Robust Linear Reinforcement Learning

Sam Lobel  
Ruo Yu Tao  
Tuluhan Akbulut

Abstract

Reinforcement learning is the study of learning in sequential decision-making problems and has shown commercial success in simulated environments like games. However, their application is limited in real-world environments like robotics due to their uncertainty and complexity. Robust reinforcement learning aims to overcome this limitation by modeling the challenges of real-world environments with an adversary corrupting the data. In this project, we focus on learning in two different settings. First, we consider policy evaluation in arbitrary Markov Decision Processes (MDPs) where an adversary can select actions some $\epsilon$ fraction of the time. Next, we consider learning value functions from data in linear MDPs when some $\epsilon$ fraction of data is corrupted.

1 Introduction

Reinforcement learning (RL) aims to learn a policy by utilizing a trial and error mechanism in an environment where an agent gets feedback (reward) based on its actions Sutton and Barto\cite{Sutton2018}. Although RL got well-known for its success in game environments Mnih et al.\cite{Mnih2015}; Silver et al.\cite{Silver2016}, its usage in real-world environments has been limited Dulac-Arnold et al.\cite{Dulac-Arnold2019}.

Real-world environments have uncertainty, disturbances, and structural changes, and RL algorithms can not cope. Robust RL studies this problem in a min-max game setting where an adversarial agent models the uncertainty and disturbances Moos et al.\cite{Moos2022}. While policy learning algorithms are training, the adversary corrupts the data, so that the trained agent will be robust to such data shifts.

In this project, we study the problem of value estimation in such corrupted sequential decision-making problems. Value estimation concerns learning the value of following a policy from a set of states. We consider two corruption settings: characterizing the effect an $\epsilon$-adversarial policy can have on expected value, and characterizing the effect that adversarially replacing $\epsilon$ fraction of all training data, including states, rewards, and transitions, can have on value estimation in linear MDPs.

2 Background

We study sequential decision-making problems in the fully-observable, finite-horizon setting. We model our environment as a Markov Decision Process (MDP) described by $< S, A, R, T >$, where $S$ is the set of states and $A$ is the set of actions. The transition and reward functions are given by $T(s, a)$ and $R(s, a)$, respectively.

We focus on the finite horizon setting, where each episode of interaction lasts for exactly $H$ timesteps. A policy $\pi : (S \times A \times H) \rightarrow \mathbb{R}$ describes the probability of taking any given action from any given state, at a given timestep. We define the per-timestep policy $\pi_t(s)$ as a random variable that takes on the value of each action with some probability. A particular quantity of interest in reinforcement

Preprint. Under review.
We first consider the case where we corrupt a policy for the policy evaluation setting. In this setting, we highlight two ways to estimate a value function in reinforcement learning: using dynamic programming after being given the transition and reward functions, or estimating these quantities through data.

**Policy Corruption:** In this case, the adversary changes \( \epsilon \) fraction of policy’s actions. This \( \epsilon \) -contaminated policy is denoted by \( \hat{\pi}_t(s) \) where as adversary policy is denoted by \( \alpha_t(s) \)

\[
\hat{\pi}_t(s) = (1 - \epsilon)\pi_t(s) + \epsilon\alpha_t(s)
\]

**Data Corruption:** In this case, the adversary changes \( \epsilon \) fraction of data tuples \( D = \{(s_i, a_i, r_i, s'_i)\}_{i=1:N} \) , meaning for \( \epsilon N \) samples \( (s_i, a_i, r_i, s'_i) \neq (\tilde{s}_i, \tilde{a}_i, \tilde{r}_i, \tilde{s}'_i) \). \( \sim \) denotes the true data tuple.

The former is a subset of the latter. The probability of the tuple:

\[
P(s_i, a_i, r_i, s'_i) = P(s_i)\pi(a_i|s_i)T(s_i, a_i, r_i, s'_i)
\]

Assume the adversary changes the action, we can find a transition function that creates the same behavior by mapping the original actions probabilities to corrupt one:

\[
P(s_i, a_i, r_i, s'_i) = P(s_i)\pi(a_i|s_i)T(s_i, a_i, r_i, s'_i) = P(s_i)\pi(a_i|s_i)\hat{T}(s_i, a_i, r_i, s'_i)
\]

where \( \hat{T}(s_i, a_i, r_i, s'_i) = T(s_i, \tilde{a}_i, r_i, s'_i) \). The reverse is not possible, because, for an arbitrary next state and reward, there may not exist such an action. For the policy corruption, we will show the general bound. For the data corruption, we will make the linearity assumption.

### 3 Policy Corruption

We first consider the case where we corrupt a policy for the policy evaluation setting. In this setting, some policy is corrupted \( \epsilon \)-portion of the time steps, and the goal is to recover the ground-truth value function, \( V_\pi \), to as high accuracy as possible.

Policy corruption can happen in a myriad of settings for reinforcement learning. In robotics, untrustworthy controls due to environmental and/or actuation issues are important considerations. Even in deep reinforcement learning research, randomly repeating actions (or action repeats) in the Arcade Learning Environment (Machado et al., 2018) are widely used for testing reinforcement learning algorithms.

In this section, we consider initial bounds on the worst-case accuracy of policy evaluation. To do so, we first formalize this problem setting.

#### 3.1 Problem Setting

In the policy corruption problem setting. Our action \( a_i \) is sampled from \( \pi(\cdot | s_t) \), and our rewards and next states are sampled from \( T \) and \( R \) respectively. The corrupted policy \( \hat{\pi} \), which we also call the \( \epsilon \)-corrupted policy, is defined as:

\[
\hat{\pi}(\cdot | s) = \begin{cases} \pi(\cdot | s) & \text{w.p. } (1 - \epsilon) \\ \alpha(\cdot | s) & \text{o.w.} \end{cases}
\]
This is the policy where for \(1 - \epsilon\) fraction of the actions are sampled according to the regular policy \(\pi\), whereas some other policy, \(\alpha\), gets sampled the remaining time. With this definition of policy corruption, the policy evaluation error, or the measure in which the value function \(V^\pi\) is inaccurate with respect to \(V^\pi\) is:

\[
|V^\pi_0(s_0) - V^\hat{\pi}_0(s_0)|
\]

where \(s_0\) is the initial state of the MDP. Lastly, for this problem setting we consider the case where \(\forall s, a \in S, A, R(s, a) \in [0, 1]\).

Given this problem setting, we now consider how to derive robust methods and how to form an adversary.

### 3.2 Adversarial Policy

With this form of policy corruption, we consider an adversary on the policy \(\alpha\). This allows the adversary to re-assign \(\epsilon\) portion of the probability mass of the policy at every state and time step, \(\pi(\cdot | s_t)\). The objective function of the adversary for this error function is:

\[
\max_{\alpha} |V^\pi_0(s_0) - V^\hat{\pi}_0(s_0)|
\]

where \(\hat{\pi}\) is the corrupted policy defined in Equation 1. This amounts to an adversarial setting where the adversary has access to the transition function \(T\), the reward function \(R\), and the ground-truth policy \(\pi\). The adversary is allowed to choose an adversarial policy \(\alpha\), where this policy is applied \(\epsilon\) fraction of the time at every time step, resulting in the policy \(\hat{\pi}\). The algorithm designer then receives samples from this policy \(\hat{\pi}\) and designs an algorithm to try and approximate \(V^\pi\).

Due to the temporal nature of value functions, this adversarial objective is not straightforward to optimize, even with access to the underlying MDP dynamics. By the definition of \(\alpha\) from Equation 1, the adversarial policy is dependent on state. This means that at every step, given a state, an adversary has to decide how to change \(\epsilon\) of the probability mass to maximize the objective function as per Equation 3. Considering the definition of value functions as the sum of future discounted rewards, the adversarial policy \(\alpha\) at a given state \(s_t\) has to try and maximize this error, which is an error with respect to entire trajectories. This means that the to maximize this objective function, the adversarial policy at time step \(t\), \(\alpha(\cdot | s_t)\), has to try and maximize error with respect to all future expected returns—an adversarial policy cannot only consider maximizing the difference between the next reward \(R(s_t, a_t)\), but must also consider transitioning to a next state \(s_{t+1}\) that may result in larger future trajectory return differences.

With our adversarial objective defined, we now consider an initial bound on Equation 3 given some \(\epsilon\), what is the worst-case this error could be?

### 3.3 Policy Corruption Lemma

We now present a lemma on the upper bound of this adversarial objective, given our problem setting and corruption model.

**Lemma 3.1.** Given a target policy \(\pi\) and its \(\epsilon\)-corrupted policy \(\hat{\pi}\), the value function error between the two policies is bounded by:

\[
|V^\pi_0(s_0) - V^\hat{\pi}_0(s_0)| \leq H + 1 - \frac{(1 - \epsilon)^{H+1}}{\epsilon}
\]

where \(H\) is the horizon of the finite-horizon MDP, and \(s_0\) is any initial state of the MDP.

**Proof.** We show this bound with a proof by induction on the time step \(t\).
We begin with the base case, when $t = H$ and the value function is defined by only the immediate rewards given the policies $\pi$ and $\hat{\pi}$:

$$|V^\pi_H(s) - V^\hat{\pi}_H(s)| = \left| \sum_{a \in A} \pi(a | s) R(s, a) - \sum_{a \in A} \hat{\pi}(a | s) R(s, a) \right|$$

$$= \sum_{a \in A} R(s, a)[\pi(a | s) - \hat{\pi}(a | s)]$$

$$\leq \sum_{a \in A} [\pi(a | s) - \hat{\pi}(a | s)] \quad \text{(Since $R(s, a) \in [0, 1]$)}$$

$$\leq \epsilon \quad \text{(At most $\epsilon$ prob. changes)}.$$  

For any time step $t$, we rely on the fact that at any step, the worst the adversary policy $\alpha$ could do is take an action at some given state to a part of the MDP that has maximal difference in future returns for the remaining $H - t$ steps. This happens with probability $\epsilon$ at every step, and with probability $1 - \epsilon$ we transition to the next step, where a similar process occurs, but for the remaining $H - t - 1$ steps:

$$|V^\pi_t(s) - V^\hat{\pi}_t(s)| \leq 1 + \epsilon(H - t) + (1 - \epsilon)\epsilon(H - t - 1) + \cdots + (1 - \epsilon)^{H-t-1}\epsilon$$

$$= 1 + \epsilon \sum_{i=0}^{H-t-1} (1 - \epsilon)^i (H - t - i)$$

$$= 1 + \epsilon \left( \frac{(1 - \epsilon)^{H-t-1} - (1 - \epsilon)(H - t + 1) + H - t}{\epsilon^2} \right) \quad \text{(By Lemma 5.1)}$$

$$= 1 + \frac{(1 - \epsilon)^{H-t+1} - (1 - \epsilon)(H - t + 1) + H - t}{\epsilon}. \quad \text{(7)}$$

Now set $t = 0$ to get the desired bound:

$$|V^\pi_0(s_0) - V^\hat{\pi}_0(s_0)| \leq 1 + \frac{(1 - \epsilon)^{H+1} - (1 - \epsilon)(H + 1) + H}{\epsilon}$$

$$= H + 1 - \frac{(1 - \epsilon)^{H+1}}{\epsilon}. \quad \text{(8)}$$

This expression is a familiar expression in reinforcement learning. The worst-case scenario can be visualized with a 2-state MDP where the agent tries to stay at the first state, and it gets a $+1$ reward every time it succeeds. The adversary tries to direct it to the second state, where it can not escape until the end of the time horizon and gets $0$ rewards for all time steps. In this setting, if we start from the last time step and roll back:

$$V^\pi_H = 1$$

$$V^\hat{\pi}_{H-1} = (1 - \epsilon)V^\hat{\pi}_H + 1$$

$$V^\hat{\pi}_{H-2} = (1 - \epsilon)V^\hat{\pi}_{H-1} + 1$$

$$V^\pi_0 = (1 - \epsilon)^H + (1 - \epsilon)^{H-1} + \cdots + 1 = \frac{1 - (1 - \epsilon)^{H+1}}{\epsilon}$$

Since the agent will get an $H+1$ reward without an adversary, the bound becomes the same as above. That also suggests that the epsilon policy contamination behaves similarly as a discount factor in the worst case and with the assumption $R(s, a) \in [0, 1]$.

In the same settings, we can also derive the bound for the infinite horizon case where we need to introduce a discount factor $\gamma$ so that the values converge without an adversary.
With the adversary:

\[
V_0^\pi = \frac{1}{1 - \gamma} + \gamma \frac{1}{1 - \gamma} + \ldots
\]

After calculating all values of \( \epsilon \)

To create an adversary for this problem setting, as mentioned in Section 3.2, our adversarial policy must take into account values of future states and rewards. One way of creating an adversarial policy in the finite-horizon case is to leverage a dynamic programming approach with \( H \) and \( R \). Starting at the horizon \( t = H \), the adversarial policy for every state \( s_H \) would be the policy that maximizes the objective (final reward) at time step \( H \):

\[
\max_{a_H} R(a_H, s_H) |\pi(a_H | s_H) - \alpha(a_H | s_H)|
\]

After calculating all values of \( V_0^\pi \) for this final state, we can continue "rolling back" our adversarial policy by first finding \( \alpha \) for every state at time step \( t, s_t \), based on the objective:

\[
\max_{a_t} \left[ R(a_t, s_t) + \sum_{a \in A} \hat{\pi}(a | s_t) \sum_{s_{t+1} \in S} T(s_{t+1} | s_t, a) \hat{V}_H^\pi(s_{t+1}) |\pi(a_t | s_t) - \alpha(a_H | s_H)\right],
\]

where \( \hat{\pi}(a | s_t) \) is defined as the \( \epsilon \)-corrupted policy when \( \alpha(a_t | s_t) = 1 \). After finding \( \alpha(a_t | s_t) \), the adversary then calculates \( \hat{V}_t^\pi \). We continue this dynamic programming approach until we reach \( t = 0 \). We defer empirically testing this approach to future work.

### 3.4 Creating Adversaries For Policy Corruption

To create an adversary for this problem setting, as mentioned in Section 3.2, our adversarial policy must take into account values of future states and rewards. One way of creating an adversarial policy in the finite-horizon case is to leverage a dynamic programming approach with \( T \) and \( R \). Starting at the horizon \( t = H \), the adversarial policy for every state \( s_H \) would be the policy that maximizes the objective (final reward) at time step \( H \):

\[
\max_{a_H} R(a_H, s_H) |\pi(a_H | s_H) - \alpha(a_H | s_H)|
\]

After calculating all values of \( V_0^\pi \) for this final state, we can continue "rolling back" our adversarial policy by first finding \( \alpha \) for every state at time step \( t, s_t \), based on the objective:

\[
\max_{a_t} \left[ R(a_t, s_t) + \sum_{a \in A} \hat{\pi}(a | s_t) \sum_{s_{t+1} \in S} T(s_{t+1} | s_t, a) \hat{V}_H^\pi(s_{t+1}) |\pi(a_t | s_t) - \alpha(a_H | s_H)\right],
\]

where \( \hat{\pi}(a | s_t) \) is defined as the \( \epsilon \)-corrupted policy when \( \alpha(a_t | s_t) = 1 \). After finding \( \alpha(a_t | s_t) \), the adversary then calculates \( \hat{V}_t^\pi \). We continue this dynamic programming approach until we reach \( t = 0 \). We defer empirically testing this approach to future work.

### 3.5 3-step MDP example

Consider the MDP in Figure 1, where nodes and edges correspond to states and actions, respectively. The MDP is deterministic, i.e. when the agent selects an edge, it will end up in the state that the edge points to if there is no adversary. The rewards are shown on the edges. The agent starts in \( s_k \). After solving this MDP without adversary:

\[
V_{1,2,3}^\pi(s_O) = 0 \text{ (The agent cannot escape O and there is no rewarding action.)}
\]

\[
V_3^\pi(s_P) = 0 \text{ (P is end state for } H = 3)\]

\[
V_{2}^\pi(s_N) = 1(1 + V_3^\pi(s_P)) + 0(0 + V_3^\pi(s_O)) = 1 \text{ (} \pi^* \text{ always selects the more rewarding action.)}
\]

\[
V_{1}^\pi(s_L) = 1(0 + V_2^\pi(s_N)) + 0(0 + V_2^\pi(s_O)) = 1 \text{ (} \pi^* \text{ always selects the more rewarding action.)}
\]

\[
V_{0}^\pi(s_K) = 1(0 + V_1^\pi(s_L)) + 0(0.3 + V_1^\pi(s_K)) + 0(0 + V_1^\pi(s_O)) = 1 \text{ (} \pi^* \text{ always selects the more rewarding action.)}
\]

\[
V_{1}^\pi(s_K) = 0.6, \ V_2^\pi(s_K) = 0.3, \ V_3^\pi(s_L) = 0 \text{ (Calculated similarly with Bellmann equation)}
\]

For \( 1 \leq t \), staying at \( K \) is the best. Otherwise, there are not enough time steps to reach \( P \).

Then, we include an adversary with the rate \( \epsilon = 0.2 \), the adversary tries to maximize the value difference for each state, as explained previously. For the final actionable state \( N \), it would choose the edge that points to \( O \). Then the new value function for state \( N \) becomes:

\[
V_2^\pi(s_N) = 0.8(1 + V_3^\pi(s_P)) + 0.2(0 + V_3^\pi(s_O)) = 0.8
\]
For this state, the best action does not change. The agent will try to go to P with the remaining 0.8 probability. Similarly,

\[ V_1^\pi(s_L) = 0.8(0 + V_2^\pi(s_N)) + 0.2(0 + V_2^\pi(s_O)) = 0.64 \]
\[ V_2^\pi(s_K) = 0.8(0.3 + 0) + 0.2 \times 0 = 0.24 \]
\[ V_1^\pi(s_K) = 0.8(0.3 + V_2^\pi(s_K)) + 0.2 \times 0 = 0.432 \]

For the first state, the best action changes because:

\[ V_1^\pi(s_K) + 0.3 = 0.732 \geq 0.64 = V_1^\pi(s_L) + 0 \]

The adversary will try to direct the agent again toward state O. Therefore, the final value function becomes:

\[ V_0^\pi(s_K) = 0(0 + V_1^\pi(s_L)) + 0.8(0.3 + V_1^\pi(s_K)) + 0.2(0 + V_1^\pi(s_O)) = 0.5856 \]

The value difference w.r.t. the bound:

\[ |V_0^\pi(s_0) - V_0^\pi(s_0)| = 0.4144 \leq H + 1 - \frac{(1 - \epsilon)^H + 1}{\epsilon} = 1.048 \]

4 Data Corruption

We now consider the problem of estimating the value function of a policy from data in the setting of Linear MDPs. Whereas before, the transition and reward functions \( T \) and \( R \) were provided to the algorithm, now in order to use the quantities for value estimation, they must be learned from samples taken from the MDP. Whereas most applications of RL are done in simulation where sensors are assumed perfect, using these tools on real problems requires robustness to misspecifications of agent state and transition. Thus, robust methods can be helpful in hardening learning algorithms for real-world problems.

4.1 Problem setting

Recall that we can recursively define the value at a given timestep like so:

\[ V_t^\pi(s) = \sum_a \left( \pi(s, a) R(s, a) + \sum_{s'} T(s', s, a) V_{t+1}^\pi(s') \right) \]

By learning accurate estimates \( \hat{R} \) and \( \hat{T} \), we can learn an approximation of \( V_t^\pi \):

\[ \hat{V}_t^\pi(s) = \sum_a \left( \pi(s, a) \hat{R}(s, a) + \sum_{s'} \hat{T}(s', s, a) V_{t+1}^\pi(s') \right) \]

We now define our data and corruption model. Here we consider d-dimensional continuous-state MDPs, where \( s \in \mathbb{R}^d \). Let \( \mathcal{X} \) be a probability distribution over \( \mathbb{R}^d \) from which \( n \) states \( \{s_i : i \in [n]\} \) are sampled. Tuples of states, actions, rewards, and next-states are created from these samples using the policy, transition, and reward functions:
\[ \mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1:N} \]

The corruption model is then defined as so: an adversary is allowed to inspect \( \mathcal{D} \) and arbitrarily change some fraction \( \epsilon \) of this data. The algorithm’s task is then to learn an accurate estimate of \( \hat{V} \) from the corrupted data.

In order for this problem to be tractable, we must make some assumptions on the state distribution \( \mathcal{X} \), transition function \( \mathcal{T} \), and reward function \( \mathcal{R} \). For simplicity, let the queried states come from a normal distribution over the state space: \( s \sim \mathcal{N}(0, I) \). Furthermore, let the state-state transition induced by the policy, \( \mathcal{T}_\pi \) be an (unknown) linear transformation, plus unit Gaussian noise: \( s' \sim \mathcal{N}(\mathcal{T}_\pi s, I) \). Additionally, let the reward function also be an (unknown) linear transformation plus unit Gaussian noise: \( r \sim \mathcal{N}(\mathcal{R}_\pi \cdot s, 1) \). Finally, we assume that the \( L_2 \) norm of the transition function is larger than 1, \( \| \mathcal{T}_\pi \|_2 \geq 1 \), to present more meaningful and unified bounds.

### 4.2 Main Result

We can now state our main result for this section:

1. Given the above assumptions on \( \mathcal{T} \) and \( \mathcal{R} \), \( V^0_\pi(s) \) is a linear function of state: \( V^0_\pi(s) = V_0 \cdot s \).
2. Furthermore, given \( \epsilon \)-corrupted data as described above, we can learn an approximation \( \hat{V}_0 \) such that \( \epsilon \) such that
   \[
   \| V_0 - \hat{V}_0 \|_2 = O(H^2(\| \mathcal{T}_\pi \|_2^H + \epsilon H d_{H/2})(\| R \|_2 Ver\mathcal{E}(\| \mathcal{T}_\pi \|_2^H + \epsilon))
   \] (9)

### 4.3 Proof of claim 1

The intuition behind the claim is that since \( \mathcal{T}_\pi \) and \( \mathcal{R} \) are linear with symmetric noise, the noise integrates out and we are left with just the linear mapping. We do this by breaking the expectation into a constant term, and the symmetric noise component, which integrates to zero. We prove this claim using induction. Our base case is the final timestep’s value function \( V^H_H \):

\[
V^H_H(s) = \mathbb{E}[R(s)] = \mathbb{E}[R \cdot s + \mathcal{N}(0, 1)] = \mathbb{E}[R \cdot s] + \mathbb{E}[\mathcal{N}(0, 1)] = R \cdot s
\]

Now, assuming that \( V_{t+1}^\pi (s) = V_{t+1} \cdot s \):

\[
V_{t+1}^\pi (s) = \mathbb{E}[R(s)] + \mathbb{E}[V_{t+1}^\pi (T_\pi(s))]
\]

\[
= R \cdot s + \mathbb{E}[V_{t+1} \cdot T_\pi(s)]
\]

\[
= R \cdot s + \mathbb{E}[V_{t+1} \cdot (T_\pi \cdot s + \mathcal{N}(0, 1))]
\]

\[
= R \cdot s + \mathbb{E}[V_{t+1} \cdot T_\pi \cdot s] + \mathbb{E}[V_{t+1} \cdot \mathcal{N}(0, 1)]
\]

\[
= R \cdot s + V_{t+1} \cdot T_\pi \cdot s + 0
\]

\[
= R \cdot s + V_{t+1} \cdot T_\pi \cdot s
\]

Since the sum of two linear functions is linear, this completes our proof.

### 4.4 Proof of Claim 2

Now that we have proved that \( \mathcal{T}_\pi \) and \( \mathcal{R} \) as described imply linear \( V^\pi_t \), we can represent any \( V^\pi_t \) as a vector \( V_t \). We seek to learn a vector \( \hat{V}_0 \) from the corrupted data that describes the value function at the begin of an episode, i.e. \( \| V_0 - \hat{V}_0 \|_2 \) is small. Proving an accuracy bound on learning this vector has two components. The first component is deriving a relationship between \( \| V_0 - \hat{V}_0 \|_2 \) and the following terms: \( \| \mathcal{T}_\pi - \mathcal{T}_\pi \|_2 \), \( \| \mathcal{R} - \hat{\mathcal{R}} \|_2 \), \( \| \mathcal{T}_\pi \|_2 \) and \( \| \mathcal{R} \|_2 \). The second component is bounding terms \( \| \mathcal{T}_\pi - \mathcal{T}_\pi \|_2 \) and \( \| \mathcal{R} - \hat{\mathcal{R}} \|_2 \) with respect to the corruption fraction \( \epsilon \) and state-space dimension \( d \).
4.4.1 Proof of claim 2.1

We now relate \( \|V_0 - \hat{V}_0\|_2 \) to \( \|T_\pi - \hat{T}_\pi\|_2, \|R - \hat{R}\|_2, \|T_\pi\|_2 \) and \( \|R\|_2 \). We do this using a modified version of the Simulation Lemma (Kearns and Singh, 2002). The crucial difference between our proof and the simulation lemma is that our induced transition matrix \( T_\pi \) in expectation can increase the magnitude of a state from one timestep to the next. Since reward is linear in state, this may result in an exponential dependence of \( \hat{V} \) on \( H \). This is why we consider \( \|T_\pi\|_2 \geq 1 \), which we view as the more interesting case. In contrast, the transition matrices described in the simulation lemma preserve the magnitude of \( s \) and therefore do not have this exponential dependence.

Let \( \|R - \hat{R}\|_2 \leq \Delta_1 \) and \( \|T_\pi - \hat{T}_\pi\|_2 \leq \Delta_2 \). Then,

\[
\|V_0 - \hat{V}_0\|_2 = O(H^2(\|T_\pi\|_2^H + \Delta_2^H)(\|R\|_2 \|T_\pi\|_2^H + \Delta_1))
\]  
(10)

We prove this fact in the Appendix.

4.4.2 Proof of claim 2.2

We now relate \( \Delta_1 \) and \( \Delta_2 \) to the corruption fraction \( \epsilon \) and the state-dimension \( d \). Since all of our quantities are linear plus Gaussian noise, this reduces to using robust methods for linear regression to estimate \( \hat{R} \) and \( \hat{T}_\pi \). In particular, we use the method described in (Diakonikolas et al., 2019), which states the following:

Given a learning problem of the form \( y = x \cdot \beta + \mathcal{N}(0, 1) \), with high probability one can calculate an estimate of the weight vector, \( \hat{\beta} \) such that \( \|\hat{\beta} - \beta\|_2 \leq O(\epsilon) \) (excluding log factors), where \( \epsilon \) is the fraction of corrupted data.

This directly applies to estimating the vector \( \hat{R} \):

\[
\Delta \geq \|R - \hat{R}\|_2 = O(\epsilon).
\]  
(11)

Estimating \( \hat{T} \) is slightly more complicated, since \( T \) is a matrix. However, we can use the described method to estimate each row of the matrix: \( \|T_{\pi i} - \hat{T}_{\pi i}\|_2 = O(\epsilon), \) and can combine all \( d \) such rows for a slightly looser bound:

\[
\|T_\pi - \hat{T}_\pi\|_2 = O(\epsilon \sqrt{d}).
\]  
(12)

We can now combine Equation 10 with equation 11 and equation 12 to get our final result in Equation 9, restated here:

\[
\|V_0 - \hat{V}_0\|_2 = O(H^2(\|T_\pi\|_2^H + \epsilon^H \|T_\pi\|_2^H + \Delta_1))
\]

4.5 Conclusion and Future Work

This project highlights three interesting mathematical directions of improvement. First, we note that the policy proof yields a far tighter bound than the simulation lemma: indeed, the Simulation Lemma’s bound becomes vacuous for \( \epsilon > \frac{H}{\sqrt{H}} \). It would be interesting to dig into why this is, and possibly find a tighter bound for the Simulation Lemma. Second, it is unclear whether the \( \sqrt{d} \) dependence in estimating \( T \) is truly necessary. Overall, we present preliminary work in understanding the effect of an adversary interfering at different points in the RL training process.
References


5 Appendix

5.1 Lemma for Sum of Decreasing Series

Lemma 5.1.

\[ \sum_{i=0}^{n-1} c^i (n - i) = \frac{c^{n+1} - c(n + 1) + n}{(1 - c)^2} \]  

(13)

Proof.

\[ \sum_{i=0}^{n-1} c^i (n - i) = \sum_{i=0}^{n-1} c^i n - c^i i \]

\[ = n \left( \frac{1 - c^n}{1 - c} \right) - c \frac{\partial}{\partial c} \left( \sum_{i=1}^{n-1} c^i \right) \]

\[ = n \left( \frac{1 - c^n}{1 - c} \right) - c \frac{\partial}{\partial c} \left( \frac{1 - c^n}{1 - c} \right) \]

\[ = n \left( \frac{1 - c^n}{1 - c} \right) - c \left( -n c^{n-1} (1 - c) + (1 - c^n) \right) \]

\[ = n(1 - c^n)(1 - c) + n c^n(1 - c) - c + c^{n+1} \]

\[ = \frac{c^{n+1} - c(n + 1) + n}{(1 - c)^2} \]

\[ \square \]

5.2 Full proof of Claim 2.1

The first step of proving equation 10 is to upper bound \( \|V_t\|_2 \). Noting that \( V_H = R \) and \( V_t = R + V_{t+1} T_\pi \), we can use induction to show that \( V_t = \sum_{i=0}^{H-t} R T_\pi^i \), and therefore

\[ \|V_t\|_2 \leq \sum_{i=0}^{H-t} \|R\|_2 \|T_\pi\|_2^i \leq H \|R\|_2 \|T_\pi\|_2^H. \]  

(14)

Now, we borrow the proof technique from the Simulation Lemma to recursively upper bound a value-difference. recalling the definitions \( \|R - \hat{R}\|_2 \leq \Delta_1 \) and \( \|T_\pi - \hat{T}_\pi\|_2 \leq \Delta_2 \):

\[ V_t - \hat{V}_t = R - \hat{R} + V_{t+1} T_\pi - \hat{V}_{t+1} \hat{T}_\pi \]

\[ = R - \hat{R} + V_{t+1} T_\pi - V_{t+1} \hat{T}_\pi + V_{t+1} \hat{T}_\pi - \hat{V}_{t+1} \hat{T}_\pi \]

\[ \implies \|V_t - \hat{V}_t\|_2 \leq \|R - \hat{R}\|_2 + \|V_{t+1}\|_2 \|T_\pi - \hat{T}_\pi\|_2 + \|V_{t+1} - \hat{V}_{t+1}\|_2 \|\hat{T}_\pi\|_2 \]

\[ \leq \Delta_1 + \|V_{t+1}\|_2 \Delta_2 + \|V_{t+1} - \hat{V}_{t+1}\|_2 (\|\hat{T_\pi\|_2 + \Delta_2) \]

Plugging in Equation 10 and expanding the recursion, we get

\[ \|V_t - \hat{V}_t\|_2 \leq (\Delta_1 + H \|R\|_2 \|T_\pi\|_2^H) \sum_{i=0}^{H-t-1} (\|\hat{T_\pi\|_2 + \Delta_2)^i \]

And therefore
\[ \|V_0 - \hat{V}_0\|_2 \leq (\Delta_1 + H \|R\|_2 \|T_\pi\|^H H(\|T\|_2 + \Delta_2)^H \] (15)

Reducing to runtime notation, we get

\[ \|V_0 - \hat{V}_0\|_2 = O(H^2(\|T_\pi\|^H, \Delta_2^H)(\|R\|_2 \|T_\pi\|^H + \Delta_1)) \]