Policy Optimization in Robust Markov Decision Processes with Transition Gradient Theorem

A

Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

in partial fulfillment

of the requirements for the degree

Master of Science

by

Licheng Luo

December 2024

APPROVAL SHEET

This

Thesis

is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

Author: Licheng Luo

This Thesis has been read and approved by the examing committee:

Advisor:	Shangtong Zhang
Advisor:	
Committee Member:	Chen-Yu Wei
Committee Member:	Yen-Ling Kuo
Committee Member:	
Committee Member:	
Committee Member:	
Committee Member:	

Accepted for the School of Engineering and Applied Science:

[illegible]

Jennifer L. West, School of Engineering and Applied Science

December 2024

Abstract

Reinforcement Learning (RL) is a powerful framework for sequential decision making. However, standard RL methods often struggle when the environment dynamics are uncertain, leading to poor performance in real-world applications such as autonomous navigation, financial portfolio management, and robotic control. This limitation is a significant factor contributing to the lack of widespread adoption of RL-based control systems in industry.

To address this challenge, researchers introduced the robust Markov Decision Process (MDP), a sequential decision-making framework that explicitly models uncertainty in transition functions. Robust MDP aims to find a policy that consistently performs well across a range of possible transition functions. It has great potential for application in various domains, where the environment dynamics are uncertain or changing.

In this thesis, we model a robust MDP as a two-player game. The first player represents the policy, trained via standard policy optimization methods. The second player is an adversary that selects transition functions aimed at deteriorating the performance of the policy. A key contribution of this work is the transition gradient theorem, which enables effective training of the adversary by providing a structured way to optimize the transition functions. The two players are updated in an alternating fashion.

We validate the proposed approach in simple environments to demonstrate robustness and then scale up to complex robotic manipulation tasks. Our findings showcase the scalability and efficacy of robust MDP methods in handling real-world uncertainties, highlighting their potential for practical applications.

To my parents.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Shangtong Zhang, for his invaluable guidance, encouragement, and support throughout the course of my research. His insights and expertise have been instrumental in shaping this work, and his patience and understanding have made this journey truly rewarding.

I am also grateful to my committee members, Prof. Chen-Yu Wei and Prof. Yen-Ling Kuo, for their constructive feedback and thoughtful suggestions, which greatly enhanced the quality of this thesis.

I would like to extend my thanks to my friends and Labmates Jiuqi Wang, Kefan Song, Lei Gong, Chen Gong, Zhiyan Tang, Make Li and many others, for their unwavering support and encouragement. Their friendship has been a constant source of inspiration and motivation.

In the end, I am profoundly thankful to my parents for their unwavering love and support. Their belief in me has been a constant source of motivation throughout my academic journey. I dedicate this thesis to them, with all my love and gratitude.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, I will commence with a concise yet comprehensive exploration of the history of machine learning (ML) and reinforcement learning (RL), shedding light on the pivotal milestones that have profoundly influenced their evolution. Then I will demonstrate our key contributions. In the end, I will provide an overview of the thesis structure.

1.1 Background of ML and RL

Reinforcement Learning (RL), a distinct branch of ML, takes a different approach by learning through interaction with an environment. Unlike supervised learning, RL relies on trial and error, where an agent makes decisions in a sequential manner, aiming to maximize cumulative rewards over time. This is formulated using the Markov Decision Process (MDP) framework (Bellman, 1957; Sutton & Barto, 2018). The agent observes the current state, selects an action, and receives feedback in the form of a reward, updating its policy to optimize future actions based on this feedback loop.

RL has demonstrated remarkable (even beyond-human) success in applications such as robotics, games, and control systems (Garcia & Fernandez, 2015; Kaelbling et al., 1996; Schrittwieser et al., 2020). However, one major limitation of RL is its sensitivity to environmental changes, such as observation errors, policy fluctuations, or dynamic uncertainties (Amodei et al., 2016).

1.2 Robust Reinforcement Learning

To mitigate these issues, **Robust Reinforcement Learning (Robust RL)** has been developed to ensure that the agent can maintain high performance even in uncertain environments (Iyengar, 2005;

Morimoto & Doya, 2005a; Nilim & El Ghaoui, 2005). The uncertainties in RL can be categorized into three main types (Kuang et al., 2022): *observation uncertainty*, which arises from sensor noise or measurement errors; *policy uncertainty*, referring to the inherent randomness or ambiguity in policy decisions; and *dynamics uncertainty*, where the state transition model itself is unknown or changes unpredictably.

In recent years, much of the focus in robust reinforcement learning (robust RL) research has been on handling uncertainties related to state observations. A substantial body of work has examined how adversarial perturbations or noise in state observations can be mitigated to improve the robustness of RL agents. For instance, Zhang et al. have extensively explored this area, including their work on adversarial perturbations (H. Zhang et al., 2020) and several related studies (He & Lv, 2023; Liu et al., 2022; H. Zhang et al., 2021; H. Zhang et al., 2019). These approaches typically aim to enhance agent performance by defending against observation disturbances using adversarial training techniques and stability-optimized architectures.

While observation robustness has garnered considerable attention, dynamics Robustness offer distinct advantages. In those scenarios, the agent faces uncertainty in the underlying transition dynamics of the environment, which can vary unpredictably. By considering worst-case scenarios in the transition probabilities, dynamics robust RL ensures that the agent learns strategies that perform well even under the most adverse conditions. This is particularly important for real-world applications where environments are dynamic and non-stationary, such as robotics and autonomous systems (Morimoto & Doya, 2005a; Nilim & El Ghaoui, 2005; Wiesemann et al., 2013a). Robustness in dynamics is especially critical in complex environments where state transitions are difficult to model or predict accurately (Blanchet et al., 2019; Iyengar, 2005). By leveraging robust optimization techniques, agents can maintain stability and performance even in the presence of significant model uncertainties.

Interestingly, some works have attempted to reframe dynamics uncertainties as observation uncertainties, simplifying the problem by assuming that the dynamics changes can be inferred from perturbations in the observations. (H. Zhang et al., 2019). These transformations provide a more tractable framework for handling uncertainty but may not fully capture the complexity of dynamic environment changes.

On the theoretical side, several papers have explored dynamics robust RL with a focus on mathematical guarantees. Researchers, e.g., Li et al. (Li & Lan, 2023) and Wang et al. (Y. Wang et al., 2024) have developed theoretical frameworks that provide guarantees for robust learning in environments with uncertain dynamics. These works primarily focus on developing robust policies under various model uncertainty frameworks and offer convergence guarantees. However, despite these theoretical advances, such approaches have limited applicability in large-scale environments, where the computational complexity and the amount of data required make it difficult to deploy these algorithms effectively in real-world scenarios.

A series of studies have utilized an adversarial player to simulate challenging conditions in dynamics by applying external forces or perturbations to the agent's dynamics. Pinto (Pinto et al., 2017) introduced an adversarial force to alter the agent's state transitions, aiming to test robustness under adversarial conditions. Similarly, Tessler (Tessler, Jinnai, et al., 2019) , Kumar (Kumar et al., 2020) and Pan (Pan et al., 2021) applied adversarial disturbances to affect action-level decisions. While these approaches demonstrate the potential of adversarial training, they are inherently limited by the adversarial player's bounded influence, often confined to small-scale perturbations or predefined parameters, which restricts the flexibility of dynamics changes.

In contrast, our approach, which directly generates adversarial transitions within a feasible uncertainty set, enables the agent to adapt to a wider spectrum of environmental variations. This flexibility surpasses the capabilities of adversarial player models by allowing unrestricted, yet controlled, alterations in the transition dynamics, making it more suitable for handling complex, real-world uncertainties.

1.3 Contributions

The contributions of this thesis are threefold:

1. **Solution to the Optimal Adversarial Transition**: We provide a formal proof demonstrating that identifying the optimal adversarial transition for each policy can be solved using regular MDP methods. This approach simplifies the complexity of the robust MDP framework by obtaining the optimal adversary through a conventional MDP solution.

2. **Novel Transition Gradient Approach**: To address the limitations of existing dual-based approaches in solving robust MDPs via stochastic approximation, we propose a novel method that parameterizes two networks to represent the policy and adversary. By introducing a new transition gradient mechanism, we effectively solve the inner problem, bypassing the challenges associated with traditional dual optimization techniques.

3. **Upper Bound on the Value Function Gap**: Given the non-convex, non-concave nature of the value function in robust MDPs, optimality of the policy cannot be guaranteed. However, we

establish an upper bound on the gap between the value function in the worst-case scenario and the interference-free case. Furthermore, we substantiate the theoretical results with extensive experiments, which validate the correctness of the theory and demonstrate the effectiveness and robustness of the feasible solutions obtained.

1.4 Thesis Structure

In Chapter 2, we establish the foundational background necessary for understanding the context of this work. Chapter 3 delves into our theoretical contributions, where we analyze the proposed framework and introduce a practical optimization objective. To address this objective, we develop an optimization method grounded in Transition Gradient techniques. Chapters 4 and 5 provide a comprehensive overview of our implementation and experimental details. Finally, in Chapter 6, we summarize the key findings and propose avenues for future research.

Chapter 2

Background

In this chapter I will provide a brief overview of the background of the research, including Markov Decision Processes (MDPs), Reinforcement Learning (RL), and different kinds of Uncertainty Sets.

2.1 Markov Decision Process

2.1.1 Basic Concepts

In this section, we discuss the underlying framework of Markov Decision Processes (MDPs), which provides the mathematical foundation for Reinforcement Learning.

In the discounted setting, we consider a tuple $(\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$, where $\mathcal{S}$ denotes a state space, $\mathcal{A}$ denotes an action space, $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ denotes a transition function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\rho_0 : \mathcal{S} \to [0, 1]$ denotes an initial distribution, and $\gamma \in (0, 1]$ is a discount factor.

At the beginning of the process, the agent's initial state s_0 is sampled from the initial state distribution ρ_0. At each time step t, the agent observes the current state s_t, and selects an action a_t based on its policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, which is a mapping from states to probability distributions on $\mathcal{A}$. The agent samples an action according to the probability distribution, i.e., $a_t \sim \pi(\cdot|s_t)$.

Once the agent takes the action a_t, the environment responds by transitioning the agent to a new state s_{t+1}, governed by the transition function p, i.e., $s_{t+1} \sim p(\cdot|s_t, a_t)$. After that, it receives a scalar reward r_{t+1} from the environment. This reward reflects the immediate benefit or cost of taking action a_t in state s_t.

To balance immediate and future rewards, the agent uses the discount factor γ, which ensures that rewards received earlier in time are more valuable than those in the distant future. The total discounted reward, also called the return, is calculated as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k})$$

where G_t denotes the return at time step t.

Given the start state s and policy π, we define the state value function as:

$$V_\pi(s) = \mathbb{E}_{\pi,p}\left[G_t | s_t = s\right]$$

Following the same logic, given the start state-action pair (s, a), we define the action value function Q_π to represent future rewards:

$$Q_\pi(s, a) = \mathbb{E}_{\pi,p}\left[G_t | s_t = s, a_t = a\right]$$

The state value function and action value function quantify the expected cumulative reward (return) an agent will receive. To optimize a policy, we define an optimization objective, the performance metric J_π, also known as the objective function. It quantifies how good a policy is in terms of the expected total return. We use the most common form of J_π, which is the expected return starting from the initial state distribution ρ_0:

$$J_\pi = \mathbb{E}_{s_0 \sim \rho_0}\left[V_\pi(s_0)\right]$$

2.1.2 Bellman Equation

To efficiently compute the value of a policy, we use the Bellman Equation, which provides a recursive decomposition of the value function. It expresses the value of a state in terms of the immediate reward plus the expected value of the next state. Given a policy π, the value of a state s can be expressed recursively through the Bellman Equation:

$$V_\pi(s) = \mathbb{E}_{\pi,p}\left[r(s, a) + \gamma V_\pi(s') | s\right]$$

and it can be expanded as:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a)\left[r(s, a) + \gamma V_\pi(s')\right]$$

Similarly, the action value function can also be written as:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi\left[r(s,a) + \gamma V_\pi(s')|s,a\right] \\ &= \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r(s,a) + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_\pi(s',a')\right] \end{aligned}$$

2.1.3 Bellman Operator

To better understand how value functions evolve and converge to optimal solutions, we introduce the Bellman Operator, which expresses the Bellman Equation in a more compact and formal way. The Bellman operator is central to the iterative procedures that compute value functions in both policy evaluation and optimal policy finding.

Bellman Operator for Policy Evaluation

The Bellman operator for policy evaluation is a mapping that transforms the current value function into a new value function based on the Bellman equation. Given a policy π, the Bellman operator $\mathcal{T}^\pi$ applied to a value function V is defined as:

$$\mathcal{T}^\pi V(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r(s,a) + \gamma V(s')\right]$$

This operator is central in policy evaluation, as it defines how the value function can be updated iteratively:

$$V_{k+1}(s) = \mathcal{T}^\pi V_k(s)$$

By repeatedly applying the Bellman operator, the value function converges to the true value of the policy π, denoted as V_π. The convergence is guaranteed because the Bellman operator for policy evaluation is a contraction mapping under certain conditions, ensuring that the sequence of value functions converges to the fixed point, V_π.

Bellman Optimal Operator

The Bellman optimal operator, denoted by $\mathcal{T}$, is used to compute the value function associated with the optimal policy. This operator is based on the Bellman optimality equation, which recursively defines the value of a state under the optimal policy. The Bellman optimal operator is defined as:

$$\mathcal{T}V(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r(s,a) + \gamma V(s')\right]$$

The operator updates the value function by choosing the action that maximizes the expected return at each state. Repeatedly applying this operator allows the value function to converge to the optimal value function V^*, which satisfies:

$$V^*(s) = \mathcal{T}V^*(s)$$

Thus, the Bellman optimal operator is essential for solving the **optimal control problem**, and its fixed point corresponds to the optimal value function.

2.2 Reinforcement Learning

In this section, we discuss the foundations of single-agent reinforcement learning (RL) and how the agent learns to interact with its environment to maximize its expected cumulative reward (Sutton & Barto, 2018).

2.2.1 Basic Framework

Reinforcement learning operates on the same underlying framework as Markov Decision Processes (MDPs), where an agent interacts with an environment (Bellman, 1957). The goal of the agent is to learn a policy π that maximizes the expected return G_t over time. The agent receives feedback from the environment in the form of rewards, and through trial and error, it improves its decision-making strategy (Kaelbling et al., 1996).

In single-agent RL, the agent acts alone in the environment, meaning that there is no external interference from other agents or adversaries. The agent is tasked with learning an optimal policy by balancing **exploration** (trying new actions to discover their effects) and **exploitation** (choosing actions that are known to yield high rewards) (Thrun, 1992).

2.2.2 Policy Optimization Objective

The objective in reinforcement learning is to find a policy π that maximizes the expected return. The performance of a policy is quantified by the objective function $J(\pi)$, which is defined as the expected return starting from the initial state distribution (Sutton & Barto, 2018):

$$J(\pi) = \mathbb{E}_{s_0 \sim \rho_0}\left[V_\pi(s_0)\right]$$

Policy gradient methods seek to maximize this objective by iteratively updating the policy parameters using gradient ascent. The gradient of the performance objective $J(\pi)$ with respect to the policy parameters θ can be expressed as (Sutton & Barto, 2018):

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q_\pi(s,a)\right]$$

This gradient expression forms the basis of policy gradient algorithms, which adjust the policy in the direction that maximizes the expected cumulative reward (Konda & Tsitsiklis, 2000).

2.2.3 Policy-based and Value-based Methods

Reinforcement learning algorithms can be broadly classified into two categories: **value-based methods** and **policy-based methods** (Sutton & Barto, 2018).

Value-based Methods

Value-based methods focus on learning a value function, which estimates the expected cumulative reward from any given state (or state-action pair). The agent uses this value function to select actions that lead to higher rewards. Common algorithms in this category include:

- **Q-learning** (Watkins & Dayan, 1992): A model-free RL algorithm that learns an action-value function $Q(s, a)$ and updates it iteratively using the Bellman equation. The optimal policy is derived by selecting actions that maximize $Q(s, a)$.

- **SARSA** (Rummery & Niranjan, 1994): An on-policy algorithm that updates the action-value function based on the state-action-reward-next state-next action tuple (s, a, r, s', a').

Policy-based Methods

Policy-based methods directly learn a policy π that maps states to actions without explicitly learning a value function. These methods can be beneficial in environments with large or continuous action spaces, where value-based methods might struggle. Common policy-based algorithms include:

- **REINFORCE** (Williams, 1992): A Monte Carlo-based algorithm that learns a stochastic policy by computing gradients of the expected return with respect to the policy parameters and updating the policy in the direction of higher returns.

- **Actor-Critic Methods** (Konda & Tsitsiklis, 2000): These methods combine policy-based and value-based approaches. The **actor** learns the policy, while the **critic** evaluates the policy by estimating a value function, providing feedback for the actor.

2.2.4 On-Policy and Off-Policy

In reinforcement learning, algorithms can be categorized as either **on-policy** or **off-policy** based on how they utilize experience for learning.

- **On-Policy Methods**: These methods learn the value of a policy while following that same policy. In other words, the agent learns by using the policy it is currently following to make decisions and update its estimates. An example of an on-policy algorithm is SARSA (Rummery & Niranjan, 1994), where the agent updates its action-value function based on the actions it actually takes under the current policy (Sutton & Barto, 2018). Common on-policy algorithms also include A2C, A3C (Mnih et al., 2016), PPO (Schulman et al., 2017), TRPO (Schulman, Levine, et al., 2015), GAE (Schulman, Moritz, et al., 2015), REINFORCE (Williams, 1992),

- **Off-Policy Methods**: Off-policy methods, on the other hand, learn the value of one policy while following a different policy. The agent can learn from data collected by a behavior policy that may differ from the target policy it is optimizing. Q-learning (Watkins & Dayan, 1992) is a typical off-policy method, where the agent updates its action-value function based on the maximum possible reward, regardless of the action taken by the behavior policy. Common off-policy algorithms also include DQN (Mnih et al., 2015), DDQN (Van Hasselt et al., 2016), Dueling DQN (Z. Wang et al., 2016), Prioritized Experience Replay (Schaul et al., 2015), SAC (Haarnoja et al., 2018), TD3 (Fujimoto et al., 2018)

2.2.5 Exploration-Exploitation Trade-off

In reinforcement learning, the agent must balance between **exploration** (choosing actions that may yield unknown rewards) and **exploitation** (choosing actions that are known to maximize rewards based on past experience). The trade-off between these two is a central challenge in RL (Thrun, 1992).

One common approach to balancing exploration and exploitation is the ϵ-greedy strategy, where the agent selects the action with the highest estimated value with probability $1 - \epsilon$ and explores a random action with probability ϵ (Sutton & Barto, 2018). The parameter ϵ controls the degree of exploration, and it often decays over time to allow more exploitation as the agent learns the environment.

Another method to encourage exploration is **entropy regularization**, which is commonly used in policy-based methods. By adding an entropy term to the optimization objective, the agent is encouraged to maintain a stochastic policy and explore more during training (Mnih et al., 2016).

2.2.6 Learning from Interaction with the Environment

In reinforcement learning, the agent continuously interacts with the environment to improve its policy. The process of learning involves collecting experience in the form of transitions $(s_t, a_t, r_{t+1}, s_{t+1})$, and using this data to update the policy or value functions (Sutton & Barto, 2018). There are two main approaches for this:

Model-Free Methods

Model-free methods do not assume any knowledge of the environment's dynamics (transition probabilities and reward function). Instead, they rely on direct interaction with the environment to learn the optimal policy. Common model-free algorithms include Q-learning, SARSA, and REINFORCE (Sutton & Barto, 2018).

Model-Based Methods

In contrast, model-based methods aim to learn a model of the environment's dynamics (i.e., the transition function and reward function) and use this model to plan future actions. These methods can be more sample-efficient than model-free methods, as they can simulate future states based on the learned model and optimize the policy accordingly (Kaelbling et al., 1996).

2.3 Uncertainty Sets in Reinforcement Learning

In robust reinforcement learning, uncertainty sets play a crucial role in defining the space of perturbations or deviations that the agent must consider when learning policies (Kothari et al., 2020; Nilim & El Ghaoui, 2005). Different types of uncertainty sets reflect different assumptions about the nature and structure of these uncertainties. In this section, we explore various kinds of uncertainty sets used in reinforcement learning, focusing on their construction and theoretical implications.

2.3.1 Rectangular Uncertainty Sets

One of the most common uncertainty structures in reinforcement learning is the **rectangular uncertainty set**, often applied to both state and action pairs. I list a few kinds of commonly used uncertainty sets below.

(s, a)-Rectangular Uncertainty

In the (s, a)-rectangular uncertainty model, the uncertainty in the transition probabilities for each state-action pair (s, a) is independent of other state-action pairs. This form of uncertainty is particularly useful for decomposing the problem, as it allows for independent optimization over each (s, a) pair (Iyengar, 2005; Wiesemann et al., 2013a).

Mathematically, the uncertainty set for a transition kernel under (s, a)-rectangular uncertainty can be expressed as:

$$\mathcal{U}_{\text{rect}} = \{p(s'|s,a) : \|p(s'|s,a) - \hat{p}(s'|s,a)\| \leq \delta(s,a) \ \forall (s,a) \in \mathcal{S} \times \mathcal{A}\}$$

where $\delta(s, a)$ defines the uncertainty bound for each state-action pair (s, a), and $\hat{p}(s'|s, a)$ is the nominal transition model.

Uniqueness: The (s, a)-rectangular uncertainty set is particularly useful in settings where the environment's transition dynamics are assumed to vary independently across state-action pairs. It simplifies the optimization problem by allowing for local uncertainty handling, leading to tractable solutions in large-scale reinforcement learning problems (Nilim & El Ghaoui, 2005).

2.3.2 Wasserstein Uncertainty

The **Wasserstein uncertainty set** is based on the Wasserstein distance, which measures the distance between two probability distributions (Esfahani & Kuhn, 2018). This type of uncertainty set is useful when the agent's uncertainty over transitions or outcomes can be described by a shift in probability mass across states, rather than simple independent perturbations.

For a given nominal transition kernel $\hat{p}(s'|s, a)$, the Wasserstein uncertainty set is defined as:

$$\mathcal{U}_{\text{wass}} = \{p(s'|s,a) : \mathcal{W}(p(s'|s,a), \hat{p}(s'|s,a)) \leq \delta_{\text{wass}}\}$$

where $\mathcal{W}$ denotes the Wasserstein distance, and δ_{wass} is the allowed threshold for deviations from the nominal distribution.

Uniqueness: The Wasserstein uncertainty set is particularly well-suited for situations where the uncertainty arises from shifts in probability mass, such as in transportation problems or in scenarios where the agent must account for correlated changes across states (Blanchet et al., 2019). Unlike rectangular uncertainty, which assumes independence, Wasserstein uncertainty captures global distributional shifts.

2.3.3 n-Contamination Uncertainty

***n*-contamination uncertainty** is another type of uncertainty that assumes the nominal model is "contaminated" by a certain proportion of adversarial transitions. In this model, the true transition kernel is a mixture of the nominal model and an adversarial component (Jansen, 2003).

The n-contamination uncertainty set can be written as:

$$\mathcal{U}_{n-\text{cont}} = \{p(s'|s,a) : p(s'|s,a) = (1-\alpha)\hat{p}(s'|s,a) + \alpha q(s'|s,a),\ \alpha \leq \delta_{\text{cont}}\}$$

where α represents the contamination factor, $\hat{p}(s'|s,a)$ is the nominal transition, and $q(s'|s,a)$ is an adversarial distribution. The contamination level δ_{cont} controls the extent to which the adversarial distribution can affect the true transition model.

Uniqueness: This model is unique in that it assumes the environment contains both a nominal and an adversarial component, which is a common assumption in settings where agents must operate under potential malicious attacks or extreme disturbances (Jansen, 2003).

2.3.4 Ellipsoidal Uncertainty

Ellipsoidal uncertainty sets model uncertainty as a region that forms an ellipsoid around the nominal model. These sets are commonly used in robust control and optimization, where uncertainty is constrained to lie within a certain ellipsoidal region (Ben-Tal et al., 2009).

An ellipsoidal uncertainty set is defined as:

$$\mathcal{U}_{\text{ellip}} = \left\{p(s'|s,a) : (p(s'|s,a) - \hat{p}(s'|s,a))^\top Q(p(s'|s,a) - \hat{p}(s'|s,a)) \leq \delta_{\text{ellip}}\right\}$$

where Q is a positive definite matrix that defines the shape of the ellipsoid, and δ_{ellip} is the radius that bounds the uncertainty.

Uniqueness: Ellipsoidal uncertainty sets are well-suited for scenarios where uncertainties are not independent but are instead correlated in a structured manner. The shape of the ellipsoid, governed by the matrix Q, allows for directional sensitivity to uncertainties, which can be advantageous in systems where certain dimensions of uncertainty are more critical than others (Ben-Tal et al., 2009).

2.3.5 Polyhedral Uncertainty

Polyhedral uncertainty assumes that the uncertainty lies within a polyhedron, which can be represented as the intersection of multiple linear inequalities (Bertsimas et al., 2011). This type of uncertainty set is particularly useful in linear programming and robust optimization.

A polyhedral uncertainty set is expressed as:

$$\mathcal{U}_{\text{poly}} = \{p(s'|s,a) : Ap(s'|s,a) \leq b\}$$

where A and b are matrices that define the linear inequalities bounding the uncertainty.

Uniqueness: Polyhedral uncertainty sets are flexible and allow for a broad range of shapes, making them suitable for applications in which the uncertainty has a complex structure (Bertsimas et al., 2011). These sets are computationally efficient to handle, especially in optimization problems, as they lead to tractable linear programming formulations.

2.3.6 Summary

Uncertainty sets provide a framework for modeling deviations from the nominal model in reinforcement learning. Different uncertainty sets capture different types of perturbations and offer unique advantages depending on the structure of the environment and the kind of uncertainty being modeled (Iyengar, 2005; Nilim & El Ghaoui, 2005; Wiesemann et al., 2013a). From (s,a)-rectangular uncertainty that assumes independence across state-action pairs, to Wasserstein, contamination, ellipsoidal, polyhedral, and Hellinger uncertainty sets, each approach brings its own set of assumptions and computational benefits. The choice of uncertainty set is crucial for designing robust policies that can handle real-world variability and adversarial perturbations.

2.4 State Perturbation

Building on the concept of uncertainty sets discussed in the previous section, state perturbation focuses on the impact of uncertainties in state transitions or observations on policy robustness and task performance. This section reviews the existing work on state perturbation within the robust Markov Decision Process (MDP) framework, emphasizing theoretical formulations, adversarial training, hierarchical methods, and representation learning approaches.

2.4.1 Theoretical Foundations of Robust MDPs with State Perturbation

The foundational work by (Nilim & El Ghaoui, 2005) and (Iyengar, 2005) introduced robust MDPs with state perturbations. They formulated the problem as a bi-level optimization, where the outer level optimizes the policy, and the inner level minimizes the worst-case performance under bounded uncertainties in state transitions. This is formally expressed as:

$$\max_{\pi} \min_{\tilde{p}(s,a)\in\mathcal{B}(s,a)} \mathbb{E}_{s_0,a_0}[Q_{\pi,\tilde{p}}(s_0,a_0)], \tag{2.1}$$

where π represents the policy, $\tilde{p}(s,a)$ is the transition model, and $\mathcal{B}(s,a)$ denotes the uncertainty set. This framework provides theoretical performance guarantees under bounded state perturbations.

Building on this, (Xu & Mannor, 2010) extended the robust MDP framework to incorporate distributional robustness using Wasserstein distances to address state distribution shifts, offering a more flexible treatment of state uncertainties in stochastic settings. Similarly, (Wiesemann et al., 2013b) generalized robust MDPs to Markov games, providing insights into multi-agent systems under state uncertainties, and (Banerjee & Ghosh, 2021) derived robust Bellman operators to tackle adversarial perturbations in theoretical reinforcement learning frameworks.

2.4.2 Adversarial Training for State Perturbation

Adversarial approaches have become a dominant method for addressing state perturbation in robust reinforcement learning. (Pinto et al., 2017) introduced a robust adversarial reinforcement learning framework where perturbations are injected during training to simulate worst-case scenarios, forcing the policy to adapt to adversarial state dynamics. Extending this concept, (J. Zhang & Xu, 2021) employed adversarial training in deep RL, demonstrating its effectiveness in countering state noise and achieving robustness in high-dimensional environments. These approaches highlight the potential of adversarially robust training to handle extreme uncertainties.

2.4.3 Hierarchical and Representation Learning Approaches

Robustness in hierarchical and representation learning methods has also been studied extensively. (Mankowitz et al., 2018) explored robust options in hierarchical reinforcement learning, showing their effectiveness in mitigating state perturbations by incorporating robust sub-policies. In continuous control tasks, (Heess et al., 2015) proposed stability-driven learning, addressing the challenges posed by state perturbations by embedding stability criteria into policy optimization.

Robust representation learning methods have been proposed to address state perturbations by focusing on invariant features. (Tessler, Efroni, et al., 2019) presented adaptive representation learning to improve policy generalization under unseen perturbations, while (Lee et al., 2020) introduced sparse perturbation models to analyze policy degradation under targeted state changes.

2.4.4 Robust Reinforcement Learning with Neural Networks

Combining robust MDPs with neural networks has led to significant advancements in addressing state perturbations in large-scale environments. (Morimoto & Doya, 2005b) demonstrated the integration of neural network-based policies with robust control principles to mitigate state noise. (Vinitsky et al., 2020) further proposed iterative methods to fine-tune policies for dynamic robustness, providing a practical approach to adapt to evolving state perturbations.

Recent studies have also focused on first-order optimization techniques for robust reinforcement learning. (Li & Lan, 2023) developed a convex uncertainty framework for addressing dynamics perturbations, offering convergence guarantees and applicability to complex control tasks. Similarly, (Y. Wang et al., 2024) provided theoretical bounds for robust policies under non-stationary dynamics, emphasizing their applicability in adversarial environments, and extended previous work to average reward settings.

These studies collectively provide a comprehensive view of addressing state perturbations in robust reinforcement learning. While significant progress has been made in theoretical foundations, adversarial training, and practical applications, challenges remain in scaling these approaches to highly dynamic and stochastic environments. Future work could further explore hybrid methods that integrate adversarial robustness, hierarchical learning, and neural network architectures to achieve greater scalability and reliability under state uncertainties.

Chapter 3

Transition Gradient

In this chapter, we present a robust Markov Decision Process (MDP) framework to address uncertainties in transition dynamics.

The core idea of our approach is to model the problem as a two-player game, where the first player represents the policy being optimized, and the second player is an adversary that generates new transition probabilities to challenge the policy's performance. This adversarial setting allows us to derive a robust policy capable of withstanding worst-case transition dynamics. We solve the inner problem of robust MDP by proposing a novel transition gradient method and establishing theoretical guarantees for policy performance under the strongest adversarial perturbations. By combining traditional policy optimization techniques with an adversarially-driven transition model, we ensure that the resulting policies remain robust across a range of perturbations, leading to significantly improved performance in environments with inherent uncertainties.

The remainder of this chapter details the mathematical formulations, the novel transition gradient theorem, and the practical implementation of our robust policy training approach.

3.1 Definitions

In addition to the notations introduced in Chapter 2, we define $d_{\pi,p} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ as the stationary distribution of states induced by a policy π and transition probability p.

Formally, it is given by:

$$d_{\pi,p}(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s|\pi, p)$$

We define the transition probability after perturbation as $\tilde{p}$, represented as $\tilde{p}(s'|s,a) = \tilde{\Pr}(s_{t+1} = s'|s_t = s, a_t = a)$. Formally, the perturbation function is denoted by $\tilde{p} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$.

Definition 1 *For each (s,a), we define a perturbation set $\mathcal{B}(s,a)$, which contains all possible transition probabilities after perturbations. Formally, $\tilde{p}(s,a) \in \mathcal{B}(s,a)$, where $\mathcal{B}(s,a)$ is a set of probability distributions, $s \in \mathcal{S}$ and $a \in \mathcal{A}$. This ensures that the perturbation does not become too drastic:*

$$\mathcal{B}(s,a) = \{p'(s,a) \mid D_{KL}(p'(s,a) \| p(s,a)) < \epsilon\}$$

Here, $p(s,a)$ denotes the true transition probability in the environment before the perturbation, and ϵ is a constant. The constant ϵ provides the supremum of the KL divergence between the transition probabilities before and after perturbation.

The perturbed value and action-value functions under $\tilde{p}$ are analogous to those in a regular MDP:

$$\tilde{V}_{\tilde{p},\pi}(s) = \mathbb{E}_{\tilde{p},\pi}[\sum_{k=0}^{\infty} \gamma^k r_{r+k+1} | s_t = s], \quad \tilde{Q}_{\tilde{p},\pi}(s,a) = \mathbb{E}_{\tilde{p},\pi}[\sum_{k=0}^{\infty} \gamma^k r_{r+k+1} | s_t = s, a_t = a]$$

Objective 1 *Our objective is to first obtain an adversarial transition that deviates insignificantly from the true transition, minimizing the cumulative rewards. We then train the policy under this adversarial setting to achieve a more robust policy.*

Formally, this objective is defined as:

$$\max_{\pi} \min_{\tilde{p}(s,a) \in \mathcal{B}(s,a)} \mathbb{E}_{s_0,a_0}[Q_{\pi,\tilde{p}}(s_0,a_0)] \tag{3.1}$$

This formulation not only seeks to maximize the reward but also aims to achieve the highest reward under the worst-case scenario. We now begin with the fundamental case: the Bellman equation for a fixed π and $\tilde{p}$.

3.2 Uncertain Transition and Bellman Equations

The definition of the new MDP is provided below, introducing dynamics uncertainty on top of the standard MDP framework. This extension incorporates uncertain transitions, paving the way for further analysis.

Theorem 1 (Bellman equation for a fixed policy π and $\tilde{p}(s,a)$) *The state-value function and action-value function are represented as follows:*

$$\tilde{V}_{\tilde{p},\pi}(s) = \sum_{a\in A} \pi(a|s) \sum_{s'\in S} \tilde{p}(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}_{\tilde{p},\pi}(s')\right] \tag{3.2}$$

$$\tilde{Q}_{\tilde{p},\pi}(s,a) = \sum_{s'\in S} \tilde{p}(s'|s,a) \left[R(s,a,s') + \gamma \sum_{a'\in A} \pi(a'|s') \tilde{Q}_{\tilde{p},\pi}(s',a')\right] \tag{3.3}$$

3.3 Optimal Adversarial Policy

Natually, the first step is to determine the "optimal" adversarial transition policy $\tilde{p}^*$ under a fixed policy π, which minimizes the cumulative reward for this π. The value and action-value functions under the optimal adversarial policy $\tilde{p}^*$ for a fixed policy π are defined as:

$$\tilde{p}^*(\pi) = \arg\min_{\tilde{p}} \tilde{Q}_{\pi,\tilde{p}}(s,a)$$

Note: The adversarial transition $\tilde{p}$ does not vary with the policy π, whereas the optimal transition $\tilde{p}^*$ is specific to a given policy π and evolves as the policy changes. This dependency is denoted by $\tilde{p}^*(\pi)$.

Then we proceed to prove that this "optimal" adversarial transition is achievable. We first derive a lemma that demonstrates the equivalence between finding the optimal adversary in a TA-MDP and finding the optimal policy in a regular MDP:

Lemma 1 (Equivalence of finding the optimal adversary in a TA-MDP and finding the optimal policy in an MDP) *Given a TA-MDP $M = (S, A, B, R, p, \gamma)$ and a fixed policy π, there exists a regular MDP $\hat{M} = (\hat{S}, \hat{A}, \hat{R}, \hat{p}, \gamma)$ such that the optimal policy $\hat{\pi}^*$ of $\hat{M}$ is the optimal adversary for the TA-MDP.*

This lemma also provides a useful property, namely that $-\hat{V}_{\hat{\pi}}(\hat{s}) = \tilde{Q}_{\tilde{p}}(s,a)$. This implies that by finding $\hat{\pi}^*$, which maximizes $\hat{V}_{\hat{\pi}}(\hat{s})$, we simultaneously solve the regular MDP and determine the optimal function $\tilde{p}^*$ that minimizes $\tilde{Q}_{\tilde{p}}(s,a)$, and vice versa.

Next, in Theorem 2, we will continue to prove that this optimal adversarial transition $\tilde{p}^*$ can be found.

Theorem 2 (Bellman contraction for optimal adversary) *Define the robust Bellman operator $\mathcal{T} : \mathbb{R}^{|S||A|} \to \mathbb{R}^{|S||A|}$ as:*

$$(\mathcal{T}\tilde{Q})(s,a) = \min_{\tilde{p}(s,a)\in\mathcal{B}(s,a)} \sum_{s'\in S} \tilde{p}(s'|s,a) \cdot \left[R(s,a,s') + \gamma \sum_{a'\in A} \pi(a'|s')\tilde{Q}(s',a') \right] \tag{3.4}$$

The robust Bellman equation for the optimal adversary $\tilde{p}^*$ can be written as $\tilde{Q}_{\tilde{p}^*(\pi)} = \mathcal{T}\tilde{Q}_{\tilde{p}^*(\pi)}$. Additionally, $\mathcal{T}$ is a contraction and converges to $\tilde{Q}_{\tilde{p}^*(\pi)}$.

Theorem 2 demonstrates that, given a fixed policy π, we can evaluate its performance under the optimal (strongest) adversary through a Bellman contraction. This is functionally analogous to the "policy evaluation" procedure in regular MDPs.

Finally, we address the ultimate objective of finding an optimal policy π^* under the strongest adversary $\tilde{p}^*(\pi^*)$. For a fixed perturbed transition, the process of obtaining the optimal policy follows the same steps as in regular MDPs. By Bellman's Optimality Principle, we know that an optimal policy always exists.

However, in our case, the optimal adversarial policies vary depending on the current behavior policy (see note). Taking this into account, in Proposition 1, it has been proven that an overall optimal policy can be found iteratively.

Proposition 1 (Markov Optimality) *There exists a deterministic optimal policy $\pi^* \in \Pi_{MD}$ such that $\tilde{V}_{\pi^*,\tilde{p}^*(\pi^*)}(s) \geq \tilde{V}_{\pi,\tilde{p}^*(\pi)}(s)$ for all s, where $\tilde{p}^*(\pi)$ denotes the optimal adversary for a specific policy π.*

With this guarantee, we can iteratively solve for both the optimal policy and adversarial policy until they converge to a fixed point. In Iyengar (Iyengar, 2005), several dual problems are proposed, along with guarantees on computational complexity.

3.4 Transition Gradient

Unfortunately, we cannot directly apply stochastic gradient descent to solve these problems due to the inability to express the gradient computation in the form of expectations. Given the difficulty in optimizing Objective 1 directly, we propose an alternative objective. Specifically, we can employ the policy gradient approach:

Objective 2 *We parameterize the policy π by ω and the adversarial transition $\tilde{p}$ by θ:*

$$\max_{\omega} \min_{\theta} \mathbb{E}_{s_0,a_0} \left[Q_{\omega,\theta}(s_0, a_0) + D_{KL} \left(\tilde{p}_\theta(s_0, a_0) \| p(s_0, a_0) \right) \right] \tag{3.5}$$

Specifically, we first fix the policy π_ω and perform gradient descent over $\tilde{p}_\theta$ on the objective function $Q_{\omega,\theta}$, followed by gradient ascent on π_ω with $\tilde{p}_\theta$ fixed. During the gradient descent phase, we incorporate a KL-divergence regularization term to ensure that the perturbed transition probabilities do not deviate significantly from the true transition probabilities. This iterative process refines the policy and enhances its robustness.

We now delve into the implementation details.

Assumption 1 (Availability of true transition) *Since we operate in a simulated environment, we assume that the true transition probabilities are fully accessible.*

Note: We use the true transition probabilities for certain computations but prevent the model from directly accessing them as part of the states. This forces the model to generalize from its own experiences.

This assumption provides significant convenience, allowing us to bypass the challenges of estimating the true transition probabilities. It prevents adversarial transitions from deviating excessively from the original transitions, which could otherwise lead to training instability and compromise the reliability of the learned policy in the simulated environment.

Theorem 3 (Transition Gradient) *Given a fixed π, and with the perturbed transition function parameterized by θ, we have:*

$$\nabla_\theta \mathbb{E}_{s_0,a_0}[Q_{\pi,\theta}(s_0,a_0)] \propto \mathbb{E}_{\substack{s\sim d^{\theta,\pi}(s)\\ a\sim\pi(\cdot|s)\\ s'\sim\tilde{p}_\theta(\cdot|s,a)}} \left[\left(R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s')Q_{\pi,\theta}(s',a')\right)\nabla_\theta \log \tilde{p}_\theta(s'|s,a)\right]$$

$$\begin{aligned}&\nabla_\theta \mathbb{E}_{s\sim d^{\pi,\theta}(s),a\sim\pi(\cdot|s)}[D_{KL}(p(\cdot|s,a)\|\tilde{p}_\theta(\cdot|s,a))]\\ &= \mathbb{E}_{\substack{s\sim d^{\pi,\theta}(s),a\sim\pi(\cdot|s)\\ s'\sim\tilde{p}_\theta(\cdot|s,a)}}\left[\left(\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}+1\right)\nabla_\theta \log \tilde{p}_\theta(s'|s,a)\right]\end{aligned} \tag{3.6}$$

Proposition 2 (Policy Gradient with Transition Gradient) *Given a fixed $\tilde{p}$, and with the policy parameterized by ω, we have:*

$$\nabla_\omega \mathbb{E}_{s_0,a_0}[Q_{\omega,\tilde{p}}(s_0,a_0)] \propto \mathbb{E}_{\substack{s\sim d^{\omega,\tilde{p}}(s),a\sim\pi_\omega(\cdot|s)\\ s'\sim\tilde{p}(\cdot|s,a),a'\sim\pi_\omega(\cdot|s')}}[Q_{\omega,\tilde{p}}(s',a')\nabla_\omega \log\pi_\omega(a'|s')] \tag{3.7}$$

Theorems 3 and Proposition 2 show how to compute the gradients. Since both the true probability p and the perturbed transition $\tilde{p}$ are known, it is possible to use importance sampling. This allows us to sample from the true environment and estimate the value function in the adversarial environment. This is particularly important in practical applications, as we generally do not want to manipulate the environment excessively.

Theorem 4 (Transition Adjustment Sampling) *Inspired by the regular importance sampling, we propose the transition adjustment sampling to adjust the gradient. The gradient of the action value function could be obtained by:*

$$\mathbb{E}_{\substack{s\sim d^{\pi}(s)\\ a\sim\pi(\cdot|s)\\ s'\sim p(\cdot|s,a)}}\left[\frac{d^{\pi,\theta}(s)}{d^{\pi}(s)}\left[R(s,a,s')+\gamma\sum_{a'\in\mathcal{A}}\pi(a'|s')Q_{\pi,\theta}(s',a')\right]\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}\nabla_\theta\log\tilde{p}_\theta(s'|s,a)\right] \tag{3.8}$$

and the gradient of the KL-divergence term could be obtained by:

$$\mathbb{E}_{\substack{s\sim d^{\pi}(s),a\sim\pi(\cdot|s)\\ s'\sim p(\cdot|s,a)}}\left[\frac{d^{\pi,\theta}(s)}{d^{\pi}(s)}\left\{\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}+1\right\}\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}\nabla_\theta\log\tilde{p}_\theta(s'|s,a)\right] \tag{3.9}$$

when we update the policy, we can use the following gradient:

$$\mathbb{E}_{\substack{s\sim d^{\omega,p}(s),a\sim\pi_\omega(\cdot|s)\\ s'\sim p(\cdot|s,a),a'\sim\pi_\omega(\cdot|s')}}\left[\frac{d^{\omega,\tilde{p}}(s)}{d^{\omega,p}(s)}\cdot\frac{\tilde{p}(s'|s,a)}{p(s'|s,a)}Q_{\omega,\tilde{p}}(s',a')\nabla_\theta\log\pi_\omega(a'|s')\right] \tag{3.10}$$

However, we cannot directly get the action value function $Q_{\pi,\tilde{p}_\theta}$, which associated with the policy π and the adversarial transition $\tilde{p}_\theta$, with samples we obtained in the original environment. To address this issue, we can use weighted return estimation to estimate the action value function.

$$Q_{\pi,\tilde{p}_\theta}(s_t, a_t) \approx \sum_{t'=t}^{T} W_{t'} \gamma^{t'-t} r_{t'}, \tag{3.11}$$

where $W_{t'} = \prod_{i=t}^{t'} \frac{\tilde{p}_\theta(s'_i|s_i,a_i)}{p(s'_i|s_i,a_i)}$ is the cumulative transition adjustment ratio. Here, $\tilde{p}_\theta(s'_i|s_i, a_i)$ is the generated transition and $p(s'_i|s_i, a_i)$ is the original transition.

Similarly, for those algorithms which utilize funtion approximator to estimate the action value function, we can use offline function approximation:

$$Q_{\pi,\tilde{p}_\theta}(s_t, a_t) \approx Q_{\pi,\tilde{p}_\theta}(s_t, a_t) + \alpha \left[r_t + \gamma Q_{\pi,\tilde{p}_\theta}(s', \pi(s')) - Q_{\pi,\tilde{p}_\theta}(s_t, a_t)\right] \tag{3.12}$$

where $s' = \tilde{p}_\theta(s_t, a_t)$

However, in practical implementation, ensuring the simultaneous convergence of the policy and adversarial transition to their optima is challenging. Nevertheless, we demonstrate that under certain assumptions, the performance loss due to an optimal adversary can be bounded.

Theorem 5 (Performance Gap) *Given a policy π, let the state value function in a non-adversarial MDP be $V_\pi(s)$, and let the state value function under the optimal adversary $\tilde{p}^*$ and the same policy π in a TA-MDP be $\tilde{V}_{\pi,\tilde{p}^*(\pi)}(s)$. For all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, the following inequality holds:*

$$\max_{s,a} |V_\pi(s) - \tilde{V}_{\pi,\tilde{p}^*(\pi)}(s)| \leq \frac{2\max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{a\in\mathcal{A}} \pi(a|s) \max_{\tilde{p}(s,a)\in\mathcal{B}(s,a)} D_{TV}(p(\cdot|s,a), \tilde{p}(\cdot|s,a)) \tag{3.13}$$

3.5 Algorithms

Based on the theoretical foundation discussed above, we propose updated versions of the Policy Gradient (Sutton et al., 1999), Actor-Critic (Konda & Tsitsiklis, 2000), and Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithms. The changes we made to the original algorithms are highlighted in blue.

Algorithm 1 Policy Gradient with Transition Gradient and Transition Adjustment Sampling

```
1:  Initialize policy parameters $\omega$, and transition parameters $\theta$
2:  for each episode $= 1, 2, \dots, E$ do
3:      Generate a trajectory $s_1, a_1, r_2, \dots, s_T, a_T, r_T$, following $\pi_\omega$ and transition $p$
4:      for each step of the episode $t = 1, 2, \dots, T$ do
5:          $W_{t'} \leftarrow \prod_{i=t}^{t'} \frac{\tilde{p}_\theta(s'_i|s_i,a_i)}{p(s'_i|s_i,a_i)}$
6:          $G_t \leftarrow W_{t'} \sum_{t'=t}^{T} \gamma^{t'-t} r'_t$
7:          $\rho_t \leftarrow \frac{\tilde{p}_\theta(s_{t'+1}|s_{t'},a_{t'})}{p(s_{t'+1}|s_{t'},a_{t'})}$
8:          $\theta \leftarrow \theta - \alpha \sum_t^T \rho_t G_t \nabla_\theta \log \tilde{p}_\theta(s_{t+1}|a_t, s_t)$
9:                  $-\beta \rho_t \left(\log \frac{\tilde{p}_\theta(s_{t+1}|s_t,a_t)}{p(s_{t+1}|s_t,a_t)} + 1\right) \nabla_\theta \log \tilde{p}_\theta(s_{t+1}|a_t, s_t)$
10:         $\omega \leftarrow \omega + \alpha \sum_t^T \rho_t G_t \nabla_\omega \log \pi_\omega(a_t|s_t)$
11:     end for
12: end for
13: Return $\omega$ and $\theta$
```

Algorithm 2 Actor-Critic with Transition Gradient and Transition Adjustment Sampling

```
1:  Initialize actor parameters $\omega_a$, critic parameters $\omega_c$ and transition parameters $\theta$
2:  for each episode $= 1, 2, \dots, E$ do
3:      Generate a trajectory $s_1, a_1, r_2, \dots, s_T, a_T, r_T$, following $\pi_{\omega_a}$ and transition $p$
4:      for each step $t$ of the episode, $t = 1, 2, \dots, T$ do
5:          $s' \leftarrow \tilde{p}_\theta(s_t, a_t)$
6:          $\delta_t \leftarrow r_t + \gamma V_{\omega_c}(s') - V_{\omega_c}(s_t)$
7:          $\rho_t \leftarrow \frac{\tilde{p}_\theta(s_{t'+1}|s_{t'},a_{t'})}{p(s_{t'+1}|s_{t'},a_{t'})}$
8:          $\theta \leftarrow \theta - \alpha_t \rho_t \left(\log \frac{\tilde{p}_\theta(s_{t+1}|s_t,a_t)}{p(s_{t+1}|s_t,a_t)} + 1\right) \nabla_\theta \log \tilde{p}_\theta(s_{t+1}|a_t, s_t)$
9:                  $-\beta \rho_t \delta_t \nabla_\theta \log \tilde{p}_\theta(s_{t+1}|a_t, s_t)$
10:         $\omega_c \leftarrow \omega_c - (-\alpha_c \delta_t \nabla_{\omega_c} V_{\omega_c}(s_t))$
11:         $\omega_a \leftarrow \omega_a + \alpha_a \rho_t \delta_t \nabla_{\omega_a} \log \pi_{\omega_a}(a_t|s_t)$
12:     end for
13: end for
14: Return $\omega_a$, $\omega_c$ and $\theta$
```

Algorithm 3 PPO with Transition Gradient and Transition Adjustment Sampling

```
1: Initialize policy parameters $\omega$, value function parameters $\omega_v$, and transition parameters $\theta$
2: for each iteration $= 1, 2, \ldots, E$ do
3:     Collect trajectories using $\pi_\omega$ and transition $p$
4:     for each step $t = 1, 2, \ldots, T$ do
5:         $s' \leftarrow \tilde{p}_\theta(s_t, a_t)$
6:         $\delta_t \leftarrow r_t + \gamma V_{\omega_c}(s') - V_{\omega_c}(s_t)$
7:         Compute advantage estimate $\hat{A}_t$ using $\delta_t$
8:         $\rho_t \leftarrow \frac{\tilde{p}_\theta(s_{t+1}|s_t,a_t)}{p(s_{t+1}|s_t,a_t)}$
9:         Compute the ratio $r_t(\omega) = \frac{\pi_\omega(a_t|s_t)}{\pi_{\omega_{old}}(a_t|s_t)}$
10:        Update $\omega$ using the PPO clipped objective:
               $$\omega \leftarrow \omega + \alpha \rho_t \hat{A}_t \nabla_\omega \min\left(r_t(\omega)\hat{A}_t, \text{clip}(r_t(\omega), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$$
11:        Update $\theta$ using importance-weighted transition update
12:        Update value function parameters $\omega_v$
13:    end for
14: end for
15: Return $\omega$, $\omega_v$, and $\theta$
```

Chapter 4

Implementation

The device and code base used in this thesis are provided in this chapter.

4.1 Computing Environment

The computing environment used in this thesis is a Linux server with:

- 3x AMD EPYC 7742 64-Core Processor
- 64GB RAM
- 500GB File System
- 1 NVIDIA 4080 GPU

4.2 Stochastic Transition

Many existing reinforcement learning (RL) environments are deterministic, such as the CartPole (Barto et al., 1983), MountainCar (Moore, 1990), Acrobot (Sutton, 1996), and several Atari games (e.g., Pong, Breakout, and Space Invaders) (Bellemare et al., 2013). MuJoCo environments like HalfCheetah and Hopper (Todorov et al., 2012) are also typically deterministic. However, to properly validate our theoretical contributions, it is essential to modify these environments to incorporate stochastic transitions.

Introducing noise in the spatial domain is often risky, as it can lead to a range of non-smooth behaviors, such as teleportation (Haarnoja et al., 2018; Plappert et al., 2018). In our experiments, we empirically adjusted which states to perturb, focusing primarily on perturbations to velocity rather than position. This approach is safer and results in more natural transitions (Heess et al., 2017; Schulman, Levine, et al., 2015).

Overall, to introduce stochasticity in the environment, we added Gaussian noise with a relatively small standard deviation to specific states. This allows us to simulate stochastic transitions while maintaining system stability, ensuring the behavior remains close to real-world conditions (Tang et al., 2018; Zhu et al., 2020).

Since neural networks have difficulty directly generating Gaussian distributions, unless the mean and standard deviation are explicitly modeled, this would result in overly constrained perturbations (e.g., visually appealing but rigid Gaussian distributions).

To introduce more flexibility in the perturbations, we opted to use a neural network to generate nine elements to simulate a discrete probability distribution. This approach allows for greater adaptability in the perturbations (Goodfellow et al., 2016), enabling us to avoid the rigid limitations of traditional Gaussian noise. In practice, a bias factor will be sampled from the generated distribution (See Figure 4.1), then be multiplied by a scaler.

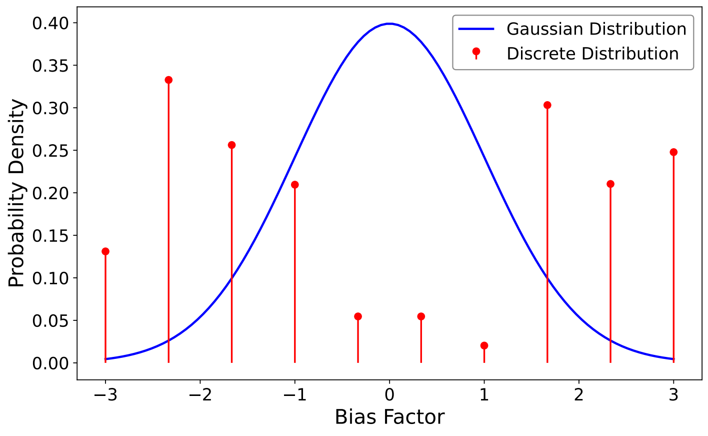


Figure 4.1: Distribution

4.3 Other details

One of the core idea in our algorithms is to calculate the ratio between the actual transition probability and the perturbed transition probability:

$$\frac{\tilde{p}(s'|s,a)}{p(s'|s,a)} \tag{4.1}$$

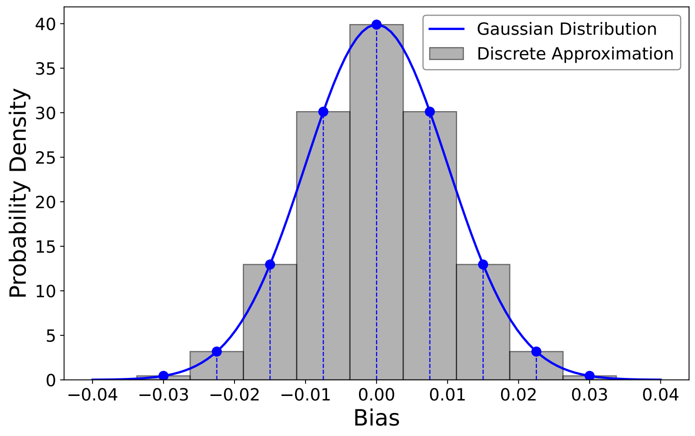


Figure 4.2: Generated Transition Probability

Although there is a KL divergence restriction between the generated transition and the actual transition, the new transition does not necessarily follow a Gaussian distribution. If we apply the transition adjustment sampling, the actual bias is sampled according to a continuous Gaussian distribution, and it will not necessarily be the same as one of the discrete generated bias. The solution are discuess below.

Generated Transition Probability I generated a probability distribution consisting of nine discrete probabilities for each dimension. The differences between them and zero are based on the mean and standard deviation of Gaussian noises. According to the Gaussian distribution formula, the differences between them and zero, represented by these nine probabilities, are:

$$n \cdot \sigma \cdot \sqrt{2 \log 2}, \quad n = -4, -3, \ldots, 3, 4 \tag{4.2}$$

where σ denotes the standard deviation. I refer to these nine elements as "sigma points." During training, the generated probabilities vary, but the differences they represent remain constant.

One issue is that the actual noise values are continuous, and directly computing $\tilde{p}(s'|s,a)$ is impossible. To address this, I let the environment return the next state both with and without noise at each step. Then, I calculate the difference to determine the actual noise added to the state. I match the difference to the closest "sigma point" and assign the probability of this "sigma point" to $\tilde{p}(s'|s,a)$.

Example: if we take action 1 at state 4.4, the next state should be 4.5, but we get 4.52. We first obtain the difference—0.02. Then we find that the distance between 0.02 and the 8th "sigma point" is the shortest (as shown in Figure 4.2). Thus, we assign the probability of the 8th "sigma point" to the transition $\tilde{p}(4.52|4.4, 1)$.

Actual Transition Probability For the actual transition probability $p(s'|s, a)$, we first obtain the difference between the transition with and without Gaussian noise. Then, we calculate the probability corresponding to this difference in the Gaussian distribution by integrating the probability density in the nearby interval.

Example: if we take action 1 at state 4.4, and the next state should be 4.5, but we get 4.52, we first obtain the difference—0.02. Then we calculate the sum of the probability density over the interval $[0.02 - \epsilon, 0.02 + \epsilon]$ to get the transition probability $p(4.52|4.4, 1)$. In this experiment, I pick $\epsilon = \sigma \cdot \sqrt{2 \log 2}/4$.

Chapter 5

Experiment

In this chapter, we include the experimental results of the proposed method. There are in total three experiments conducted in this chapter.

The first experiment, in section 5.1, is to validate the correctness of the proposed method in grid world, e.g., frozenlake.

The second experiment, in section 5.2, is to compare the performance of the proposed method with the baseline methods in the more complex control environment, e.g., CartPole.

The third experiment, in section 5.3, is to evaluate the performance of the proposed method in more complex robot manipulation environment, such as Mujoco.

5.1 Validation - Grid World

5.1.1 Environmental Settings

States: In this environment, there are three kinds of states: ice surface, holes, and the destination. The top-left corner is the starting point, and the bottom-right corner is the destination. Additionally, there are several holes scattered throughout the environment. An episode ends when the agent reaches the destination, falls into a hole, or reaches the maximum number of steps. A sample environment is shown in Figure 5.1.

Actions: The agent has four possible actions. When attempting to move in a certain direction, there is a 0.2 probability of moving in each perpendicular direction and a 0.6 probability of

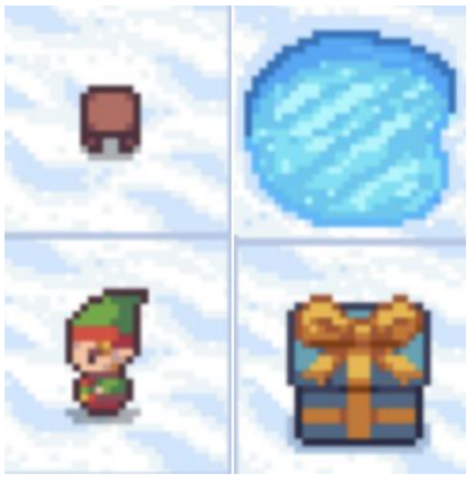
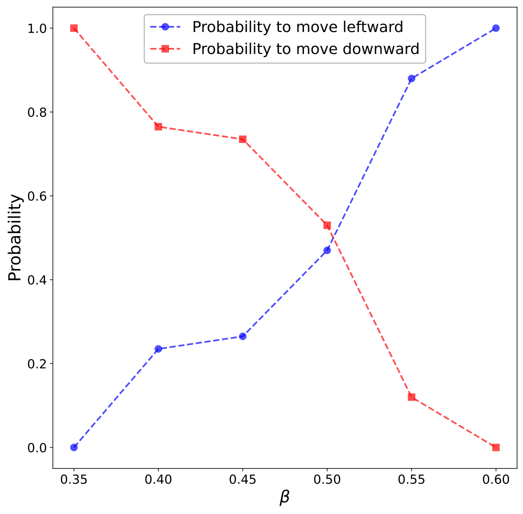



Figure 5.1: The Sample Environment of Frozen Lake (left) and Policy and Policy Change (right)

moving in the intended direction. If the next state falls outside the grid boundaries, the agent remains in its current position.

For more details, refer to Table 5.1.

Table 5.1: Frozenlake Environment Settings

Component	**Description**
States	Ice surface, Holes, Destination
Start Position	Top-left corner
End Condition	- Reach destination - Fall into hole - Max steps reached
Actions	0: Move left 1: Move down 2: Move right 3: Move up
Move Probabilities	0.6: Intended direction 0.2: Perpendicular directions
Boundaries	Stay in position if out of bounds
Rewards	+1: Reach destination 0: Max steps reached -0.3: Fall into hole

5.1.2 Frozenlake with Holes - Fixed Policy

These experiments aim to validate the gradient descent portion of Algorithm 1. We are specifically testing the first loop (for calculating the optimal perturbation), meaning the policy remains unchanged.

We first manually identify the optimal policy in the original environment to serve as our fixed policy, then compare the transitions before and after perturbation.

Intuitively, if we assign a large negative reward for falling into a hole (e.g., reward = -100), the agent starting at state 0 will likely avoid moving down, since that could lead to falling into the hole. Instead, the agent would prefer moving left, even though the probability of accidentally moving down is 0.2. This is still preferable to receiving a large penalty.

However, this is a trade-off: if the penalty is small (e.g., reward = -0.3), the agent may prefer moving down rather than moving conservatively to the left, which could result in exceeding the maximum steps and receiving a reward of 0. In this case, a "balance" in the penalty can be mathematically computed.

In this experiment, we identify the trade-off boundary (which is approximately -0.3).

Examples: Since this is a simple environment, I manually computed the optimal policy as follows:

- At state 0: move down
- At state 2: move right

The transitions before and after perturbation are as follows:

- At state 0, the true transition is [0.2, 0.2, 0.6, 0], while the perturbed transition is [0.21, 0.24, 0.55, 0]. After perturbation, the agent is more likely to fall into the hole but less likely to move to state 2. The action value function dropped from the initial value of 0.6 to 0.476.
- At state 2, the true transition is [0.2, 0, 0.2, 0.6], while the perturbed transition is [0.26, 0, 0.21, 0.52]. The action value function dropped from the initial value of 0.9 to 0.815.

If we replace the gradient descent over the action-value function in the first loop with gradient ascent, the transitions before and after perturbation are as follows:

- At state 0, the transition changes from [0.2, 0.2, 0.6, 0] to [0.2, 0.15, 0.65, 0]. After "perturbation", the agent is less likely to fall into the hole and more likely to move to state 2, which is close to the goal. The action value function rose from the initial value of 0.6 to 0.69.
- At state 2, the transition changes from [0.2, 0, 0.2, 0.6] to [0.15, 0, 0.17, 0.66]. The action value function rose from the initial value of 0.9 to 0.93.

Those results demonstrate the correctness of the proposed method.

5.1.3 Frozenlake with Holes - Policy Change

We also conducted the experiment to validate the second loop, which involves updating the policy. In this experiment, we define a uncertainty set by β, which is the KL divergence between the original transition and the perturbed transition. We gruadually increase the β to see how the policy shifts.

In Figure 5.1 (right), we can observe the policy changed more significantly when β is larger. The optimal poicy in the original environment is to move down at state 0 (moving downward). As β increases, the approved perturbations become larger, making it more likely to fall into a pit. Therefore, at this point, the agent needs to take a more conservative approach by moving leftward and approaching the goal with a higher probability of parallel movement. the allowed uncertainty set is larger,

5.1.4 Summary

The fixed policy and policy shifting experiments demonstrate our perturbation increases the risk of falling into holes and decreases the probability of reaching the destination.

As mentioned earlier, the pre-selected punishment leads to a trade-off: The agent can accept this punishment and maintain the same optimal policy as in the case where the punishment is zero.

If the probability of falling into a hole increases, which is equivalent to adding punishment, the balance will be broken. In such a case, applying the new transition will cause the agent to favor moving left at state 0.

5.2 Validation - Simple Gym Environment

In this section, I will present the experimental results of Policy Gradient and Actor-Critic methods.

5.2.1 Results: Policy Gradient

In the weak adversary experiment, I set the standard deviation to 0.003 and trained policy gradient agents both with and without considering adversaries. In the strong adversary experiment, I set the standard deviation to 0.01 and followed the same procedure to train and test the agents. The agents trained without considering adversaries serve as baselines. All of the curves are obtained by 15 agents with different seeds.

As shown in Figure 5.2, the policy gradient agent with our transition gradient method could achive better performance in the end of training.

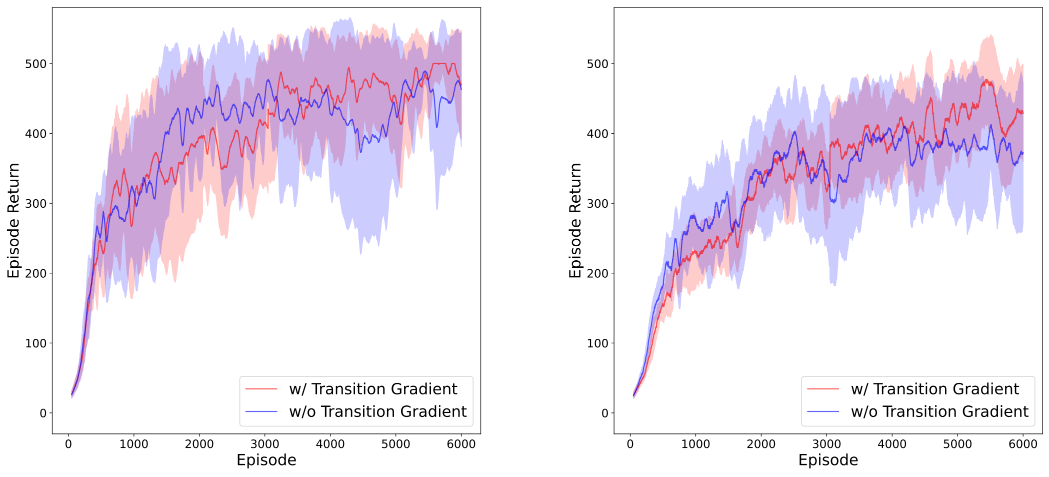


Figure 5.2: Comparison of the performance between policy gradient with and without Transitoin Gradient (left) under 0.003-std Gaussian noise (right) under 0.01-std Gaussian noise

5.2.2 Results: Actor-Critic

Due to the inherent advantages of the actor-critic approach, such as employing function approximators instead of relying solely on sampled returns, it exhibits stronger resistance to adversarial perturbations compared to policy gradient methods in practical experiments.

Therefore, to emphasize the effectiveness of the transition gradient, I increased the level of the adversarial challenge. In this experiment, the standard deviation for the weak adversary is set to 0.01, while for the strong adversary, it is set to 0.02. I keep the remaining setting the same as in the policy gradient experiment.

As shown in Figure 5.3, similar to the results in the policy gradient experiment, the actor-critic agent with transition gradient, could achive better performance in the end of training.

To demonstrate the robustness of the proposed method, I conducted experiments under Gaussian noise with standard deviations of 0.001, 0.01, 0.015, and 0.02, and then tested the saved models under different adversaries.

Saved models: The problem is considered solved when the average reward over the previous 100 episodes reaches 500 (the maximum reward). I save the model when the problem is solved. If the agent does not reach this goal during the entire training, I save the model with the highest average reward over 100 episodes.

As shown in Figure 5.4 and Figure 5.5, the agent trained with the transition gradient method is less sensitive to the adversaries, demonstrated by the less performance drop under higher level of

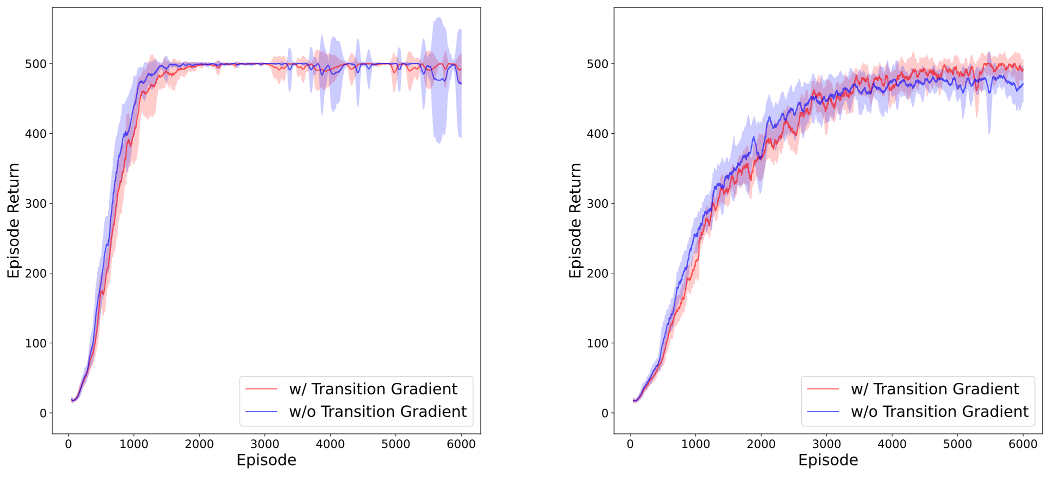


Figure 5.3: Comparison of the performance between actor critic with and without Transitoin Gradient (left) under 0.01-std Gaussian noise (right) under 0.02-std Gaussian noise

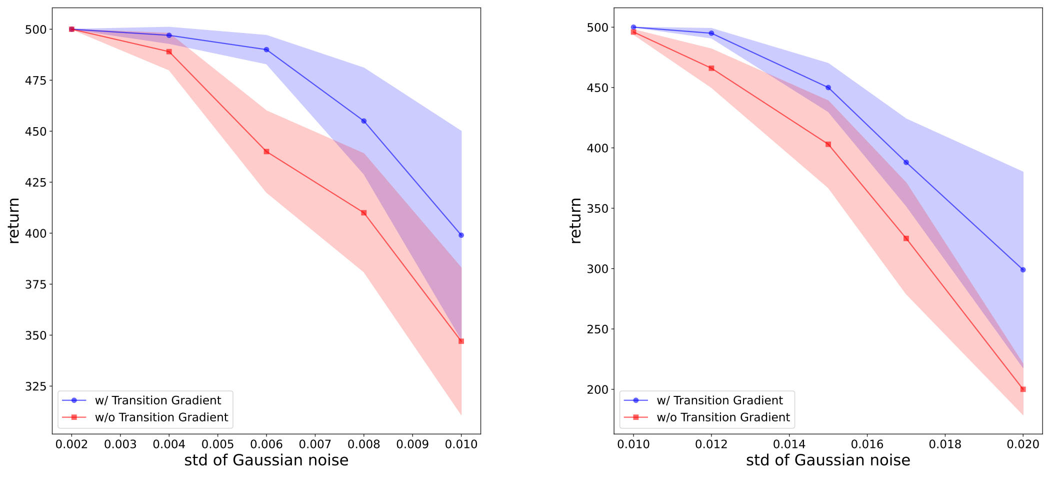


Figure 5.4: CartPole: Comparison the performance of agents training with (left) 0.001-std Gaussian noise (right) 0.01-std Gaussian noise

adversarial perturbations. These results suggest that the transition gradient method can effectively improve the robustness of the agent.

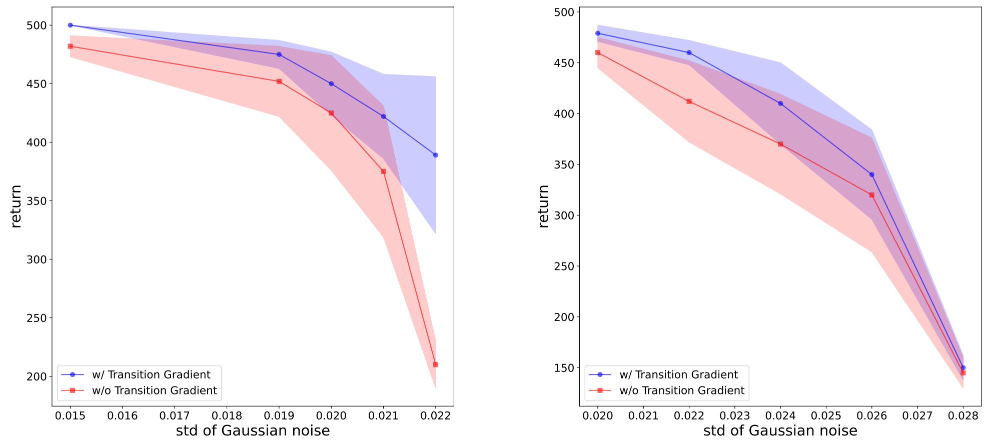


Figure 5.5: CartPole: Comparison the performance of agents training with (left) 0.015-std Gaussian noise (right) 0.02-std Gaussian noise

Table 5.2: Mean ± standard error of the returns under Weak Adversary, averaged across 15 agents over 50 episodes each.

Algorithm	Walker2d	Hopper	Half-Cheetah
DR-U PPO	**2473** ± **75**	1310 ± 33	1888 ± 37
DR-G PPO	2214 ± 66	1438 ± 31	1729 ± 23
TG-PPO	2200 ± 35	**1698** ± **24**	**2101** ± **28**

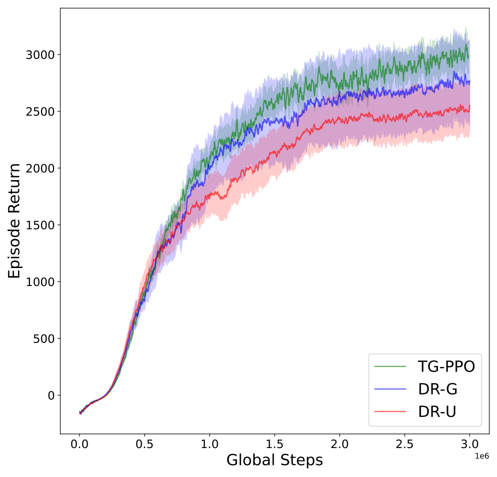
5.3 Mujoco Environments

5.3.1 Training

We utilized the high-quality benchmark implementation from Huang et al., 2022, and used their optimal hyperparameters without further tuning to establish our baseline. We verified our baseline by ensuring that locally trained PPO agents achieved similar performance to their reported results.

The same as in CartPole and Frozenlake, since all transitions in the Mujoco environment are deterministic and driven by the physics engine, we introduced stochasticity by adding Gaussian noise to the states representing velocity and angular velocity. We ran all environments for 3×10^6 steps to ensure convergence. Two baselines were included: (1) vanilla PPO and (2) PPO with Gaussian/Uniform Domain Randomization (DR), where the standard deviation of the Gaussian noise was set to 0.01. Our TG-PPO agents were trained in the same stochastic environment.

Figure 5.6, Figure 5.7 and 5.8 shows the episode returns during training. All curves are averaged over at least 15 independent runs, with shaded regions indicating standard errors. The **blue** line



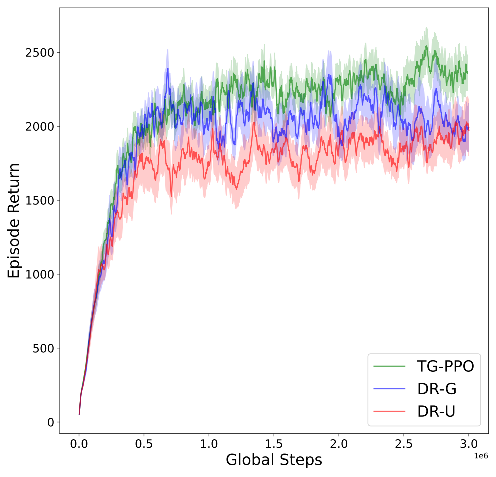
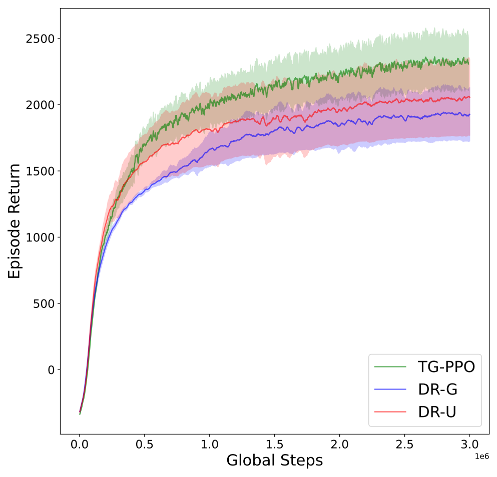


Figure 5.6: Training curves for (left) Walker2d-V4 and (right) Hopper-V4



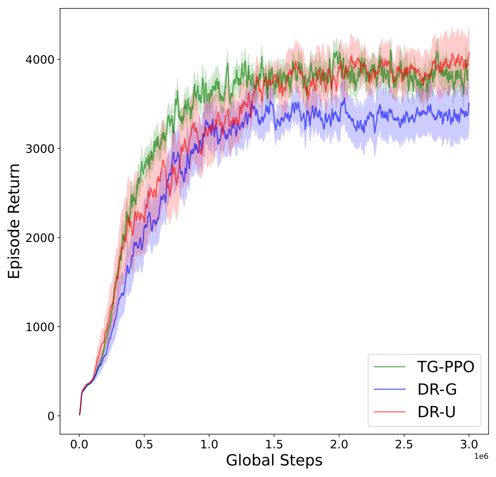


Figure 5.7: Training curves for (left) HalfCheetah-V4 and (right) Ant-V4

represents DR-PPO with Gaussian randomization (std = 0.01), the **red** line represents DR-PPO with uniform randomization (range: [-0.0173, 0.0173]), and the **green** line represents TG-PPO.

5.3.2 Evaluation

We evaluated both vanilla PPO, DR-U and TG-PPO by testing 15 agents per algorithm, trained in environments with varying noise levels (e.g., 1x noise: 0.01, 10x noise: 0.1). For each agent, we initialized the environment randomly and conducted tests across 50 episodes to calculate the average return for further analysis.

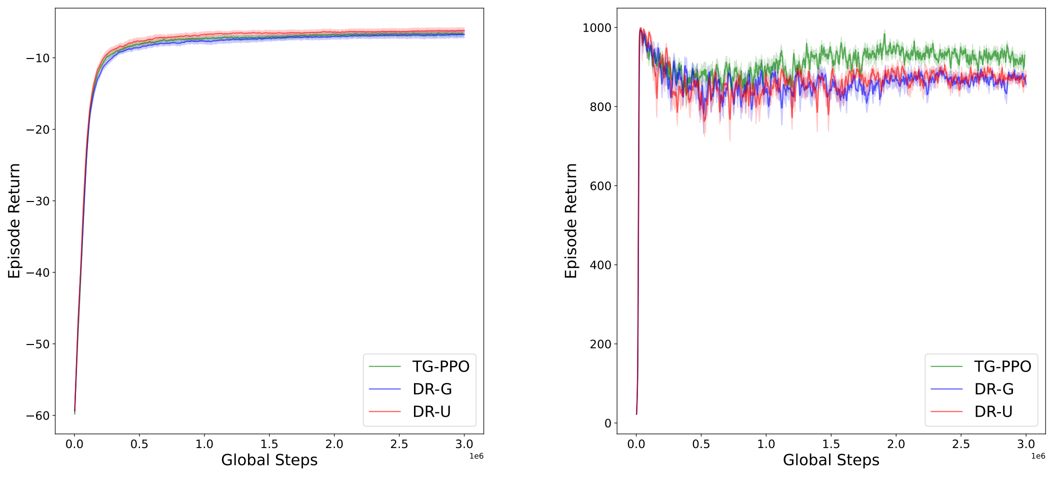


Figure 5.8: Training curves for (left) Reacher-V4, and (right) InvertedPendulum-V4.

Table 5.3: Mean ± standard error of the returns under Strong Adversary, averaged across 15 agents over 50 episodes each.

Algorithm	Walker2d	Hopper	Half-Cheetah
DR-U PPO	1815 ± 64	1061 ± 27	1271 ± 19
DR-G PPO	1850 ± 62	1204 ± 26	1154 ± 15
TG-PPO	**1878 ± 31**	**1392 ± 21**	**1288 ± 14**

Table 5.2, 5.3, 5.4 and Table 5.5 show the mean and the standard error of the returns. We can ovserve that TG-PPO outperforms both DR-PPO and vanilla PPO in all environments under the strong adversary.

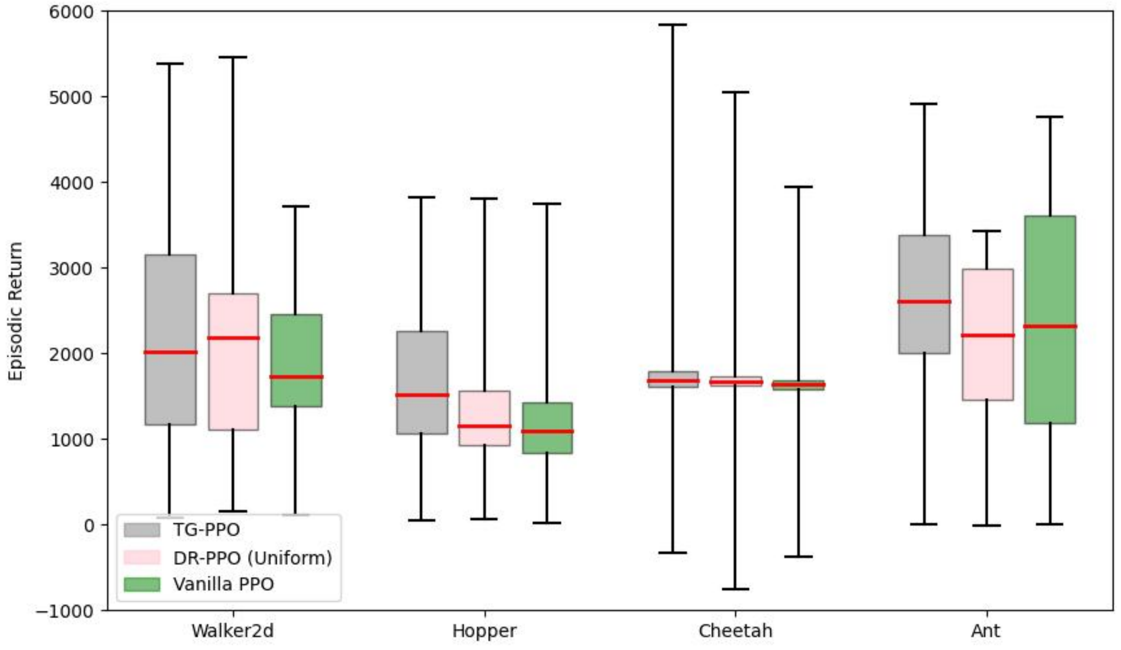
Figure 5.9 shows the box plots of the returns for Vanilla PPO, DR-PPO and TG-PPO. We can also observe that when the adversary goes from weak to strong, TG-PPO outperforms both DR-PPO and vanilla PPO in all environments.

Table 5.4: Mean ± standard error of the returns under Weak Adversary, averaged across 15 agents over 50 episodes each.

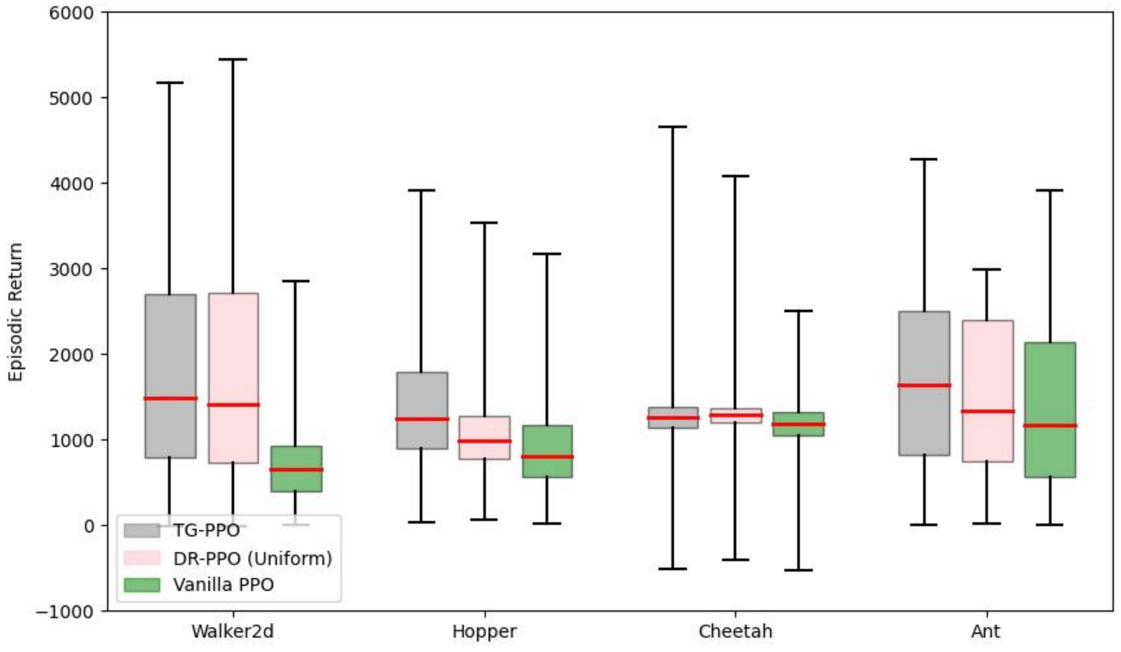
Algorithm	InvertedPendulum	Ant	Reacher
DR-U PPO	822 ± 14	2152 ± 45	-13.6 ± 0.16
DR-G PPO	835 ± 13	2305 ± 66	-14.3 ± 0.18
TG-PPO	**846 ± 9.7**	**2598 ± 29**	**-13.5 ± 0.09**

Table 5.5: Mean ± standard error of the returns under Strong Adversary, averaged across 15 agents over 50 episodes each.

Algorithm	InvertedPendulum	Ant	Reacher
DR-U PPO	681 ± 17	1451 ± 41	-18.2 ± 0.13
DR-G PPO	759 ± 15	1420 ± 52	-17.4 ± 0.13
TG-PPO	**783 ± 10**	**1688 ± 26**	**-17.1 ± 0.08**



(a) In environments with weak adversary



(b) In environments with strong adversary

Figure 5.9: Box plots of returns for DR-PPO and TG-PPO. Each box represents results from at least 15 agents trained with the same hyperparameters as those in Table 5.2. Each agent was tested for 50 episodes. The red lines inside the boxes represent the median rewards, and the upper and lower edges of the boxes indicate the 25th and 75th percentile returns. Line segments outside the boxes indicate minimum and maximum rewards.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed a novel framework to address the problem of Robust Reinforcement Learning (RRL). This framework is based on a modified Markov Decision Process (MDP), enabling seamless integration into existing reinforcement learning algorithms. We began by defining this new MDP, drawing inspiration from (Iyengar, 2005), and analyzed its theoretical properties. Motivated by (H. Zhang et al., 2020), we demonstrated that solving for an optimal adversary under a given policy could be reduced to solving a standard MDP.

To overcome the limitations of existing methods in handling complex environments, we relaxed the original strict constraints into soft constraints and introduced a novel optimization objective. This objective was addressed using a bilevel optimization approach, and we derived an upper bound on the gap between the value function under worst-case perturbations and the interference-free case.

In our experiments, we conducted extensive evaluations across environments of varying complexity, including gridworld, simple gym environments, and complex robotic manipulation tasks. The results showcased the robustness and effectiveness of our approach. However, in environments with significantly extended time horizons, the practical implementation of our framework demonstrated limitations in scalability and computational efficiency, highlighting areas for future improvement.

However, we note that the bilevel optimization framework may face challenges in long-horizon problems, where compounding errors and the complexity of the adversarial dynamics can significantly reduce its effectiveness. Additionally, our approach for generating transitions relies on sigma points, where the number of sigma points and the standard deviation (`std`) used in experiments are manually specified. This sensitivity to hyperparameter settings limits the adaptability of the framework to different tasks and environments. Developing methods for automatic hyperparameter tuning or adaptive sigma point generation would significantly enhance its applicability to a broader range of tasks. Addressing these issues is critical for extending the applicability of our framework to real-world, long-horizon tasks.

6.2 Future Work

During our research, several challenges remained unresolved, providing opportunities for future exploration and development.

1. **Convergence Analysis for Policy Gradient**: Optimizing the optimal perturbation through a policy gradient approach introduces a non-convex, non-concave optimization problem, complicating convergence analysis. While our experiments demonstrated the effectiveness of the proposed framework, a formal convergence analysis remains an open question. Future work will focus on providing theoretical guarantees for convergence to ensure a deeper understanding of the method's behavior under different perturbation scenarios.

2. **Scalability to Long-Horizon Problems**: One of the key limitations of our framework lies in its potential inefficiency in long-horizon problems. As the horizon lengthens, compounded errors and adversarial dynamics become increasingly significant, potentially reducing the effectiveness of our bilevel optimization approach. Addressing this issue will involve developing hierarchical or approximate methods that can better manage the complexity of long-horizon tasks, while maintaining computational feasibility.

3. **Integration with Real-World Applications**: While our experiments included a range of simulated environments, real-world applications often involve higher levels of uncertainty, more complex dynamics, and resource constraints. Future work will focus on adapting the framework to real-world scenarios, such as robotic manipulation and autonomous navigation, where robust decision-making under dynamic and adversarial conditions is critical.

Bibliography

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565.*

Banerjee, S., & Ghosh, A. (2021). Adversarially robust bellman operators in reinforcement learning. *arXiv preprint arXiv:2107.00754.*

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, *13*(5), 834–846.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bellman, R. (1957). *Dynamic programming.* Princeton University Press.

Ben-Tal, A., El Ghaoui, L., & Nemirovski, A. (2009). Robust solutions to uncertain semidefinite programs. *Management Science*, *52*(4), 597–618.

Bertsimas, D., Brown, D. B., & Caramanis, C. (2011). *The theory and practice of robust optimization.* Society for Industrial; Applied Mathematics (SIAM).

Blanchet, J., Kang, Y., & Murthy, K. (2019). Quantifying distributional model risk via optimal transport. *Mathematics of Operations Research*, *44*(2), 565–600.

Esfahani, P. M., & Kuhn, D. (2018). Data-driven distributionally robust optimization using the wasserstein metric: Performance guarantees and tractable reformulations. *Mathematical Programming*, *171*(1), 115–166.

Fujimoto, S., Van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning*, 1587–1596.

Garcia, J., & Fernandez, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, *16*, 1437–1480.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proceedings of the 35th International Conference on Machine Learning (ICML).*

He, J., & Lv, C. (2023). Observation-robust reinforcement learning framework for autonomous driving with perception uncertainties. *International Journal of Intelligent Transportation Systems Research.*

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Tassa, Y., Erez, T., & Riedmiller, M. (2015). Learning continuous control policies by stochastic value gradients. *Advances in Neural Information Processing Systems*, 2944–2952.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., & Silver, D. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286.*

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., & Araújo, J. G. (2022). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, *23*(274), 1–18.

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

Jansen, J. (2003). Robust reinforcement learning in continuous spaces: Time complexity and stability of optimistic algorithms. *Neurocomputing*, *55*(1-2), 319–340.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. *Advances in Neural Information Processing Systems*, *12*, 1008–1014.

Kothari, P., Kakade, S., & Jiang, N. (2020). Agnostic reinforcement learning with low-rank mdps and rich observations. *Advances in Neural Information Processing Systems (NeurIPS).*

Kuang, Y., Lu, M., Wang, J., Zhou, Q., Li, B., & Li, H. (2022). Learning robust policy against disturbance in transition dynamics via state-conservative policy optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*, 7247–7254.

Kumar, A., Mahajan, D., Xiao, C., Liu, B., & Gleicher, M. (2020). Adversarial robustness for deep reinforcement learning policies. *International Conference on Learning Representations.*

Lee, K., Raghu, M., & Matni, N. (2020). Sparse perturbation analysis for policy robustness in reinforcement learning. *Advances in Neural Information Processing Systems*, 13479–13489.

Li, Y., & Lan, G. (2023). First-order policy optimization for robust policy evaluation. *arXiv preprint arXiv:2307.15890.*

Liu, H., Hu, J., & Fan, F. (2022). Robustness of safe reinforcement learning under observational perturbations. *arXiv preprint arXiv:2205.14691.*

Mankowitz, D. J., Mann, T. A., Bacon, P.-L., Precup, D., & Mannor, S. (2018). Learning robust options. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of the International Conference on Machine Learning (ICML)*, 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Moore, A. W. (1990). Efficient memory-based learning for robot control. *Proceedings of the AAAI Conference on Artificial Intelligence*, 73–78.

Morimoto, J., & Doya, K. (2005a). Robust reinforcement learning. *Neural Computation*, *17*(2), 335–359.

Morimoto, J., & Doya, K. (2005b). Robust reinforcement learning. *Neural Computation*, *17*(2), 335–359.

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

Pan, Z., Yu, W., Zheng, W., Moura, J. M., Sadigh, D., & Zhou, M. T. (2021). Robust reinforcement learning through dynamic adversarial training. *IEEE Transactions on Intelligent Transportation Systems*, *22*(6), 3773–3786.

Pinto, L., Davidson, J., Sukthankar, R., & Gupta, A. (2017). Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702.*

Plappert, M., Andrychowicz, M., Ray, A., Wolski, F., McGrew, B., Cabi, S., Sinha, V., Stooke, A., Ho, J., Schneider, J., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464.*

Rummery, G. A., & Niranjan, M. (1994). *On-line q-learning using connectionist systems* (tech. rep.). University of Cambridge, Department of Engineering.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *International Conference on Learning Representations.*

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, *588*(7839), 604–609.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust region policy optimization. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438.*

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

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 1038–1044.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 1057–1063.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., & Abbeel, P. (2018). Exploration: A study of count-based exploration for deep reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2753–2762.

Tessler, C., Efroni, Y., & Mannor, S. (2019). Adapting to distributional shift in reinforcement learning. *Advances in Neural Information Processing Systems*, 17457–17468.

Tessler, C., Jinnai, Y., Efroni, Y., & Brunskill, E. (2019). Action robustness and regularization in reinforcement learning. *arXiv preprint arXiv:1908.08681*.

Thrun, S. B. (1992). *Efficient exploration in reinforcement learning* (tech. rep. CMU-CS-92-102). Carnegie Mellon University, Department of Computer Science.

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, *48*, 5026–5033.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *Proceedings of the AAAI conference on artificial intelligence*, *30*(1).

Vinitsky, E., Delle Fave, F., Jiang, X., & Bayen, A. M. (2020). Iterative robustness optimization for reinforcement learning policies. *IEEE Transactions on Intelligent Transportation Systems*, *21*(12), 5208–5219.

Wang, Y., Velasquez, A., Atia, G., Prater-Bennette, A., & Zou, S. (2024). Robust average-reward reinforcement learning. *Journal of Artificial Intelligence Research*, *80*, 719–803.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H. v., Lanctot, M., & de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *International conference on machine learning*, 1995–2003.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3-4), 279–292.

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

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

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3-4), 229–256.

Xu, H., & Mannor, S. (2010). Distributionally robust markov decision processes. *Advances in Neural Information Processing Systems*, 2505–2513.

Zhang, H., Liu, Y., Li, Y., Zhao, Z., & Li, B. (2020). Robust deep reinforcement learning against adversarial perturbations on state observations. *arXiv preprint arXiv:2003.08938*.

Zhang, H., Javanmard, A., Dhillon, I. S., & Hsieh, C.-J. (2021). Stability of q-learning with linear function approximation under adversarial perturbations. *arXiv preprint arXiv:2102.09831*.

Zhang, H., Yu, H., Javanmard, A., Hsieh, C.-J., & Dhillon, I. S. (2019). Towards stable and robust reinforcement learning under adversarial perturbations. *arXiv preprint arXiv:1906.01179*.

Zhang, J., & Xu, L. (2021). Adversarially robust deep reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(8), 3674–3686.

Zhu, H., Zhao, Y., Yang, Y., Hong, M., Xu, Z., Zhang, T., & Huang, H. (2020). Robustness to adversarial attacks in reinforcement learning via robust adversarial reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*.

Appendix A

Proof Details

Theorem 1 Proof Given the definition of $\tilde{V}_{\tilde{p},\pi}(s)$, we have:

$$\begin{aligned}
\tilde{V}_{\tilde{p},\pi}(s) &:= \mathbb{E}_{\tilde{p},\pi}[\sum_{k=0}^{\infty}\gamma^k r_{r+k+1}|s_t = s] \\
&= \mathbb{E}_{\tilde{p},\pi}[r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{r+k+2}|s_t = s] \\
&= \sum_{a\in A}\pi(a|s)\sum_{s'\in S}[\tilde{p}(s'|s,a)][r_{t+1} + \gamma\mathbb{E}_{\tilde{p},\pi}[\sum_{k=0}^{\infty}\gamma^k r_{r+k+2}|s_{t+1} = s']] \\
&= \sum_{a\in A}\pi(a|s)\sum_{s'\in S}\tilde{p}(s'|s,a)[r_{t+1} + \gamma\mathbb{E}_{\tilde{p},\pi}[\sum_{k=0}^{\infty}\gamma^k r_{r+k+2}|s_{t+1} = s']] \\
&= \sum_{a\in A}\pi(a|s)\sum_{s'\in S}\tilde{p}(s'|s,a)[R(s,a,s') + \gamma\tilde{V}_{\tilde{p},\pi}(s')]
\end{aligned} \tag{A.1}$$

Then we can easily derive $\tilde{Q}_{\tilde{p},\pi}$ following relationship between value function and state-value function. ■

Lemma 1 proof For an TA-MDP $M = (\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$ and a fixed π, we define a regular MDP $\hat{M} = (\hat{\mathcal{S}}, \hat{\mathcal{A}}, \hat{R}, \hat{p}, \gamma)$, such that $\hat{\mathcal{S}} = \mathcal{S}\times\mathcal{A}$, $\hat{a}\in\hat{\mathcal{A}} : \mathcal{S}\times\mathcal{A}\to\mathcal{P}(\mathcal{S})$. To prove this lemma, we use an extention of stochastic adversary policy $\hat{\pi}(\cdot|s,a)$, where $\hat{\pi} : \mathcal{S}\times\mathcal{A}\to\mathcal{P}(\hat{\mathcal{A}})$. $\hat{\pi}$ gives the probability that we perturb a transition $p(\cdot|s,a)$ to $\hat{a}(\cdot|s,a)$. For the transition $\hat{p}(\hat{s}'|\hat{s},\hat{a})$:

$$\begin{aligned}
\hat{p}(\hat{s}'|\hat{s},\hat{a}) &= \hat{p}(s',a'|s,a,\hat{a}) \\
&= \hat{a}(s'|s,a)\cdot\pi(a'|s')
\end{aligned} \tag{A.2}$$

Reward $\hat{R}(\hat{s},\hat{a},\hat{s}')$ is defined as:

$$\hat{R}(\hat{s},\hat{a},\hat{s}') = \begin{cases} \hat{R}(s,a,\hat{a},s',a') = -R(s,a,s') & \text{if } \hat{a}\in\mathcal{B}(s,a) \\ C & \text{if } \hat{a}\notin\mathcal{B}(s,a) \end{cases} \tag{A.3}$$

where the constant C satisfies:

$$C < \min\{-\overline{M}, \frac{\gamma}{1-\gamma}\underline{M} - \frac{1}{a-\gamma}\overline{M}\} \tag{A.4}$$

where $\underline{M}$ means the minimum of $R(s,a,s')$ and $\overline{M}$ means the maximum of $R(s,a,s')$.

Therefore we have the state value function $\hat{V}_{\hat{\pi}}$:

$$\begin{aligned}\hat{V}_{\hat{\pi}}(\hat{s}) &= \sum_{\hat{a}\in\hat{A}} \hat{\pi}(\hat{a}|\hat{s}) \sum_{\hat{s}'\in\hat{S}} \hat{p}(\hat{s}'|\hat{s},\hat{a})[\hat{R}(\hat{s},\hat{a},\hat{s}') + \gamma\hat{V}_{\hat{\pi}}(\hat{s}')] \\ &= \sum_{\hat{a}\in\hat{A}} \hat{\pi}(\hat{a}|\hat{s}) \sum_{s'\in S} \hat{a}(s'|s,a) \cdot \sum_{a'\in A} \pi(a'|s')[-R(s,a,s') + \gamma\hat{V}_{\hat{\pi}}(\hat{s}')]\end{aligned} \tag{A.5}$$

The optimal policy $\hat{\pi}^*$ in this case should maximize the expected reward $\hat{V}_{\hat{\pi}}$, which means $\forall s, \forall a, \hat{V}_{\hat{\pi}^*}(\hat{s}) \geq \hat{V}_{\hat{\pi}}(\hat{s}), \hat{Q}_{\hat{\pi}^*}(\hat{s},\hat{a}) \geq \hat{Q}_{\hat{\pi}}(\hat{s},\hat{a})$ and the optimal value function and action-value function will be:

$$\forall\hat{s}, \forall\hat{a}, \hat{V}_{\hat{\pi}^*}(\hat{s}) = \max_{\hat{\pi}} \hat{V}_{\hat{\pi}}(\hat{s}), \quad \hat{Q}_{\hat{\pi}^*}(\hat{s},\hat{a}) = \max_{\hat{\pi}} \hat{Q}_{\hat{\pi}}(\hat{s},\hat{a})$$

Recall Equation (A.4), if C is small enough, we can guarantee that optimal $\hat{\pi}^*$ will not choose $\hat{a}$ out of $\mathcal{B}(s,a)$, and below is the proof:
For those $\hat{a} \notin \mathcal{B}(s,a)$:

$$\begin{aligned}\hat{V}_{\hat{\pi}^*}(\hat{s}) &= \mathbb{E}_{\hat{\pi}^*}[\sum_{k=0}^{\infty} \gamma^k \hat{r}_{t+k+1}|\hat{s}_t = \hat{s}] \\ &= C + \mathbb{E}_{\hat{\pi}^*}[\sum_{k=1}^{\infty} \gamma^k \hat{r}_{t+k+1}|\hat{s}_t = \hat{s}] \\ &\leq C - \frac{\gamma}{1-\gamma}\underline{M} \quad = C + \sum_{k=1}^{\infty}\gamma^k(-\underline{M}) \\ &< -\frac{1}{1-\gamma}\overline{M} \quad (\text{Note: } C < \min\{-\overline{M}, \frac{\gamma}{1-\gamma}\underline{M} - \frac{1}{1-\gamma}\overline{M}\}) \\ &\leq \mathbb{E}_{\hat{\pi}'}[\sum_{k=0}^{\infty} \gamma^k \hat{r}_{t+k+1}|\hat{s}_t = \hat{s}] \quad (\text{Note: } \mathbb{E}_{\hat{\pi}'}[\sum_{k=0}^{\infty} \gamma^k \hat{r}_{t+k+1}|\hat{s}_t = \hat{s}] \geq \sum_{k=0}^{\infty}\gamma^k(-\overline{M})) \\ &= \hat{V}_{\hat{\pi}'}(\hat{s})\end{aligned} \tag{A.6}$$

The last inequality holds for any $\hat{\pi}'$. In this case $\hat{V}_{\hat{\pi}^*}(\hat{s}) < \hat{V}_{\hat{\pi}'}(\hat{s})$,which contradicts the assumption that $\hat{\pi}^*$ is optimal. Therefore it's evidence that the occurrence of $\hat{a} \notin \mathcal{B}(s,a)$ is impossible under a small enough C.

According to the basic properties of MDP, we know that $\hat{M}$ has a optimal policy $\hat{\pi}^*$. We also know that this $\hat{\pi}^*$ can be deterministic and will be assigned unit mass probability for optimal action on each (s,a).

$\hat{\pi}$ is deterministic means $\hat{\pi} \in \Omega = \{\hat{\pi} : \forall\hat{s}, \exists\hat{a} \in \mathcal{B}(s,a) \text{ let } \hat{\pi}(\hat{a}|\hat{s}) = 1\}$, since we have disscussed the case that $\hat{\pi} \notin \Omega$ in Equation (A.6) (and we can avoid this case by choosing a small constant C), we only discuss cases that $\hat{\pi} \in \Omega$. Note that $\hat{\pi}^*$ is also in Ω.

Since $\hat{\pi}$ is deterministic, we can always find $\hat{a}$ that makes $\hat{\pi}(\hat{a}|s,a) = 1$, and $\text{argmax}_{\hat{a}}[\hat{\pi}(\hat{a}|s,a)]$ could be represented as a deterministic function $\tilde{p}_{\hat{\pi}}(s,a)$.

Given transition $p(s,a)$, action a and state s, $\tilde{p}_{\hat{\pi}}(s,a)$ will be a function. Then we inverse the sign of each term of Equation (A.5), after which we have:

$$\begin{aligned}
-\hat{V}_{\hat{\pi}}(\hat{s}) &= -\sum_{s'\in S}\tilde{p}_{\hat{\pi}}(s'|s,a)\cdot\sum_{a'\in A}\pi(a'|s')[-R(s,a,s')+\gamma\hat{V}_{\hat{\pi}}(\hat{s}')] \\
&= \sum_{s'\in S}\tilde{p}_{\hat{\pi}}(s'|s,a)\cdot\sum_{a'\in A}\pi(a'|s')[R(s,a,s')+(-\gamma\hat{V}_{\hat{\pi}}(\hat{s}'))] \\
&= \sum_{s'\in S}\tilde{p}_{\hat{\pi}}(s'|s,a)\cdot[\sum_{a'\in A}\pi(a'|s')R(s,a,s')+\gamma\sum_{a'\in A}\pi(a'|s')(-\hat{V}_{\hat{\pi}}(\hat{s}'))] \\
&= \sum_{s'\in S}\tilde{p}_{\hat{\pi}}(s'|s,a)\cdot[R(s,a,s')+\gamma\sum_{a'\in A}\pi(a'|s')(-\hat{V}_{\hat{\pi}}(\hat{s}'))]
\end{aligned} \tag{A.7}$$

We recall Equation (3.3) first:

$$\begin{aligned}
\tilde{Q}_{\tilde{p},\pi}(s,a) &= \sum_{s'\in S}\tilde{p}(s'|s,a)[R(s,a,s')+\gamma\sum_{a'\in A}\pi(a'|s')\tilde{Q}_{\tilde{p},\pi}(s',a')] \\
&= \sum_{s'\in S}\tilde{p}(s'|s,a)[R(s,a,s')+\gamma\sum_{a'\in A}\pi(a'|s')\tilde{Q}_{\tilde{p},\pi}(s',a')]
\end{aligned}$$

Note if we use the same perturbation function $\tilde{p}_{\hat{\pi}}$ in $\tilde{Q}_{\tilde{p}}(s,a)$, we have:

$$\tilde{Q}_{\tilde{p}_{\hat{\pi}}}(s,a) = \sum_{s'\in S}\tilde{p}_{\hat{\pi}}(s'|s,a)[R(s,a,s')+\gamma\sum_{a'\in A}\pi(a'|s')\tilde{Q}_{\tilde{p}_{\hat{\pi}}}(s',a')] \tag{A.8}$$

Compared Equation (A.7) with Equation (A.8), we have $-\hat{V}_{\hat{\pi}}(\hat{s}) = \tilde{Q}_{\tilde{p}}(s,a)$. This property proves that we can find an optimal policy of a regular MDP $\hat{M}$ that could determine the optimal adversary $\tilde{p}^*$ for TA-MDP.
The optimal value function $\hat{V}_{\hat{\pi}^*}$ satisfies:

$$\hat{V}_{\hat{\pi}^*}(\hat{s}) = \max_{\tilde{p}_{\hat{\pi}}(s,a)\in\mathcal{B}(s,a)}\sum_{s'\in S}\tilde{p}_{\hat{\pi}}(s'|s,a)\cdot[-R(s,a,s')+\gamma\sum_{a'\in A}\pi(a'|s')\hat{V}_{\hat{\pi}^*}(\hat{s}')] \tag{A.9}$$

■

Theorem 2 Proof In Lemma 1 we already establish $\hat{V}_{\hat{\pi}}(\hat{s}) = -\tilde{Q}_{\tilde{p},\pi}(s,a)$, which shows the process that we find the optimal adversary under a fixed π could be done like obtaining the optimal policy in a regular MDP.

The difference is that, we use $\max_{\hat{\pi}}$ to get the optimal policy $\hat{\pi}$ in regular MDP; But in our case, we use $\min_f$ to find the optimal perturbation $\tilde{p}^*$.

Firstly by the definition of $\tilde{Q}_{\tilde{p}^*,\pi}(s,a)$, we have:

$$\begin{aligned}
\tilde{Q}_{\tilde{p}^*(\pi)}(s,a) &:= \min_{\tilde{p}} \tilde{Q}_{\tilde{p}}(s,a) \\
&= \min_{\tilde{p}} \mathbb{E}_{\tilde{p}}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \\
&= \min_{\tilde{p}} \mathbb{E}_{\tilde{p}}[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a] \\
&= \min_{\tilde{p}} \sum_{s' \in S} [\tilde{p}(s'|s,a)][r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') \mathbb{E}_{\tilde{p}}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s', a_{t+1} = a']] \\
&= \min_{\tilde{p}(s,a) \in \mathcal{B}(s,a)} \sum_{s' \in S} \tilde{p}(s'|s,a)[r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') \min_{\tilde{p}} \mathbb{E}_{\tilde{p}}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s', a_{t+1} = a']] \\
&= \min_{\tilde{p}(s,a) \in \mathcal{B}(s,a)} \sum_{s' \in S} \tilde{p}(s'|s,a)[r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') \tilde{Q}_{\tilde{p}^*,\pi}(s',a')] \\
&= \mathcal{T} \tilde{Q}_{\tilde{p}^*(\pi)}(s,a)
\end{aligned}$$

Where the $\tilde{p}^*$ is a fixed point of the Bellman operator. Now we will demonstrate this Bellman operator is a contraction.

For arbitrary two initialized $\tilde{Q}_{\tilde{p}_1,\pi}(s,a)$ and $\tilde{Q}_{\tilde{p}_2,\pi}(s,a)$, if $\mathcal{T}\tilde{Q}_{\tilde{p}_1,\pi}(s,a) \geq \mathcal{T}\tilde{Q}_{\tilde{p}_2,\pi}(s,a)$, and with the inequality that:

$$\min_{x_1} g(x_1) - \min_{x_2} k(x_2) \leq g(x_2^*) - k(x_2^*) \leq \max_x (g(x) - k(x))$$

where $x_2^* = \text{argmin}_x k(x)$, we have:

$$\begin{aligned}
\forall s, \forall a \quad & \mathcal{T}\tilde{Q}_{\tilde{p}_1,\pi}(s,a) - \mathcal{T}\tilde{Q}_{\tilde{p}_2,\pi}(s,a) \\
&= \min_{\tilde{p}_1(s,a) \in \mathcal{B}(s,a)} \sum_{s' \in S} \tilde{p}_1(s'|s,a)[r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') \tilde{Q}_{\tilde{p}_1,\pi}(s',a')] \\
&- \min_{\tilde{p}_2(s,a) \in \mathcal{B}(s,a)} \sum_{s' \in S} \tilde{p}_2(s'|s,a)[r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') \tilde{Q}_{\tilde{p}_2,\pi}(s',a')] \\
&\leq \max_{\tilde{p}(s,a) \in \mathcal{B}(s,a)} \{ \sum_{s' \in S} \tilde{p}(s'|s,a)[r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') \tilde{Q}_{\tilde{p}_1,\pi}(s',a')] \\
&- \sum_{s' \in S} \tilde{p}(s'|s,a)[r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') \tilde{Q}_{\tilde{p}_2,\pi}(s',a')] \} \\
&= \gamma \max_{\tilde{p}(s,a) \in \mathcal{B}(s,a)} \sum_{s' \in S} \tilde{p}(s'|s,a) \sum_{a' \in A} \pi(a'|s') [\tilde{Q}_{\tilde{p}_1,\pi}(s',a') - \tilde{Q}_{\tilde{p}_2,\pi}(s',a')] \\
&\leq \gamma \max_{\tilde{p}(s,a) \in \mathcal{B}(s,a)} \sum_{s' \in S} \tilde{p}(s'|s,a) \sum_{a' \in A} \pi(a'|s') \|\tilde{Q}_{\tilde{p}_1,\pi} - \tilde{Q}_{\tilde{p}_2,\pi}\|_\infty \\
&= \gamma \|\tilde{Q}_{\tilde{p}_1,\pi} - \tilde{Q}_{\tilde{p}_2,\pi}\|_\infty
\end{aligned}$$

Similarly, we can prove the case that $\mathcal{T}\tilde{Q}_{\tilde{p}_2,\pi}(s,a) \geq \mathcal{T}\tilde{Q}_{\tilde{p}_1,\pi}(s,a)$, therefore we have $\forall s, \forall a$, $|\mathcal{T}\tilde{Q}_{\tilde{p}_1,\pi}(s,a) - \mathcal{T}\tilde{Q}_{\tilde{p}_2,\pi}(s,a)| \leq \gamma \|\tilde{Q}_{\tilde{p}_1,\pi} - \tilde{Q}_{\tilde{p}_2,\pi}\|_\infty$. Hence, we have:

$$\|\mathcal{T}\tilde{Q}_{\tilde{p}_1,\pi}(s,a) - \mathcal{T}\tilde{Q}_{\tilde{p}_2,\pi}(s,a)\|_\infty = \max_{s,a} |\mathcal{T}\tilde{Q}_{\tilde{p}_1,\pi}(s,a) - \mathcal{T}\tilde{Q}_{\tilde{p}_2,\pi}(s,a)| \leq \gamma \|\tilde{Q}_{\tilde{p}_1,\pi} - \tilde{Q}_{\tilde{p}_2,\pi}\|_\infty$$

Then according to the Banach fixe-point theorem, since $0 < \gamma < 1$, $\lim_{k \to \infty} \mathcal{T}^k \tilde{Q}_{\tilde{p},\pi}$ converge to a unique fixed point, which is $\tilde{Q}_{\tilde{p}^*(\pi)}$ ■

Theorem 3 proof We start with $Q_{\pi,\theta}(s,a)$,

$$\begin{aligned}
\nabla_\theta Q_{\pi,\theta}(s,a) &= \nabla_\theta \left\{ \sum_{s'\in\mathcal{S}} \tilde{p}_\theta(s'|s,a) \left[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi,\theta}(s',a') \right] \right\} \\
&= \nabla_\theta \left[\sum_{s'\in\mathcal{S}} \tilde{p}_\theta(s'|s,a) \right] \left[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi,\theta}(s',a') \right] \\
&+ \left[\sum_{s'\in\mathcal{S}} \tilde{p}_\theta(s'|s,a) \right] \nabla_\theta \left[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi,\theta}(s',a') \right] \\
&= \left[\sum_{s'\in\mathcal{S}} \nabla_\theta \tilde{p}_\theta(s'|s,a) \right] \left[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi,\theta}(s',a') \right] \\
&+ \gamma \sum_{s'\in\mathcal{S}} \tilde{p}_\theta(s'|s,a) \sum_{a'\in\mathcal{A}} \pi(a'|s') \nabla_\theta Q_{\pi,\theta}(s',a') \\
&= \left[\sum_{s'\in\mathcal{S}} \nabla_\theta \tilde{p}_\theta(s'|s,a) \right] \left[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi,\theta}(s',a') \right] \\
&+ \gamma \sum_{s',a'} \Pr\left[(s,a) \to (s',a'), k|\theta,\pi\right] \nabla_\theta Q_{\pi,\theta}(s',a')
\end{aligned} \tag{A.10}$$

where $\Pr[(s,a) \to (s',a'), k|\theta,\pi]$ denotes the probability that agent utilized k steps to transfer from state-action pair (s,a) to (s',a').
For the simplicity, we denote $\sum_{s'\in\mathcal{S}} [\nabla_\theta f_\theta(s'|s,a)] \left[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi,\theta}(s',a')\right]$ as $\phi(s,a)$. Then:

$$\begin{aligned}
&\nabla_\theta Q_{\pi,\theta}(s,a) \\
&= \phi(s,a) + \gamma \sum_{s',a'} \Pr\left[(s,a) \to (s',a'), 1|\theta,\pi\right] \nabla_\theta Q_{\pi,\theta}(s',a') \\
&= \phi(s,a) + \gamma \sum_{s',a'} \Pr\left[(s,a) \to (s',a'), 1|\theta,\pi\right] \left[\phi(s',a') + \gamma \sum_{s'',a''} \Pr\left[(s',a') \to (s'',a''), 1|\theta,\pi\right] \nabla_\theta Q_{\pi,\theta}(s'',a'') \right] \\
&= \phi(s,a) + \gamma \sum_{s',a'} \Pr\left[(s,a) \to (s',a'), 1|\theta,\pi\right] \phi(s',a') + \gamma^2 \sum_{s'',a''} \Pr\left[(s',a') \to (s'',a''), 2|\theta,\pi\right] \nabla_\theta Q_{\pi,\theta}(s'',a'') \\
&= \dots \\
&= \sum_{s'\in\mathcal{S},a'\in\mathcal{A}} \sum_{k=0}^{\infty} \gamma^k \Pr((s,a) \to (s',a'), k|\pi,\theta) \phi(s',a')
\end{aligned} \tag{A.11}$$

We denote $\eta(s,a) = \mathbb{E}_{s_0,a_0}\left[\sum_{k=0}^{\infty}\gamma^k \Pr((s_0,a_0)\to(s,a),k|\theta,\pi)\right]$, and come back to $\nabla_\theta \mathbb{E}_{s_0,a_0}[Q_{\pi,\theta}(s,a)]$:

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_{s_0,a_0}[Q_{\pi,\theta}(s_0,a_0)] \\
&= \sum_{s\in\mathcal{S},a\in\mathcal{A}} \mathbb{E}_{s_0,a_0}\left[\sum_{k=0}^{\infty}\gamma^k \Pr((s_0,a_0)\to(s,a),k|\theta,\pi)\right]\phi(s,a) \\
&= \sum_{s\in\mathcal{S},a\in\mathcal{A}} \eta(s,a)\phi(s,a) \\
&= \left(\sum_{s\in\mathcal{S},a\in\mathcal{A}} \eta(s,a)\right)\sum_{s\in\mathcal{S},a\in\mathcal{A}} \frac{\eta(s,a)}{\sum_{s\in\mathcal{S},a\in\mathcal{A}}\eta(s,a)}\phi(s,a) \\
&\propto \sum_{s\in\mathcal{S},a\in\mathcal{A}} \frac{\eta(s,a)}{\sum_{s\in\mathcal{S},a\in\mathcal{A}}\eta(s,a)}\phi(s,a) \qquad \text{(A.12)} \\
&= \sum_{s\in\mathcal{S},a\in\mathcal{A}} d^{\theta,\pi}(s,a)\sum_{s'\in\mathcal{S}}[\nabla_\theta \tilde{p}_\theta(s'|s,a)]\left[R(s,a,s')+\gamma\sum_{a'\in\mathcal{A}}\pi(a'|s')Q_{\pi,\theta}(s',a')\right] \\
&= \mathbb{E}_{\substack{s\sim d^{\theta,\pi}(s)\\ a\sim\pi(\cdot|s)}}\left[\sum_{s'\in\mathcal{S}}\left[R(s,a,s')+\gamma\sum_{a'\in\mathcal{A}}\pi(a'|s')Q_{\pi,\theta}(s',a')\right]\nabla_\theta\tilde{p}_\theta(s'|s,a)\right] \\
&= \mathbb{E}_{\substack{s\sim d^{\theta,\pi}(s)\\ a\sim\pi(\cdot|s)\\ s'\sim\tilde{p}_\theta(\cdot|s,a)}}\left[\left[R(s,a,s')+\gamma\sum_{a'\in\mathcal{A}}\pi(a'|s')Q_{\pi,\theta}(s',a')\right]\nabla_\theta\log\tilde{p}_\theta(s'|s,a)\right]
\end{aligned}$$

We can find that there is no restriction of $\tilde{p}_\theta$ which means it could be vastly different from the original transition p, one of the solutio is add a regularization using KL divergence. With a fixed π, and the perturbed transition function is parameterized by θ, then the regularization is:

$$R(\theta) = \mathbb{E}_{s\sim d^{\pi,p}(s),a\sim\pi(\cdot|s)}[D_{KL}(p(\cdot|s,a)\|\tilde{p}_\theta(\cdot|s,a))] \qquad \text{(A.13)}$$

We can easily compute the gradient of it:

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_{s\sim d^{\pi,\theta}(s),a\sim\pi(\cdot|s)}[D_{KL}(f_\theta(\cdot|s,a)\|p(\cdot|s,a))] \\
&= \nabla_\theta \sum_{s\in\mathcal{S}} d^{\pi,\theta}(s)\sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\tilde{p}_\theta(s'|s,a)\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)} \\
&= \sum_{s\in\mathcal{S}} d^{\pi,\theta}(s)\sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\left\{\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}\nabla_\theta\tilde{p}_\theta(s'|s,a)+\tilde{p}_\theta(s'|s,a)\nabla_\theta\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}\right\} \\
&= \sum_{s\in\mathcal{S}} d^{\pi,\theta}(s)\sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\left\{\left[\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}+1\right]\nabla_\theta\tilde{p}_\theta(s'|s,a)\right\} \qquad \text{(A.14)} \\
&= \sum_{s\in\mathcal{S}} d^{\pi,\theta}(s)\sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}\tilde{p}_\theta(s'|s,a)\left\{\left[\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}+1\right]\nabla_\theta\log\tilde{p}_\theta(s'|s,a)\right\} \\
&= \mathbb{E}_{\substack{s\sim d^{\pi,\theta}(s),a\sim\pi(\cdot|s)\\ s'\sim\tilde{p}_\theta(\cdot|s,a)}}\left[\left\{\log\frac{\tilde{p}_\theta(s'|s,a)}{p(s'|s,a)}+1\right\}\nabla_\theta\log\tilde{p}_\theta(s'|s,a)\right]
\end{aligned}$$

■

Theorem 5 Proof
First we denote $P_\pi(s'|s) = \sum_{a\in\mathcal{A}}\pi(a|s)p(s'|s,a)$ and $r_p(s) = \sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}p(s'|s,a)R(s,a,s')$. Let V_π denote the $|\mathcal{S}|$-dimension vector with $(s)th$ component $V_\pi(s)$, r_p denote the $|\mathcal{S}|$-dimension vector with $(s)th$ component $r_p(s)$ where p denote the transition probability, and P_π denote the $|\mathcal{S}|\times|\mathcal{S}|$ matrix with $(s,s')th$ entry given by $P_\pi(s',|s)$. We refer to P_π as the transition probability for state corresponding to the policy π, and refer to r_p as the reward vector corresponding to transition

probability p.
With a discount factor $0 < \gamma < 1$,

$$\begin{aligned}
V_\pi &= [V_\pi(s)|s \in \mathcal{S}] \\
&= [\sum_{a\in A} \pi(a|s) \sum_{s'\in S} p(s'|s,a)[R(s,a,s') + \gamma V_\pi(s')]|s \in \mathcal{S}] \\
&= [\sum_{a\in A} \pi(a|s) \sum_{s'\in S} p(s'|s,a)R(s,a,s') + \gamma \sum_{a\in A} \pi(a|s) \sum_{s'\in S} \tilde{p}(s'|s,a)V_\pi(s')|s \in \mathcal{S}] \\
&= [r_p(s) + \gamma \sum_{a\in A} \sum_{s'\in S} \pi(a|s)p(s'|s,a)V_\pi(s')|s \in \mathcal{S}]
\end{aligned}$$

can be represented by:

$$V_\pi = r_p + \gamma P_\pi V_\pi \tag{A.15}$$

Hence, we have:

$$V_\pi = (I - \gamma P_\pi)^{-1} r_p \tag{A.16}$$

Similarly, we can rewrite $\tilde{V}_{\pi,\tilde{p}^*(\pi)}(s,a)$:

$$\tilde{V}_{\pi,\tilde{p}^*(\pi)} = (I - \gamma P_{\pi,\tilde{p}^*(\pi)})^{-1} r_{\tilde{p}^*(\pi)} \tag{A.17}$$

Where $P_{\pi,\tilde{p}^*(\pi)}(s'|s) = \pi(a|s)\tilde{p}^*(s'|s,a)$. Before we proceed to the next step, let's introduce a new $|\mathcal{S}|$-dimension vector V:

$$V = (I - \gamma P_{\pi,\tilde{p}^*(\pi)})^{-1} r_p \tag{A.18}$$

For $V_\pi - V$ we have:

$$\begin{aligned}
\|V_\pi - V\|_\infty &= \|(I - \gamma P_\pi)^{-1} r_p - (I - \gamma P_{\pi,\tilde{p}^*(\pi)})^{-1} r_p\|_\infty \\
&\le \|(I - \gamma P_\pi)^{-1} - (I - \gamma P_{\pi,\tilde{p}^*(\pi)})^{-1}\|_\infty \|r_p\|_\infty \\
&\le \|(I - \gamma P_\pi)^{-1}\|_\infty \|(I - \gamma P_{\pi,\tilde{p}^*(\pi)})^{-1}\|_\infty \gamma \|P_\pi - P_{\pi,\tilde{p}^*(\pi)}\|_\infty \|r_p\|_\infty \\
&= \frac{\gamma \|r_p\|_\infty}{(1-\gamma)^2} \|P_\pi - P_{\pi,\tilde{p}^*(\pi)}\|_\infty
\end{aligned} \tag{A.19}$$

The second inequality comes from the fact that:

$$\|X^{-1} - Y^{-1}\| \le \|X^{-1}\| \|Y^{-1}\| \|X - Y\|$$

And the last equation holds because:

$$\|(I - \gamma P_\pi)^{-1}\|_\infty = \|\sum_{n=0}^{\infty} (\gamma P_\pi)^n\|_\infty \le \sum_{n=0}^{\infty} (\gamma)^n = \frac{1}{1-\gamma} \tag{A.20}$$

We note that:

$$\frac{\gamma \|r_p\|_\infty}{(1-\gamma)^2} = \frac{\gamma \max_{s,a} r_p(s,a)}{(1-\gamma)^2} \le \frac{\gamma \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \tag{A.21}$$

It's a constant that does not depend on p. Then equation (A.19) can be rewritten as:

$$\begin{aligned}
\max_s |V_\pi(s) - V(s)| &\leq \frac{\gamma \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \|P_\pi(\cdot|s) - P_{\pi,\tilde{p}^*(\pi)}(\cdot|s)\|_1 \\
&= \frac{\gamma \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{s'\in\mathcal{S}} |P_\pi(s'|s) - P_{\pi,\tilde{p}^*(\pi)}(s'|s)| \\
&= \frac{\gamma \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{s'\in\mathcal{S}} |\sum_{a\in\mathcal{A}} [\pi(a|s)p(s'|s,a) - \pi(a|s)\tilde{p}^*(s'|s,a)]| \\
&\leq \frac{\gamma \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} |p(s'|s,a) - \tilde{p}^*(s'|s,a)| \\
&= \frac{2\gamma \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{a\in\mathcal{A}} \pi(a|s) D_{TV}(p(\cdot|s,a), \tilde{p}^*(\cdot|s,a)) \\
&\leq \frac{2\gamma \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{a\in\mathcal{A}} \pi(a|s) \max_{\tilde{p}\in\mathcal{B}(s,a)} D_{TV}(p(\cdot|s,a), \tilde{p}(\cdot|s,a))
\end{aligned} \tag{A.22}$$

For $V - \tilde{V}_{\pi,\tilde{p}^*(\pi)}$, we have

$$\|V - \tilde{V}_{\pi,\tilde{p}^*(\pi)}\|_\infty = \|(I - \gamma P_{\pi,\tilde{p}^*})^{-1}\|_\infty \|r_p - r_{\tilde{p}^*(\pi)}\|_\infty \tag{A.23}$$

According to equation (A.20) and the definition of infinite norm, equation (A.23) can be rewritten as:

$$\begin{aligned}
\max_s |V(s) - \tilde{V}_{\pi,\tilde{p}^*(\pi)}(s)| &= \frac{1}{1-\gamma} \max_s |r_p(s) - r_{\tilde{p}^*(\pi)}(s)| \\
&\leq \frac{1}{1-\gamma} \max_{s,a,s'} R(s,a,s') \max_s \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} |p(s'|s,a) - \tilde{p}^*(s'|s,a)| \\
&= \frac{2}{1-\gamma} \max_{s,a,s'} R(s,a,s') \max_s \sum_{a\in\mathcal{A}} \pi(a|s) D_{TV}(p(\cdot|s,a), \tilde{p}^*(\cdot|s,a)) \\
&\leq \frac{2}{1-\gamma} \max_{s,a,s'} R(s,a,s') \max_s \sum_{a\in\mathcal{A}} \pi(a|s) \max_{\tilde{p}(s,a)\in\mathcal{B}(s,a)} D_{TV}(p(\cdot|s,a), \tilde{p}(\cdot|s,a))
\end{aligned} \tag{A.24}$$

We can derive a new bound from equation (A.22) and equation (A.24) by Triangle Inequality:

$$\max_{s,a} |V_\pi(s,a) - \tilde{V}_{\pi,\tilde{p}^*(\pi)}(s,a)| \leq \frac{2 \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{a\in\mathcal{A}} \pi(a|s) \max_{\tilde{p}(s,a)\in\mathcal{B}(s,a)} D_{TV}(p(\cdot|s,a), \tilde{p}(\cdot|s,a)) \tag{A.25}$$

In TA-MDP we restrict $\tilde{p}(s,a)$ by $D_{KL}(p(\cdot|s,a)\|\tilde{p}(\cdot|s,a)) \leq \epsilon$, so we can also bound the loss of action value function under perturbation as:

$$\max_{s,a} |V_\pi(s,a) - \tilde{V}_{\pi,\tilde{p}^*(\pi)}(s,a)| \leq \frac{\sqrt{2} \max_{s,a,s'} R(s,a,s')}{(1-\gamma)^2} \max_s \sum_{a\in\mathcal{A}} \pi(a|s) \max_{\tilde{p}(s,a)\in\mathcal{B}(s,a)} D_{KL}(p(\cdot|s,a)\|\tilde{p}(\cdot|s,a)) \tag{A.26}$$

■