we do not explicitly model transitions (e.g., the intricate operations of the simulator or differential equations solver), one has to tackle an additional challenge when requiring gradient or Hessian information to perform optimisation. Namely, if the idea of parameterising simulators through dynamic specifications in Phases I and II of Algorithm 1 is to be successfully executed, we require a procedure for estimating first and second-order information based on only function value evaluations of $\mathbb{S}_\phi$.

Next, we elaborate how one can acquire such estimates by proposing a novel zero-order method for estimating gradients and Hessians that we use in our experiments that demonstrate scalability, and robustness on high-dimensional robotic environments.

Gradient Estimation: Recalling the update rule in Phase I of Algorithm 1, we realise the need for, estimating the gradient of the loss function with respect to the vector specifying the dynamics of the environment, i.e., $g^{[k,j]} = \nabla_\phi\mathbb{E}_{\tau \sim p_\theta^\phi(\tau)}\left[\mathcal{R}_{\text{total}}(\tau)\right]\Big|_{\theta^{[k]}, \phi^{[j]}}$ at each iteration of the inner-loop j. Handling simulators as black-box models, we estimate the gradients by sampling from a Gaussian distribution with mean 0 and $\sigma^2 I$ co-variance matrix. Our choice for such estimates is not arbitrary but rather theoretically grounded as one can easily prove the following proposition:

Proposition 1 (Zero-Order Gradient Estimate)**.** *For a fixed $\boldsymbol{\theta}$ and ϕ, the gradient can be computed as:*

$$\nabla_\phi \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}^\phi(\boldsymbol{\tau})} \left[\mathcal{R}_{total}(\boldsymbol{\tau}) \right] = \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{\xi} \int_{\boldsymbol{\tau}} p_{\boldsymbol{\theta}}^{\phi+\boldsymbol{\xi}}(\boldsymbol{\tau}) \mathcal{R}_{total}(\boldsymbol{\tau}) d\boldsymbol{\tau} \right].$$

Proof. The proof of the above proposition can easily be derived by combining the lines of reasoning in (Salimans et al., 2017; Nesterov, 2011), while extending to the parameterisation of dynamical models. To commence, begin by defining $\mathcal{J}_{\boldsymbol{\theta}}(\phi) = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}^\phi} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right]$ for fixed policy parameters $\boldsymbol{\theta}$. Given any perturbation vector, $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$, we can derive (through a Taylor expansion) the following:

$$\mathcal{J}_{\boldsymbol{\theta}}(\phi + \boldsymbol{\xi}) = \mathcal{J}_{\boldsymbol{\theta}}(\phi) + \boldsymbol{\xi}^\mathsf{T} \nabla_\phi \mathcal{J}_{\boldsymbol{\theta}}(\phi) + \frac{1}{2} \boldsymbol{\xi}^\mathsf{T} \nabla_\phi^2 \mathcal{J}_{\boldsymbol{\theta}}(\phi) \boldsymbol{\xi} + \mathcal{O}\,(\text{higher-order terms})\,.$$

Multiplying by $\boldsymbol{\xi}$, and taking the expectation on both sides of the above equation, we get:

$$\mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{\xi} \mathcal{J}_{\boldsymbol{\theta}}(\phi + \boldsymbol{\xi}) \right] = \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{\xi} \mathcal{J}_{\boldsymbol{\theta}}(\phi) + \boldsymbol{\xi} \boldsymbol{\xi}^\mathsf{T} \nabla_\phi \mathcal{J}_{\boldsymbol{\theta}}(\phi) + \frac{1}{2} \boldsymbol{\xi} \boldsymbol{\xi}^\mathsf{T} \nabla_\phi^2 \mathcal{J}_{\boldsymbol{\theta}}(\phi) \boldsymbol{\xi} \right]$$

$$= \sigma^2 \nabla_\phi \mathcal{J}_{\boldsymbol{\theta}}(\phi).$$

Dividing by σ^2, we derive the statement of the proposition as:

$$\nabla_\phi \mathcal{J}_{\boldsymbol{\theta}}(\phi) = \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{\xi} \mathcal{J}_{\boldsymbol{\theta}}(\phi + \boldsymbol{\xi}) \right]$$

$\square$

Hessian Estimation: Having derived a zero-order gradient estimator, we now generalise these notions to a form allowing us to estimate the Hessian. It is also worth reminding the reader that such a Hessian estimator needs to be performed one time only before executing the instructions in Algorithm 1 (i.e., $\boldsymbol{H}_0$ is passed as an input). Precisely, we prove the following proposition:

Proposition 2 (Zero-Order Hessian Estimate)**.** *The hessian of the Wasserstein distance around ϕ_0 can be estimated based on function evaluations. Recalling that* $\boldsymbol{H}_0 = \nabla_\phi^2 \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim \pi(\cdot) \rho_\pi^{\phi_0}(\cdot)} \left[\mathcal{W}_2^2 \left(\mathcal{P}_\phi(\cdot|\boldsymbol{s},\boldsymbol{a}), \mathcal{P}_0(\cdot|\boldsymbol{s},\boldsymbol{a}) \right) \right] \Big|_{\phi=\phi_0}$, *and defining* $\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\phi) := \mathcal{W}_2^2 \left(\mathcal{P}_\phi(\cdot|\boldsymbol{s},\boldsymbol{a}), \mathcal{P}_0(\cdot|\boldsymbol{s},\boldsymbol{a}) \right)$, *we prove:*

$$\boldsymbol{H}_0 = \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})} \left[\frac{1}{\sigma^2} \boldsymbol{\xi} \left(\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim \pi(\cdot) \rho_\pi^{\phi_0}(\cdot)} \left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})} \left(\phi_0 + \boldsymbol{\xi} \right) \right] \right) \boldsymbol{\xi}^\mathsf{T} \right.$$

$$\left. - \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim \pi(\cdot) \rho_\pi^{\phi_0}(\cdot)} \left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\phi_0 + \boldsymbol{\xi}) \right] \boldsymbol{I} \right].$$

Proof. Commencing with the right-hand-side of the above equation, we perform second-order Taylor expansions for each of the two terms under under the expectation of $\boldsymbol{\xi}$. Namely, we write:

$$\boldsymbol{H}_0 = \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}\left[\frac{1}{\sigma^2}\boldsymbol{\xi}\left(\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}(\cdot)}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}\left(\boldsymbol{\phi}_0+\boldsymbol{\xi}\right)\right]\right)\boldsymbol{\xi}^\mathsf{T}\right. \tag{11}$$

$$\left.-\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}(\cdot)}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0+\boldsymbol{\xi})\right]\boldsymbol{I}\right]$$

$$\approx\frac{1}{\sigma^4}\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}\left[\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right]\boldsymbol{\xi}\boldsymbol{\xi}^\mathsf{T}+\boldsymbol{\xi}\boldsymbol{\xi}^\mathsf{T}\nabla_\phi\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right]\boldsymbol{\xi}^\mathsf{T}\right.$$

$$\left.+\frac{1}{2}\boldsymbol{\xi}^\mathsf{T}\nabla_\phi^2\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right]\boldsymbol{\xi}\boldsymbol{I}\right]$$

$$-\frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}\left[\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right]\boldsymbol{I}+\boldsymbol{\xi}^\mathsf{T}\nabla_\phi\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right]\boldsymbol{I}\right.$$

$$\left.+\frac{1}{2}\boldsymbol{\xi}^\mathsf{T}\nabla_\phi^2\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right]\boldsymbol{\xi}\boldsymbol{I}\right].$$

Now, we analyse each of the above terms separately. For ease of notation, we define the following variables:

$$\boldsymbol{g}=\nabla_\phi\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right] \qquad\qquad \boldsymbol{H}=\nabla_\phi^2\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\pi(\cdot)\rho_\pi^{\phi_0}}\left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\boldsymbol{\phi}_0)\right]$$

$$\boldsymbol{A}=\boldsymbol{\xi}\boldsymbol{\xi}^\mathsf{T}\boldsymbol{g}\boldsymbol{\xi}^\mathsf{T} \qquad\qquad\qquad\qquad\qquad \boldsymbol{B}=\boldsymbol{\xi}\boldsymbol{\xi}^\mathsf{T}\boldsymbol{H}\boldsymbol{\xi}\boldsymbol{\xi}^\mathsf{T}$$

$$c=\boldsymbol{\xi}^\mathsf{T}\boldsymbol{H}\boldsymbol{\xi}.$$

Starting with $\boldsymbol{A}$, we can easily see that any (i,j) component can be written as $\boldsymbol{A}=\sum_{n=1}^{d_2}\boldsymbol{\xi}_i\boldsymbol{\xi}_j\boldsymbol{\xi}_n\boldsymbol{g}_n$. Therefore, the expectation under $\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})$ can be derived as:

$$\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}\left[\boldsymbol{\xi}_i\boldsymbol{\xi}_j\boldsymbol{\xi}_n\boldsymbol{g}_n\right]=\boldsymbol{g}_n\mathbb{E}_{\boldsymbol{\xi}_i\sim\mathcal{N}(0,\sigma^2)}\left[\boldsymbol{\xi}_i^3\right]=0 \quad\text{if } i=j=n \text{ and } 0 \text{ otherwise.}$$

Thus, we conclude that $\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}[\boldsymbol{A}]=\boldsymbol{0}_{d_2\times d_2}$.

Continuing with the second term, i.e., $\boldsymbol{B}$, we realise that any (i,j) component can be written as $\boldsymbol{B}_{i,j}=\sum_{n=1}^{d_2}\sum_{m=1}^{d_2}\boldsymbol{\xi}_i\boldsymbol{\xi}_j\boldsymbol{\xi}_n\boldsymbol{\xi}_m\boldsymbol{H}_{m,n}$. Now, we consider two cases:

- Diagonal Elements (i.e., when $i=j$): The expectation under $\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})$ can be further split in three sub-cases

    - Sub-Case I when $i=j=m=n$: We have $\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(0,\sigma^2\boldsymbol{I})}\left[\boldsymbol{\xi}_i\boldsymbol{\xi}_j\boldsymbol{\xi}_n\boldsymbol{\xi}_m\boldsymbol{H}_{m,n}\right]=$ $\boldsymbol{H}_{i,i}\mathbb{E}_{\boldsymbol{\xi}_i\sim\mathcal{N}(0,\sigma^2)}=3\sigma^4\boldsymbol{H}_{i,i}$.

    - Sub-Case II when $i=j\neq m=n$: We have $\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(0,\sigma^2\boldsymbol{I})}\left[\boldsymbol{\xi}_i\boldsymbol{\xi}_j\boldsymbol{\xi}_n\boldsymbol{\xi}_m\boldsymbol{H}_{m,n}\right]=$ $\boldsymbol{H}_{m,m}\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}\left[\boldsymbol{\xi}_i^2\boldsymbol{\xi}_m^2\right]=\sigma^4\boldsymbol{H}_{m,m}$.

    - Sub-Case III when indices are all distinct: We have $\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(0,\sigma^2\boldsymbol{I})}\left[\boldsymbol{\xi}_i\boldsymbol{\xi}_j\boldsymbol{\xi}_n\boldsymbol{\xi}_m\boldsymbol{H}_{m,n}\right]=$ 0.

> **Diagonal Elements Conclusion:** Using the above results we conclude that $\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}[\boldsymbol{B}_{i,i}]=2\sigma^4\boldsymbol{H}_{i,i}+\sigma^4\text{trace}(\boldsymbol{H})$.

- Off-Diagonal Elements (i.e., when $i\neq j$): The above analysis is now repeated for computing the expectation of the off-diagonal elements of matrix $\boldsymbol{B}$. Similarly, this can also be split into three sub-cases depending on indices:

    - Sub-Case I when $i=m\neq j=n$: We have $\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(0,\sigma^2\boldsymbol{I})}\left[\boldsymbol{\xi}_i\boldsymbol{\xi}_j\boldsymbol{\xi}_n\boldsymbol{\xi}_m\boldsymbol{H}_{m,n}\right]=$ $\boldsymbol{H}_{i,j}\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})}\left[\boldsymbol{\xi}_i^2\boldsymbol{\xi}_j^2\right]=\sigma^4\boldsymbol{H}_{i,j}$.

11

- Sub-Case II when $i = n \neq j = m$: We have $\mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{\xi}_i \boldsymbol{\xi}_j \boldsymbol{\xi}_n \boldsymbol{\xi}_m \boldsymbol{H}_{m,n} \right] = \boldsymbol{H}_{j,i} \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{\xi}_i^2 \boldsymbol{\xi}_j^2 \right] = \sigma^4 \boldsymbol{H}_{j,i}$.

- Sub-Case III when indices are all distinct: We have $\mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{\xi}_i \boldsymbol{\xi}_j \boldsymbol{\xi}_n \boldsymbol{\xi}_m \boldsymbol{H}_{m,n} \right] = 0$.

> **Off-Diagonal Elements Conclusion:** Using the above results and due to the symmetric properties of $\boldsymbol{H}$, we conclude that $\mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})} \left[\boldsymbol{B}_{i,j} \right] = 2\sigma^4 \boldsymbol{H}_{i,j}$

Finally, analysing c, one can realise that $\mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})}[c] = \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})} \left[\sum_{i=1}^{d_2} \sum_{j=1}^{d_2} \boldsymbol{\xi}_i \boldsymbol{\xi}_j \boldsymbol{H}_{i,j} \right] = \sigma^2 \mathrm{trace}(\boldsymbol{H})$.

Substituting the above conclusions back in the original approximation in Equation 11, and using the linearity of the expectation we can easily achieve the statement of the proposition. $\square$

With the above two propositions, we can now perform the updates in Algorithm 1 without the need for performing explicit model learning. This is true as Propositions 6 and 7 devise procedure where gradient and Hessian estimates can be simply based on simulator value evaluations while perturbing ϕ and ϕ_0. It is important to note that in order to apply the above, we are required to be able to evaluate $\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a}) \sim \pi(\cdot)\rho_\pi^{\phi_0}(\cdot)} \left[\mathcal{W}_{(\boldsymbol{s},\boldsymbol{a})}(\phi_0) \right]$ under random $\boldsymbol{\xi}$ perturbations sampled from $\mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$. An empirical estimate of the p-Wasserstein distance between two measures μ and ν can be performed by computing the p-Wasserstein distance between the empirical distributions evaluated at sampled data. That is, one can approximation μ by $\mu_n = \frac{1}{n} \sum_{i=1}^n \delta_{\boldsymbol{x}_i}$ where $\boldsymbol{x}_i$ are identically and independently distributed according to μ. Approximating ν_n similarly, we then realise that[11] $\mathcal{W}_2(\mu, \nu) \approx \mathcal{W}_2(\mu_n, \nu_n)$.

5 Related work

In this section we review some of the related literature. There is a common theme running through previous works: the formulation of the problem as a max-min (or min-max, depending on whether the goal is to maximise for rewards or minimise for costs) problem. This game-theoretic view is natural formulation when viewing nature as an adversary. Thus, it will be no surprise that the papers discussed below mainly take this approach or close variations of it.

There is a long-standing thread of research on robustness in the classical control community, and the literature in this area is vast, with the $\mathcal{H}_\infty$ method being a standard approach (Doyle et al., 2013). This approach was introduced into reinforcement learning by Morimoto and Doya (2005). In that paper, a continuous time reinforcement learning setting was studied for which a max-min problem was formulated involving a modified value function, the optimal solutions of which can be determined by solving Hamilton-Jacobi-Isaacs (HJI) equation.

There is also a line of work on robust MDPs, amongst which are Iyengar (2005); Nilim and El Ghaoui (2005); Wiesemann et al. (2013); Tirinzoni et al. (2018); Petrik and Russell (2019). In particular, Yang (2017) uses the Wasserstein distance to define uncertainty sets of dynamics in similar way to this work, that is, in an ϵ-ball around a particular dynamics (referred to as *nominal* distribution in that paper). The paper shows that an optimal Markov control policy is possible the max-min Bellman equations and shows how convex-optimisation techniques can be applied to solve it.

Whilst valuable in their own right, these approaches are not sufficient for the RL setting due to the need in the latter case to give efficient solutions for large state and action spaces, and the fact that the dynamics are not known *a priori*. We emphasise once again that in our setting, cannot explicitly define the MDP of the reference dynamics, since we do not assume knowledge of it. We assume only that we can sample from it, which is the standard assumption made in RL.

---

[11] In case the dynamics are assumed to be Gaussian, a similar procedure can be followed or a closed form can be used, see Takatsu (2008).

Reasonably, one might expect that model-based reinforcement learning may be a plausible route to address robustness. In Asadi et al. (2018), the learning of Lipschitz continuous models is addressed, and a bound on multi-step prediction error is given in terms of the Wasserstein distance. However, the major stumbling-block with model-based RL techniques is that in high-dimensional state building models that are sufficient for controlling an agent can suffer greatly from model misspecification or excessive computational costs (e.g., as in training a Gaussian Process, see, e.g., Rasmussen (2003); Deisenroth and Rasmussen (2011)).

We now discuss some papers closer in objective and/or technique to our own. Rajeswaran et al. (2017) approaches the robustness problem by training on an ensemble of dynamics in order to be deployed on a target environment. The algorithm introduced, Ensemble Policy Optimisation (EPOpt), alternates between two phases: (i) given a distribution over dynamics for which simulators (or models) are available (the source domain), train a policy that performs well for the whole distribution; (ii) gather data from the deployed environment (target domain) to adapt the distribution. The objective is not max-min, but a softer variation defined by conditional value-at-risk (CVaR). The algorithm samples a set of dynamics $\{\phi_k\}$ from a distribution over dynamics $\mathcal{P}_\psi$, and for each dynamics ϕ_k, it samples a trajectory using the current policy parameter θ_i. It then selects the worst performing ϵ-fraction of the trajectories to use to update the policy parameter. Clearly this process bears some resemblance to our algorithm, but there is a crucial difference: our algorithm takes *descent steps* in the ϕ space. The difference if important when the dynamics parameters sit in a high-dimensional space, since in that case, optimisation-from-sampling could demand a considerable number of samples. A counter argument against our technique might be that our zero'th-order method for estimating gradients and Hessians also requires sampling in high dimensions. This is, indeed the case, but obtaining localised estimates (as gradients and Hessians are local properties) could be easier than global properties (the worse set of parameters in the high-dimensional space). In any case, our experiments demonstrate our algorithm performs well even in these high dimensions. The experiments of Rajeswaran et al. (2017) are on Hopper and HalfCheetah, in which their algorithm is compared to TRPO. We note that we were were unable to find the code for this paper, and did not attempt to implement it ourselves.

The CVaR criterion is also adopted in Pinto et al. (2017), in which, rather than sampling trajectories and finding a quantile in terms of performance, two policies are trained simultaneously: a "protagonist" which aims to optimise performance, and an adversary which aims to disrupt the protagonist. The protagonist and adversary train alternatively, with one being fixed whilst the other adapts. The action space for the adversary, in the tests documented in the paper includes forces on the entities (InvertedPendulum, HalfCheetah, Swimmer, Hopper, Walker2D) that aim to destabalise it. We made comparisons against this algorithm in our experiments.

More recently, Tessler et al. (2019) studies robustness with respect to action perturbations. There are two forms of perturbation addressed: (i) Probabilistic Action Robust MDP (PR-MDP), and (ii) Noisy Action Robust MDP (NR-MDP). In PR-MDP, when an action is taken by an agent, with probability α, a different, possibly adversarial action is taken instead. In NR-MDP, when an action is taken, a perturbation is added to the action itself. Like Rajeswaran et al. (2017) and Pinto et al. (2017), the algorithm is suitable for applying deep neural networks, and the paper reports experiments on InvertedPendulum, Hopper, Walker2d and Humanoid. We tested against PR-MDP in some of our experiments, and found it to be lacking in robustness (see Section 6, Figure 1 and Figure 2).

In Lecarpentier and Rachelson (2019) a non-stationary Markov Decision Process model is considered, where the dynamics can change from one time step to another. The constraint is based on Wasserstein distance, specifically, the Wasserstein distance between dynamics at time t and t' is bounded by $L|t - t'|$, i.e., is L-Lipschitz with respect to time, for some constant L. They approach the problem by treating nature as an adversary and implement a Minimax algorithm. The basis of their algorithm is that due to the fact that the dynamics changes slowly (due to the Lipschitz constraint), a planning algorithm can project into the future the scope of possible future dynamics and plan for the worst. The resulting algorithm, known as *Risk Averse Tree Search*, is - as the name implies - a tree search algorithm. It operates on a sequence "snapshots" of the evolving MDP, which are instances of the MDP at points in time. The algorithm is tested on small grid world, and does not appear to be readily extendible to the continuous state and action scenarios our algorithm addresses.

To summarise, our paper uses the Wasserstein distance for quantifying variations in possible dynamics, in common with Lecarpentier and Rachelson (2019), but is suited to applying deep neural

networks for continuous state and action spaces. Our algorithm does not require a full dynamics available to it, merely a parameterisable dynamics. It competes well with the above papers, and operates well for high dimensional problems, as evidenced by the experiments.

6 Experiments & Results

We evaluate WR^2L on a variety of continuous control benchmarks from the MuJoCo environment. Dynamics in our benchmarks were parameterised by variables defining physical behaviour, e.g., density of the robot's torso, friction of the ground, and so on. We consider both low and high dimensional dynamics and demonstrate that our algorithm outperforms state-of-the-art from both standard and robust reinforcement learning. We are chiefly interested in policy generalisation across environments with varying dynamics, which we measure using average test returns on novel systems. The comparison against standard reinforcement learning algorithms allows us to understand whether lack of robustness is a critical challenge for sequential decision making, while comparisons against robust algorithms test if we outperform state-of-the-art that considered a similar setting to ours. From standard algorithms, we compare against proximal policy optimisation (PPO) (Schulman et al., 2017), and trust region policy optimisation (TRPO) (Schulman et al., 2015b); an algorithm based on natural actor-crtic (Peters and Schaal, 2008a; Pajarinen et al., 2019). From robust algorithms, we demonstrate how WR^2L favours against robust adversarial reinforcement learning (RARL) (Pinto et al., 2017), and action-perturbed Markov decision processes (PR-MDP) proposed in (Tessler et al., 2019).

It is worth noting that we attempted to include deep deterministic policy gradients (DDPG) (Silver et al., 2014) in our comparisons. Results including DDPG were, however, omitted as it failed to show any significant robustness performance even on relatively simple systems, such as the inverted pendulum; see results reported in Appendix A. During initial trials, we also performed experiments parameterising models using deep neural networks. Results demonstrated that these models, though minimising training data error, fail to provide valid physics-grounded dynamics. For instance, we arrived at inverted pendula models that vary pole angles without exerting any angular speeds. This problem became even more apparent in high-dimensional systems, e.g., Hopper, Walker, etc due to the increased number of possible minima. As such, results presented in this section make use of our zero-order method that can be regarded as a scalable alternative for robust solutions.

6.1 MuJoCo benchmarks

Contrary to other methods rooted in model-based reinforcement learning, we evaluate our method both in low and *high-dimensional* MuJuCo tasks (Brockman et al., 2016). We consider a variety of systems including CartPole, Hopper, and Walker2D; all of which require direct joint-torque control. Keeping with the generality of our method, we utilise these dynamical as-is with no additional alterations. Namely, we use the exact setting of these benchmarks as that shipped with OpenAI gym without any reward shaping, state-space augmentation, feature extraction, or any other modifications of-that-sort. For clarity, we summarise variables parameterising dynamics in Table 1, and detail specifics next.

CartPole: The goal of this classic control benchmark is to balance a pole by driving a cart along a rail. The state space is composed of the position x and velocity $\dot{x}$ of the cart, as well as the angle θ and angular velocities of the pole $\dot{\theta}$. We consider two termination conditions in our experiments: 1) pole deviates from the upright position beyond a pre-specified threshold, or 2) cart deviates from its zeroth initial position beyond a certain threshold. To conduct robustness experiments, we parameterise the dynamics of the CartPole by the pole length l_p, and test by varying $l_p \in [0.3, 3]$.

Hopper: In this benchmark, the agent is required to control a hopper robot to move forward without falling. The state of the hopper is represented by positions, $\{x, y, z\}$, and linear velocities, $\{\dot{x}, \dot{y}, \dot{z}\}$, of the torso in global coordinate, as well as angles, $\{\theta_i\}_{i=0}^2$, and angular speeds, $\{\dot{\theta}_i\}_{i=0}^2$, of the three joints. During training, we exploit an early-stopping scheme if "unhealthy" states of the robot were visited. Parameters characterising dynamics included densities $\{\rho_i\}_{i=0}^3$ of the four links, armature $\{a_i\}_{i=0}^2$ and damping $\{\zeta_i\}_{i=0}^2$ of three joints, and the friction coefficient μ_g. To test for robustness, we varied both frictions and torso densities leading to significant variations in dynamics. We further conducted additional experiments while varying all 11 dimensional specification parameters.

	1D experiment	2D experiment	High-dimensional experiment
Inverted Pendulum	l_p	None	None
Hopper	ρ_0	$\{\rho_0, \mu_g\}$	$\{\rho_i\}_{i=0}^{3} \cup \{a_i\}_{i=0}^{2} \cup \{\zeta_i\}_{i=0}^{2} \cup \mu_g$
Walker2D	ρ_0	$\{\rho_0, \mu_g\}$	$\{\rho_i\}_{i=0}^{6} \cup \{a_i\}_{i=0}^{5} \cup \{\zeta_i\}_{i=0}^{5} \cup \mu_g$
HalfCheetah	None	$\{\rho_0, \mu_g\}$	$\{\rho_i\}_{i=0}^{7} \cup \{a_i\}_{i=0}^{5} \cup \{\zeta_i\}_{i=0}^{5} \cup \mu_g$

Table 1: Parameterisation of dynamics. See section 6.1 for the physical meaning of these parameters.

Walker2D: This benchmark is similar to Hopper except that the controlled system is a biped robot with seven bodies and six joints. Dimensions for its dynamics are extended accordingly as reported in Table 1. Here, we again varied the torso density for performing robustness experiments in the range $\rho_0 \in [500, 3000]$.

Halfcheetah: This benchmark is similar to the above except that the controlled system is a two-dimensional slice of a three-dimensional cheetah robot. Parameters specifying the simulator consist of 21 dimensions, with 7 representing densities. In our two-dimensional experiments we varied the torso-density and floor friction, while in high-dimensional ones, we allowed the algorithm to control all 21 variables.

6.2 Experimental protocol

Our experiments included training and a testing phases. During the training phase we applied Algorithm 1 for determining robust policies while updating transition model parameters according to the min-max formulation. Training was performed independently for each of the algorithms on the relevant benchmarks while ensuring best operating conditions using hyper-parameter values reported elsewhere (Schulman et al., 2017; Pinto et al., 2017; Tessler et al., 2019).

For all benchmarks, policies were represented using parametrised Gaussian distributions with their means given by a neural network and standard derivations by a group of free parameters. The neural network consisted of two hidden layers with 64 units and hyperbolic tangent activations in each of the layers. The final layer exploited linear activation so as to output a real number. Following the actor-critic framework, we also trained a standalone critic network having the same structure as that of the policy.

For each policy update, we rolled-out in the current worst-case dynamics to collect a number of transitions. The number associated to these transitions was application-dependent and varied between benchmarks in the range of 5,000 to 10,000. The policy was then optimised (i.e., Phase II of Algorithm 1) using proximal policy optimization with a generalised advantage estimation. To solve the minimisation problem in the inner loop of Algorithm 1, we sampled a number of dynamics from a diagonal Gaussian distribution that is centered at the current worst-case dynamics model. The number of sampled dynamics and the variance of the sampled distributions depended on both the benchmark itself, and well as the dimensions of the dynamics. Gradients needed for model updates were estimated using the results in Propositions 7 and 8. Finally, we terminated training when the policy entropy dropped below an application-dependent threshold.

When testing, we evaluated policies on unseen dynamics that exhibited simulator variations as described earlier. We measured performance using returns averaged over 20 episodes with a maximum length of 1,000 time steps on testing environments. We note that we used non-discounted mean episode rewards to compute such averages.

6.3 Comparison Results & Benchmarking

This section summarises robustness results showing that our method significantly outperforms others from both standard and robust reinforcement learning in terms of average testing returns as dynamics vary.

Results with One-Dimensional Model Variation: Figure 1 shows the robustness of policies on a simple inverted pendulum while varying the pole length in the ranges from 0.3 to

3.0. For a fair comparison, we trained two standard policy gradient methods (TRPO (Schulman et al., 2015b) and PPO (Schulman et al., 2017)), and two robust RL algorithms (RARL (Pinto et al., 2017), PR-MDP (Tessler et al., 2019)) with the reference dynamics preset by our algorithm. The range of evaluation parameters was intentionally designed to include dynamics out of the ϵ-Wasserstein ball. Clearly, WR^{2}L outperforms all baselines in this benchmark.

Given successful behaviour in low-dimensional state representations, we performed additional experiments on the Hopper and Walker systems to assess robustness against model changes in high-dimensional environments. Figure 2 illustrates these results depicting that our method is again capable of outperforming others including RARL and PR-MDP. It is also interesting to realise that in high-dimensional environments, our algorithm exhibits a trade-off between robustness and optimality due to the min-max definition of WR^{2}L's objective.

Figure 1: Robustness results on the inverted pendulum demonstrating that our method outperforms state-of-the-art in terms of average test returns.

Experimental Conclusion I: From the above, we conclude that WR^{2}L outperforms others when one-dimensional simulator variations are considered.

Figure 2: Robustness results on Hopper (left) and Walker (right) systems demonstrating that our method outperforms others significantly in terms of average test returns as torso densities vary. It is also interesting to realise that due to the robust problem formulation, our algorithm exhibits a trade-off between optimality and generalisation. Hopper results are with a reference $\rho_0 = 1750$; PPO$_2$ uses the same implementation as PPO but trained with $\rho_0 = 3000$. Walker results are attained with a reference model of $\rho_0 = 1750$.

Furthermore, one would expect that such advantages increase with the increase in the radius ϵ. To validate these claims, we re-ran the same experiment devised above while allowing for a larger ϵ of 0.015. It is clear from Figure 3(c) that the robustness range of the policy generated by WR^{2}L does increase with the increase in the ball's radius.

These results were also verified in two additional benchmarks (i.e., the Walker and HalfCheetah). Here, again our results demonstrate that when 2 dimensional changes are considered, our method outperforms state-of-the-art significantly. We also arrive at the same conclusion that if ϵ increases so does the robustness range. For instance, a robust policy trained with an ϵ of 0.03 achieves high average test returns on a broader range of HalfCheetahs compares to that with an $\epsilon = 0.005$, see Figures 4 (h) and 4 (i).

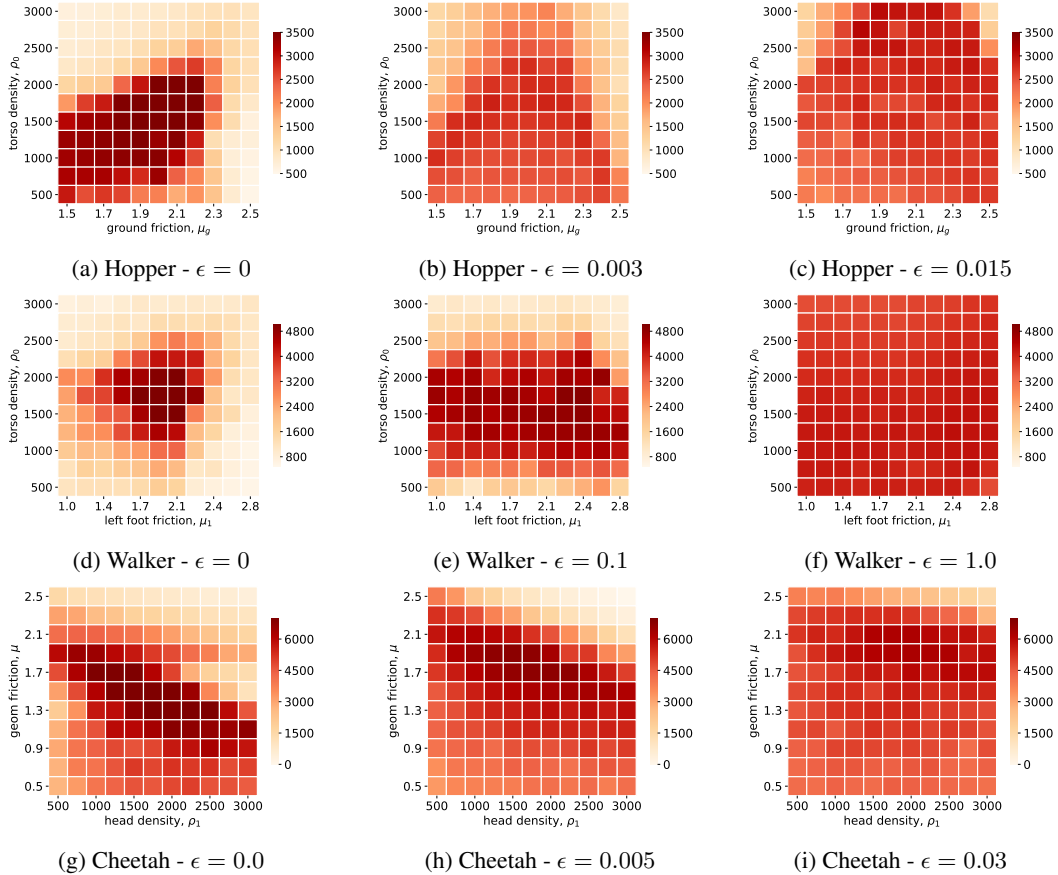This, in turn, takes us to the following conclusion:

Figure 3: Results on various benchmarks. Top row represents Hopper results, middle is concerned with Walker, and bottom denotes HalfCheetah. These graphs depict test returns as a function of changes in dynamical parameters and Wasserstein distance. These graphs again show that WR^{2}L outperforms PPO (i.e., when $\epsilon = 0$) and that its robustness improves as ϵ increases.

Experimental Conclusion II: From the above, we conclude that WR^{2}L outperforms others when two-dimensional simulator variations are considered, and that robustness increase with ϵ.

Results with High-Dimensional Model Variation: Though results above demonstrate robustness, an argument against a min-max objective can be made especially when only considering low-dimensional changes in the simulator. Namely, one can argue the need for such an objective as opposed to simply sampling a set of systems and determining policies performing-well on average similar to the approach proposed in (Rajeswaran et al., 2017).

A counter-argument to the above is that a gradient-based optimisation scheme is more efficient than a sampling-based one when high-dimensional changes are considered. In other words, a sampling procedure is hardly applicable when more than a few parameters are altered, while WR^{2}L can remain suitable. To assess these claims, we conducted two additional experiments on the Hopper and HalfCheetah benchmarks. In the first, we trained robustly while changing friction and torso densities, and tested on 1000 systems generated by varying all 11 dimensions of the Hopper dynamics, and 21 dimensions of the HalfCheetah system. Results reported in Figures 4(b) and (e) demonstrate that the empirical densities of the average test returns are mostly centered around 3000 for the Hopper, and around 4500 for the Cheetah, which improves that of PPO (Figures 4(a) and (d)) with return masses mostly accumulated at around 1000 in the case of the Hopper and almost equally distributed when considering HalfCheetah.
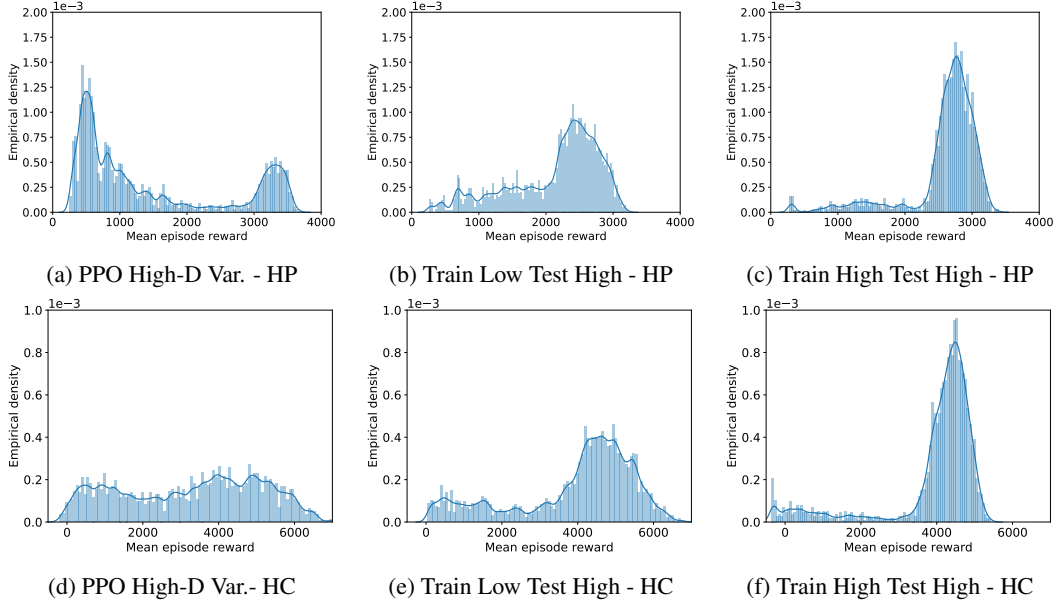
17

Figure 4: Results evaluating performance when considering high-dimensional variations on the hopper (HP - top row) and HalfCheetah (HC - bottom row) environment. All figures show the empirical distribution of returns on 1,000 testing systems. Figure (a) demonstrates the robustness of PPO. Figure (b) reports empirical test returns of WR^{2}L's policy trained on only two parameter changes (e.g., friction and density) of the environment but tested on systems with all high-dimensional dynamical parameters modified. Figure (c) trains and tests WR^{2}L altering all dimensional parameters of the simulator. Clearly, our method exhibits robustness even if high-dimensional variations were considered.

Such improvements, however, can be an artifact of the careful choice of the low-dimensional degrees of freedom allowed to be modified during Phase I of Algorithm 1. To get further insights, Figures 4(c) and (f) demonstrate the effectiveness of our method trained and tested while allowing to tune all 11 dimensional parameters of the Hopper simulator, and the 21 dimensions of the HalfCheetah. Indeed, our results are in accordance with these of the previous experiment depicting that most of the test returns' mass remains around 3000 for the Hopper, and improves to accumulate around 4500 for the HalfCheetah. Interestingly, however, our algorithm is now capable of acquiring higher returns on all systems[12] since it is allowed to alter all parameters defining the simulator. As such, we conclude:

Experimental Conclusion III: From the above, we conclude that WR^{2}L outperforms others when high-dimensional simulator variations are considered.

---

7 Conclusion & Future Work

In this paper, we proposed a novel robust reinforcement learning algorithm capable of outperforming others in terms of test returns on unseen dynamical systems. Our algorithm formalises a new min-max objective with Wasserstein constraints for policies generalising across varying domains, and considers a zero-order method for scalable solutions. Empirically, we demonstrated superior performance against state-of-the-art from both standard and robust reinforcement learning on low and high-dimensional MuJuCo environments.

---

[12]Please note that we attempted to compare against Rajeswaran et al. (2017). Due to the lack of open-source code, we were not able to regenerate their results.

There are a lot of interesting directions we plan to target in the future. First, we aim to consider robustness in terms of other components of MDPs, e.g., state representations, reward functions, and others. Second, we will implement WR^{2}L on real hardware, considering sim-to-real experiments.

8 Acknowledgements

A Deep Deterministic Policy Gradients Results

As mentioned in the experiments section of the main paper, we refrained from presenting results involving deep deterministic policy gradients (DDPG) due to its lack in robustness even on simple systems, such as the CartPole.
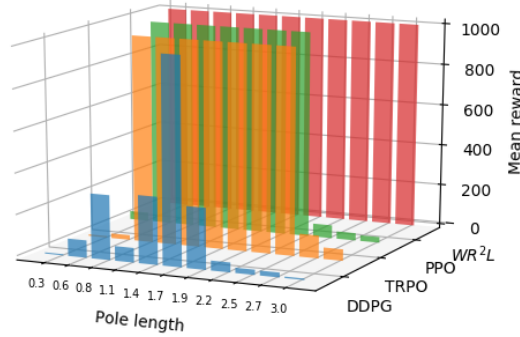

Figure 5: Robustness results on the inverted pendulum demonstrating that our method outperforms state-of-the-art in terms of average test returns and that DDPG lacks in robustness performance.

Figure 5 depicts these results showing that DDPG lacks robustness even when minor variations in the pole length are introduced. TRPO and PPO, on the other hand, demonstrate an acceptable performance retaining a test return of 1,000 across a broad range of pole lengths variations.

B Derivation of the Closed Form Solution

In Section 3.2 we presented a closed form solution to the following optimisation problem:

$$\min_{\phi} \nabla_{\phi} \mathbb{E}_{\tau \sim p_{\theta}^{\phi}(\tau)} \left[\mathcal{R}_{\text{total}}(\tau) \right] \Big|_{\theta^{[k]}, \phi^{[j]}}^{\mathsf{T}} (\phi - \phi^{[j]}) \text{ s.t. } \frac{1}{2}(\phi - \phi_0)^{\mathsf{T}} H_0(\phi - \phi_0) \leq \epsilon,$$

which took the form of:

$$\phi^{[j+1]} = \phi_0 - \sqrt{\frac{2\epsilon}{g^{[k,j]\mathsf{T}} H_0^{-1} g^{[k,j]}}} H_0^{-1} g^{[k,j]}.$$

In this section of the appendix, we derive such an update rule from first principles. We commence transforming the constraint optimisation problem into an unconstrained one using the Lagrangian:

$$\mathcal{L}(\phi, \lambda) = g^{[k,j],\mathsf{T}} \left(\phi - \phi^{[j]} \right) + \lambda \left[\frac{1}{2}(\phi - \phi_0)^{\mathsf{T}} H_0(\phi - \phi_0) - \epsilon \right],$$

19

where $\boldsymbol{\lambda}$ is a Lagrange multiplier, and $\boldsymbol{g}^{[k,j]} = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}^{\boldsymbol{\phi}}(\boldsymbol{\tau})} \left[\mathcal{R}_{\text{total}}(\boldsymbol{\tau}) \right] \Big|_{\boldsymbol{\theta}^{[k]}, \boldsymbol{\phi}^{[j]}}^{\mathsf{T}}$.

Deriving the Lagrangian with respect to the primal parameters $\boldsymbol{\phi}$, we write:

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\lambda}) = \boldsymbol{g}^{[k,j]\mathsf{T}} + \boldsymbol{\lambda}(\boldsymbol{\phi} - \boldsymbol{\phi}_0)^{\mathsf{T}} \boldsymbol{H}_0. \tag{12}$$

Setting Equation 12 to zero and solving for primal parameters, we attain:

$$\boldsymbol{\phi} = \boldsymbol{\phi}_0 - \frac{1}{\boldsymbol{\lambda}} \boldsymbol{H}_0^{-1} \boldsymbol{g}^{[k,j]}.$$

Plugging $\boldsymbol{\phi}$ back into the equation representing the constraints, we derive:

$$\left(\boldsymbol{\phi}_0 - \frac{1}{\boldsymbol{\lambda}} \boldsymbol{H}_0^{-1} \boldsymbol{g}^{[k,j]} - \boldsymbol{\phi}_0 \right)^{\mathsf{T}} \boldsymbol{H}_0 \left(\boldsymbol{\phi}_0 - \frac{1}{\boldsymbol{\lambda}} \boldsymbol{H}_0^{-1} \boldsymbol{g}^{[k,j]} - \boldsymbol{\phi}_0 \right) = 2\epsilon \implies \boldsymbol{\lambda}^2 = \frac{1}{2\epsilon} \boldsymbol{g}^{[k,j]\mathsf{T}} \boldsymbol{H}_0^{-1} \boldsymbol{g}^{[k,j]}.$$

It is easy to see that with the positive solution for $\boldsymbol{\lambda}$, the Karush–Kuhn–Tucker (KKT) conditions are satisfied. Since the objective and constraint are both convex, the KKT conditions are sufficient and necessary for optimality, thus finalising our derivation.

References

Asadi, K., Misra, D., and Littman, M. (2018). Lipschitz continuity in model-based reinforcement learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 264–273, Stockholmsmässan, Stockholm Sweden. PMLR.

Bou-Ammar, H., Eaton, E., Ruvolo, P., and Taylor, M. E. (2014). Online multi-task learning for policy gradient methods. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1206–II–1214. JMLR.org.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Busoniu, L., Babuska, R., Schutter, B. D., and Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc.

Chow, Y., Tamar, A., Mannor, S., and Pavone, M. (2015). Risk-sensitive and robust decision-making: a cvar optimization approach. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1522–1530. Curran Associates, Inc.

Deisenroth, M., Peters, J., and Neumann, G. (2013). A survey on policy search for robotics. *Found. Trends Robot*, 2:1–142.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Doyle, J. C., Francis, B. A., and Tannenbaum, A. R. (2013). *Feedback control theory*. Courier Corporation.

Emmert-Streib, F. and Dehmer, M. (2018). A machine learning perspective on personalized medicine: An automized, comprehensive knowledge base with ontology for pattern recognition. *Machine Learning and Knowledge Extraction*, 1(1):149–156.

Fischer, T. G. (2018). Reinforcement learning in financial markets - a survey. FAU Discussion Papers in Economics 12/2018, Erlangen.

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

Kober, J. and Peters, J. (2012). *Reinforcement Learning in Robotics: A Survey*, volume 12, pages 579–610. Springer, Berlin, Germany.

Koller, T., Berkenkamp, F., Turchetta, M., and Krause, A. (2018). Learning-based model predictive control for safe exploration and reinforcement learning. *CoRR*, abs/1803.08287.

Lecarpentier, E. and Rachelson, E. (2019). Non-stationary markov decision processes a worst-case approach using model-based reinforcement learning. *arXiv preprint arXiv:1904.10090*.

Lutter, M. and Peters, J. (2019). Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems. In *International Conference on Intelligent Robots and Systems (IROS)*.

Lutter, M., Ritter, C., and Peters, J. (2019). Deep lagrangian networks: Using physics as model prior for deep learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Morimoto, J. and Doya, K. (2005). Robust reinforcement learning. *Neural Comput.*, 17(2):335–359.

Namkoong, H. and Duchi, J. C. (2016). Stochastic gradient methods for distributionally robust optimization with f-divergences. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2216–2224, USA. Curran Associates Inc.

Nesterov, Y. (2011). Random gradient-free minimization of convex functions. CORE Discussion Papers 2011001, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE).

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

Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., and Song, D. (2018). Assessing generalization in deep reinforcement learning. *CoRR*, abs/1810.12282.

Pajarinen, J., Thai, H. L., Akrour, R., Peters, J., and Neumann, G. (2019). Compatible natural gradient policy search. *CoRR*, abs/1902.02823.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2017). Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537.

Peters, J. and Schaal, S. (2008a). Natural actor-critic. *Neurocomput.*, 71(7-9):1180–1190.

Peters, J. and Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Petrik, M. and Russell, R. H. (2019). Beyond confidence regions: Tight bayesian ambiguity sets for robust mdps. *arXiv preprint arXiv:1902.07605*.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826, International Convention Centre, Sydney, Australia. PMLR.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. (2017). Epopt: Learning robust neural network policies using model ensembles. *International Conference on Learning Representations (ICLR) 2017, arXiv preprint arXiv:1610.01283*.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Sargent, T. and Hansen, L. (2001). Robust control and model uncertainty. *American Economic Review*, 91(2):60–66.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France. PMLR.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015b). Trust region policy optimization. *CoRR*, abs/1502.05477.

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 *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395. JMLR.org.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Takatsu, A. (2008). Wasserstein geometry of gaussian measures.

Tessler, C., Efroni, Y., and Mannor, S. (2019). Action robust reinforcement learning and applications in continuous control. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6215–6224, Long Beach, California, USA. PMLR.

Tirinzoni, A., Petrik, M., Chen, X., and Ziebart, B. (2018). Policy-conditioned uncertainty sets for robust markov decision processes. In *Advances in Neural Information Processing Systems*, pages 8939–8949.

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

Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235.

Yang, I. (2017). A convex optimization approach to distributionally robust markov decision processes with wasserstein distance. *IEEE control systems letters*, 1(1):164–169.

Zhao, C., Sigaud, O., Stulp, F., and Hospedales, T. M. (2019). Investigating generalisation in continuous deep reinforcement learning. *CoRR*, abs/1902.07015.