Robust Deep Reinforcement Learning against Adversarial Perturbations on State Observations

Huan Zhang[*,1] **Hongge Chen**[*,2] **Chaowei Xiao**[3]
Bo Li[4] **Mingyan Liu**[5] **Duane Boning**[2] **Cho-Jui Hsieh**[1]
[1]UCLA [2] MIT [3]NVIDIA [4]UIUC [5]University of Michigan
huan@huan-zhang.com, chenhg@mit.edu, chaoweix@nvidia.com,
lbo@illinois.edu,mingyan@umich.edu,boning@mtl.mit.edu,chohsieh@cs.ucla.edu
[*]Huan Zhang and Hongge Chen contributed equally.

Abstract

A deep reinforcement learning (DRL) agent observes its states through observations, which may contain natural measurement errors or adversarial noises. Since the observations deviate from the true states, they can mislead the agent into making suboptimal actions. Several works have shown this vulnerability via adversarial attacks, but existing approaches on improving the robustness of DRL under this setting have limited success and lack for theoretical principles. We show that naively applying existing techniques on improving robustness for classification tasks, like adversarial training, is ineffective for many RL tasks. We propose the state-adversarial Markov decision process (SA-MDP) to study the fundamental properties of this problem, and develop a theoretically principled policy regularization which can be applied to a large family of DRL algorithms, including proximal policy optimization (PPO), deep deterministic policy gradient (DDPG) and deep Q networks (DQN), for both discrete and continuous action control problems. We significantly improve the robustness of PPO, DDPG and DQN agents under a suite of strong white box adversarial attacks, including new attacks of our own. Additionally, we find that a robust policy noticeably improves DRL performance even without an adversary in a number of environments. Our code is available at https://github.com/chenhongge/StateAdvDRL.

1 Introduction

With deep neural networks (DNNs) as powerful function approximators, deep reinforcement learning (DRL) has achieved great success on many complex tasks [46, 35, 33, 65, 20] and even on some safety-critical applications (e.g., autonomous driving [75, 57, 49]). Despite achieving super-human level performance on many tasks, the existence of adversarial examples [70] in DNNs and many successful attacks to DRL [27, 4, 36, 50, 82] motivate us to study robust DRL algorithms.

When an RL agent obtains its current state via observations, the observations may contain uncertainty that naturally originates from unavoidable sensor errors or equipment inaccuracy. A policy not robust to such uncertainty can lead to catastrophic failures (e.g., the navigation setting in Figure 1). To ensure safety under the *worst case* uncertainty, we consider the adversarial setting where the state observation is adversarially perturbed from s to $\nu(s)$, yet the underlying true environment state s is unchanged. This setting is aligned with many adversarial attacks on state observations (e.g., [27, 36]) and cannot be characterized by existing tools such as partially observable Markov decision process (POMDP), because the conditional observation probabilities in POMDP cannot capture the adversarial (worst case) scenario. Studying the fundamental principles in this setting is crucial.

Before basic principles were developed, several early approaches [5, 40, 50] extended existing adversarial defenses for supervised learning, e.g., adversarial training [32, 39, 88] to improve robustness

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

under this setting. Specifically, we can attack the agent and generate trajectories adversarially during training time, and apply any existing DRL algorithm to hopefully obtain a robust policy. Unfortunately, we show that for most environments, naive adversarial training (e.g., putting adversarial states into the replay buffer) leads to unstable training and deteriorates agent performance [5, 15], or does not significantly improve robustness under strong attacks. Since RL and supervised learning are quite different problems, naively applying techniques from supervised learning to RL without a proper theoretical justification can be unsuccessful. To summarize, we study the theory and practice of robust RL against perturbations on state observations:

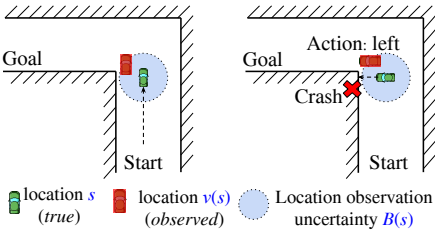


Figure 1: A car observes its location through sensors (e.g., GPS) and plans its route to the goal. Without considering the uncertainty in observed location (e.g., error of GPS coordinates), an unsafe policy may crash into the wall because $s \neq \nu(s)$.

- We formulate the perturbation on state observations as a modified Markov decision process (MDP), which we call state-adversarial MDP (SA-MDP), and study its fundamental properties. We show that under an optimal adversary, a stationary and Markovian optimal policy may not exist for SA-MDP.
- Based on our theory of SA-MDP, we propose a theoretically principled robust policy regularizer which is related to the total variation distance or KL-divergence on perturbed policies. It can be practically and efficiently applied to a wide range of RL algorithms, including PPO, DDPG and DQN.
- We conduct experiments on 10 environments ranging from Atari games with discrete actions to complex control tasks in continuous action space. Our proposed method significantly improves robustness under strong white-box attacks on state observations, including two *strong* attacks we design, the robust Sarsa attack (RS attack) and maximal action difference attack (MAD attack).

2 Related Work

Robust Reinforcement Learning Since each element of RL (observations, actions, transition dynamics and rewards) can contain uncertainty, robust RL has been studied from different perspectives. Robust Markov decision process (RMDP) [29, 47] considers the worst case perturbation from transition probabilities, and has been extended to distributional settings [83] and partially observed MDPs [48]. The agent observes the original true state from the environment and acts accordingly, but the environment can choose from a set of transition probabilities that minimizes rewards. Compared to our SA-MDP where the adversary changes only observations, in RMDP the ground-truth states are changed so RMDP is more suitable for modeling *environment parameter changes* (e.g., changes in physical parameters like mass and length, etc). RMDP theory has inspired robust deep Q-learning [63] and policy gradient algorithms [41, 12, 42] that are robust against small environmental changes.

Another line of works [51, 34] consider the adversarial setting of multi-agent reinforcement learning [71, 9]. In the simplest two-player setting (referred to as minimax games [37]), each agent chooses an action at each step, and the environment transits based on both actions. The regular Q function $Q(s, a)$ can be extended to $Q(S, a, o)$ where o is the opponent's action and Q-learning is still convergent. This setting can be extended to deep Q learning and policy gradient algorithms [34, 51]. Pinto et al. [51] show that learning an opponent simultaneously can improve the agent's performance as well as its robustness against environment turbulence and test conditions (e.g., change in mass or friction). Gu et al. [21] carried out real-world experiments on the two-player adversarial learning game. Tessler et al. [72] considered adversarial perturbations on the action space. Fu et al. [16] investigated how to learn a robust reward. All these settings are different from ours: we manipulate only the state observations but do not change the underlying environment (the true states) directly.

Adversarial Attacks on State Observations in DRL Huang et al. [27] evaluated the robustness of deep reinforcement learning policies through an FGSM based attack on Atari games with discrete actions. Kos & Song [31] proposed to use the value function to guide adversarial perturbation search. Lin et al. [36] considered a more complicated case where the adversary is allowed to attack only a subset of time steps, and used a generative model to generate attack plans luring the agent to a designated target state. Behzadan & Munir [4] studied black-box attacks on DQNs with discrete actions via transferability of adversarial examples. Pattanaik et al. [50] further enhanced adversarial attacks to DRL with multi-step gradient descent and better engineered loss functions. They require a critic or Q function to perform attacks. Typically, the critic learned during agent training is used.

We find that using this critic can be sub-optimal or impractical in many cases, and propose our two *critic-independent* and strong attacks (RS and MAD attacks) in Section 3.5. We refer the reader to recent surveys [82, 28] for a taxonomy and a comprehensive list of adversarial attacks in DRL setting.

Improving Robustness for State Observations in DRL For discrete action RL tasks, Kos & Song [31] first presented preliminary results of adversarial training on Pong (one of the simplest Atari environments) using weak FGSM attacks on pixel space. Behzadan & Munir [5] applied adversarial training to several Atari games with DQN, and found it challenging for the agent to adapt to the attacks during training time. These early approaches achieved much worse results than ours: for Pong, Behzadan & Munir [5] can improve reward under attack from -21 (lowest) to -5, yet is still far away from the optimal reward ($+21$). Recently, Mirman et al. [43], Fischer et al. [15] treat the *discrete action* outputs of DQN as labels, and apply existing certified defense for classification [44] to robustly predict actions using imitation learning. This approach outperforms [5], but it is unclear how to apply it to environments with continuous action spaces. Compared to their approach, our SA-DQN does not use imitation learning and achieves better performance on most environments.

For continuous action RL tasks (e.g., MuJoCo environments in OpenAI Gym), Mandlekar et al. [40] used a weak FGSM based attack with policy gradient to adversarially train a few simple RL tasks. Pattanaik et al. [50] used stronger multi-step gradient based attacks; however, their evaluation focused on robustness against environmental changes rather than state perturbations. Unlike our work which first develops principles and then applies to different DRL algorithms, these works directly extend adversarial training in supervised learning to the DRL setting and do not reliably improve test time performance under strong attacks in Section 4. A few concurrent works [56, 64] consider a smoothness regularizer similar to ours: [56] studied an attack setting to MDP similar to ours and proposed Lipschitz regularization, but it was applied to DQN with discrete actions only. [64] adopted virtual adversarial training also for the continuous-action settings but focused on improving generalization instead of robustness. In our paper, we provide theoretical justifications for our robustness regularizer from the perspective of constrained policy optimization [1], systematically apply our approach to multiple RL algorithms (PPO, DDPG and DQN), propose more effective adversarial attacks and conduct comprehensive empirical evaluations under a suit of strong adversaries.

Other related works include [24], which proposed a meta online learning procedure with a master agent detecting the presence of the adversary and switching between a few sub-policies, but did not discuss how to train a single agent robustly. [11] applied adversarial training specifically for RL-based path-finding algorithms. [38] considered the worst-case scenario during rollouts for existing DQN agents to ensure safety, but it relies on an existing policy and does not include a training procedure.

3 Methodology

3.1 State-Adversarial Markov Decision Process (SA-MDP)

Notations A Markov decision process (MDP) is defined as $(\mathcal{S}, \mathcal{A}, R, p, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition probability of environment, where $\mathcal{P}(\mathcal{S})$ defines the set of all possible probability measures on $\mathcal{S}$. The transition probability $p(s'|s,a) = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$, where t is the time step. We denote a stationary policy as $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, the set of all stochastic and Markovian policies as Π_{MR}, the set of all deterministic and Markovian policies as Π_{MD}. Discount factor $0 < \gamma < 1$.

In state-adversarial MDP (SA-MDP), we introduce an adversary $\nu(s) : \mathcal{S} \to \mathcal{S}$ [1]. The adversary perturbs only the state observations of the agent, such that the action is taken as $\pi(a|\nu(s))$; the environment still transits from the true state s rather than $\nu(s)$ to the next state. Since $\nu(s)$ can be different from s, the agent's action from $\pi(a|\nu(s))$ may be sub-optimal, and thus the adversary is able to reduce the reward. In real world RL problems, the adversary can be reflected as the worst case noise in measurement or state estimation uncertainty. Note that this scenario is different from the two-player Markov game [37] where both players see unperturbed true environment states and interact with the environment directly; the opponent's action can change the true state of the game.

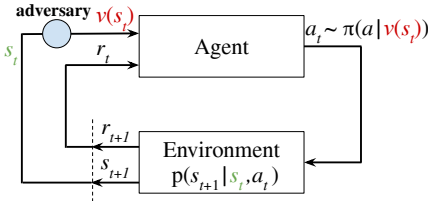


Figure 2: Reinforcement learning with perturbed state observations. The agent observes a perturbed state $\nu(s_t)$ rather than the true environment state s_t.

[1] Our analysis also holds for a stochastic adversary. The optimal adversary is deterministic (see Lemma 1).

To allow a formal analysis, we first make the assumption for the adversary ν:

Assumption 1 (Stationary, Deterministic and Markovian Adversary)**.** *$\nu(s)$ is a deterministic function $\nu : \mathcal{S} \to \mathcal{S}$ which only depends on the current state s, and ν does not change over time.*

This assumption holds for many adversarial attacks [27, 36, 31, 50]. These attacks only depend on the current state input and the policy or Q network so they are Markovian; the network parameters are frozen at test time, so given the same s the adversary will generate the same (stationary) perturbation. We leave the formal analysis of non-Markovian, non-stationary adversaries as future work.

If the adversary can perturb a state s arbitrarily without bounds, the problem can become trivial. To fit our analysis to the most realistic settings, we need to restrict the power of an adversary. We define perturbation set $B(s)$, to restrict the adversary to perturb a state s only to a predefined set of states:

Definition 1 (Adversary Perturbation Set)**.** *We define a set $B(s)$ which contains all allowed perturbations of the adversary. Formally, $\nu(s) \in B(s)$ where $B(s)$ is a set of states and $s \in \mathcal{S}$.*

$B(s)$ is usually a set of task-specific "neighboring" states of s (e.g., bounded sensor measurement errors), which makes the observation still meaningful (yet not accurate) even with perturbations. After defining B, an SA-MDP can be represented as a 6-tuple $(\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$.

Analysis of SA-MDP We first derive Bellman Equations and a basic policy evaluation procedure, then we discuss the possibility of obtaining an optimal policy for SA-MDP. The adversarial value and action-value functions under ν in an SA-MDP are similar to those of a regular MDP:

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

where the reward at step-t is defined as r_t and $\pi\circ\nu$ denotes the policy under observation perturbations: $\pi(a|\nu(s))$. Based on these two definitions, we first consider the simplest case with *fixed* π and ν:

Theorem 1 (Bellman equations for fixed π and ν)**.** *Given $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ and $\nu : \mathcal{S} \to \mathcal{S}$, we have*

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

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

The proof of Theorem 1 is simple, as when π, ν are fixed, they can be "merged" as a single policy, and existing results from MDP can be directly applied. Now we consider a more complicated case, where we want to find the value functions under *optimal adversary* $\nu^*(\pi)$, minimizing the total expected reward for a *fixed* π. The optimal adversarial value and action-value functions are defined as:

$$\tilde{V}_{\pi\circ\nu^*}(s) = \min_{\nu}\tilde{V}_{\pi\circ\nu}(s), \quad \tilde{Q}_{\pi\circ\nu^*}(s,a) = \min_{\nu}\tilde{Q}_{\pi\circ\nu}(s,a).$$

Theorem 2 (Bellman contraction for optimal adversary)**.** *Define Bellman operator $\mathscr{L} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$,*

$$(\mathscr{L}\tilde{V})(s) = \min_{s_\nu\in B(s)}\sum_{a\in\mathcal{A}}\pi(a|s_\nu)\sum_{s'\in\mathcal{S}}p(s'|s,a)\left[R(s,a,s') + \gamma\tilde{V}(s')\right]. \tag{1}$$

The Bellman equation for optimal adversary ν^* can then be written as: $\tilde{V}_{\pi\circ\nu^*} = \mathscr{L}\tilde{V}_{\pi\circ\nu^*}$. Additionally, $\mathscr{L}$ is a contraction that converges to $\tilde{V}_{\pi\circ\nu^*}$.

Theorem 2 says that given a *fixed* policy π, we can evaluate its performance (value functions) under the optimal (strongest) adversary, through a Bellman contraction. It is functionally similar to the "policy evaluation" procedure in regular MDP. The proof of Theorem 2 is in the same spirit as the proof of Bellman optimality equations for solving the optimal policy for an MDP; the important difference here is that we solve the optimal adversary, for a *fixed* policy π. Given π, value functions for MDP and SA-MDP can be vastly different. Here we show a 3-state toy environment in Figure 3; an optimal MDP policy is to take action 2 in S_1, action 1 in S_2 and S_3. Under the presence of an adversary $\nu(S_1) = S_2, \nu(S_2) = S_1, \nu(S_3) = S_1$, this policy receives zero total reward as the adversary can make the action $\pi(a|\nu(s))$ totally wrong regardless of the states. On the other hand, a

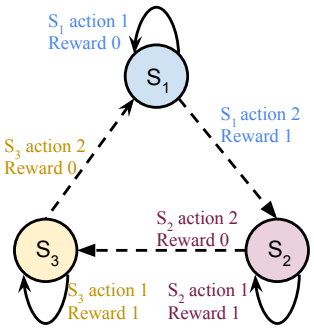


Figure 3: A toy environment.

policy taking random actions on all three states (which is a non-optimal policy for MDP) is unaffected by the adversary and obtains non-zero rewards in SA-MDP. Details are given in Appendix A.

Finally, we discuss our ultimate quest of finding an *optimal* policy π^* under the strongest adversary $\nu^*(\pi)$ in the SA-MDP setting (we use the notation $\nu^*(\pi)$ to explicit indicate that ν^* is the optimal adversary for a given π). An optimal policy should be the best among all policies on every state:

$$\tilde{V}_{\pi^* \circ \nu^*(\pi^*)}(s) \geq \tilde{V}_{\pi \circ \nu^*(\pi)}(s) \quad \text{for } \forall s \in \mathcal{S} \text{ and } \forall \pi, \tag{2}$$

where both π and ν are not fixed. The first question is, what policy classes we need to consider for π^*. In MDPs, deterministic policies are sufficient. We show that this does not hold anymore in SA-MDP:

Theorem 3. *There exists an SA-MDP and some stochastic policy $\pi \in \Pi_{MR}$ such that we cannot find a better deterministic policy $\pi' \in \Pi_{MD}$ satisfying $\tilde{V}_{\pi' \circ \nu^*(\pi')}(s) \geq \tilde{V}_{\pi \circ \nu^*(\pi)}(s)$ for all $s \in \mathcal{S}$.*

The proof is done by constructing a counterexample where some stochastic policies are better than *any* other deterministic policies in SA-MDP (see Appendix A). Contrarily, in MDP, for any stochastic policy we can find a deterministic policy that is at least as good as the stochastic one. Unfortunately, even looking for both deterministic and stochastic policies still cannot always find an optimal one:

Theorem 4. *Under the optimal ν^*, an optimal policy $\pi^* \in \Pi_{MR}$ does not always exist for SA-MDP.*

The proof follows the same counterexample as in Theorem 3. The optimal policy π^* requires to have $\tilde{V}_{\pi^* \circ \nu^*(\pi^*)}(s) \geq \tilde{V}_{\pi \circ \nu^*(\pi)}(s)$ for all s and any π. In an SA-MDP, sometimes we have to make a trade-off between the value of states and no policy can maximize the values of all states.

Despite the difficulty of finding an optimal policy under the optimal adversary, we show that under certain assumptions, the loss in performance due to an optimal adversary can be bounded:

Theorem 5. *Given a policy π for a non-adversarial MDP and its value function is $V_\pi(s)$. Under the optimal adversary ν in SA-MDP, for all $s \in \mathcal{S}$ we have*

$$\max_{s \in \mathcal{S}} \left\{ V_\pi(s) - \tilde{V}_{\pi \circ \nu^*(\pi)}(s) \right\} \leq \alpha \max_{s \in \mathcal{S}} \max_{\hat{s} \in B(s)} \mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s})) \tag{3}$$

where $\mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$ is the total variation distance between $\pi(\cdot|s)$ and $\pi(\cdot|\hat{s})$, and $\alpha := 2[1 + \frac{\gamma}{(1-\gamma)^2}] \max_{(s,a,s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}} |R(s,a,s')|$ is a constant that does not depend on π.

Theorem 5 says that as long as differences between the action distributions under state perturbations (the term $\mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$) are not too large, the performance gap between $\tilde{V}_{\pi \circ \nu^*}(s)$ (state value of SA-MDP) and $V_\pi(s)$ (state value of regular MDP) can be bounded. An important consequence is the motivation of regularizing $\mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$ during training to obtain a policy robust to strong adversaries. The proof is based on tools developed in constrained policy optimization [1], which gives an upper bound on value functions given two policies with bounded divergence. In our case, we desire that a bounded state perturbation $\hat{s}$ produces bounded divergence between $\pi(\cdot|s)$ and $\pi(\cdot|\hat{s})$.

We now study a few practical DRL algorithms, including both deep Q-learning (DQN) for discrete actions and actor-critic based policy gradient methods (DDPG and PPO) for continuous actions.

3.2 State-Adversarial DRL for Stochastic Policies: A Case Study on PPO

We start with the most general case where the policy $\pi(a|s)$ is stochastic (e.g., in PPO [60]). The total variation distance is not easy to compute for most distributions, so we upper bound it again by KL divergence: $\mathrm{D}_{\mathrm{TV}}(\pi(a|s), \pi(a|\hat{s})) \leq \sqrt{\frac{1}{2}\mathrm{D}_{\mathrm{KL}}(\pi(a|s) \| \pi(a|\hat{s}))}$. When Gaussian policies are used, we denote $\pi(a|s) \sim \mathcal{N}(\mu_s, \Sigma_s)$ and $\pi(a|\hat{s}) \sim \mathcal{N}(\mu_{\hat{s}}, \Sigma_{\hat{s}})$. The KL-divergence can be given as:

$$\mathrm{D}_{\mathrm{KL}}(\pi(a|s) \| \pi(a|\hat{s})) = \frac{1}{2} \left(\log |\Sigma_{\hat{s}} \Sigma_s^{-1}| + \mathrm{tr}(\Sigma_{\hat{s}}^{-1} \Sigma_s) + (\mu_{\hat{s}} - \mu_s)^\top \Sigma_{\hat{s}}^{-1} (\mu_{\hat{s}} - \mu_s) - |\mathcal{A}| \right). \tag{4}$$

Regularizing KL distance (4) for all $\hat{s} \in B(s)$ will lead to a smaller upper bound in (21), which is directly related to agent performance under optimal adversary. In PPO, the mean terms μ_s, $\mu_{\hat{s}}$ are produced by neural networks: $\mu_{\theta_\mu}(s)$ and $\mu_{\theta_\mu}(\hat{s})$, and we assume Σ is a diagonal matrix independent of state s ($\Sigma_{\hat{s}} = \Sigma_s = \Sigma$). Regularizing the above KL-divergence over all s from sampled trajectories and all $\hat{s} \in B(s)$ leads to the following state-adversarial regularizer for PPO, ignoring constant terms:

$$\mathcal{R}_{\mathrm{PPO}}(\theta_\mu) = \frac{1}{2} \sum_s \max_{\hat{s} \in B(s)} \left(\mu_{\theta_\mu}(\hat{s}) - \mu_{\theta_\mu}(s) \right)^\top \Sigma^{-1} \left(\mu_{\theta_\mu}(\hat{s}) - \mu_{\theta_\mu}(s) \right) := \frac{1}{2} \sum_s \max_{\hat{s} \in B(s)} \mathcal{R}_s(\hat{s}, \theta_\mu). \tag{5}$$

We replace $\max_{s\in\mathcal{S}}$ term in Theorem 5 with a more practical and optimizer-friendly summation over all states in sampled trajectory. A similar treatment was used in TRPO [33] which was also derived as a KL-based regularizer, albeit on θ_μ space rather than on state space. However, minimizing (5) is challenging as it is a minimax objective, and we also have $\nabla_{\hat{s}}\mathcal{R}(\hat{s},\theta_\mu)|_{\hat{s}=s}=0$ so using gradient descent directly cannot solve the inner maximization problem to a local maximum. Instead of using the more expensive second order methods, we propose two first order approaches to solve (5): convex relaxations of neural networks, and Stochastic Gradient Langevin Dynamics (SGLD). Here we focus on discussing convex relaxation based method, and we defer SGLD based solver to Section C.2.

Convex relaxation of non-linear units in neural networks enables an efficient analysis of the outer bounds for a neural network [80, 87, 67, 13, 79, 77, 58, 68]. Several works have used it for certified adversarial defenses [81, 44, 76, 19, 89], but here we leverage it as a generic optimization tool for solving minimax functions involving neural networks. Using this technique, we can obtain an upper bound for $\mathcal{R}_s(\hat{s},\theta_\mu)$: $\overline{\mathcal{R}}_s(\theta_\mu)\geq\mathcal{R}_s(\hat{s},\theta_\mu)$ for all $\hat{s}\in B(s)$. $\overline{\mathcal{R}}_s(\theta_\mu)$ is also a function of θ_μ and can be seen as a transformed neural network (e.g., the dual network in Wong & Kolter [80]), and computing $\overline{\mathcal{R}}_s(\theta_\mu)$ is only a constant factor slower than computing $\mathcal{R}_s(s,\theta_\mu)$ (for a fixed s) when an efficient relaxation [44, 19, 89] is used. We can then solve the following minimization problem:

$$\min_{\theta_\mu}\frac{1}{2}\sum_s\overline{\mathcal{R}}_s(\theta_\mu)\geq\min_{\theta_\mu}\frac{1}{2}\sum_s\max_{\hat{s}\in B(s)}\mathcal{R}_s(\hat{s},\theta_\mu)=\min_{\theta_\mu}\mathcal{R}_{\text{PPO}}(\theta_\mu).$$

Since we minimize an *upper bound* of the inner max, the original objective (5) is guaranteed to be minimized. Using convex relaxations can also provide certain *robustness certificates* for DRL as a bonus (e.g., we can guarantee an action has bounded changes under bounded perturbations), discussed in Appendix E. We use `auto_LiRPA`, a recently developed tool [84], to give $\overline{\mathcal{R}}_s(\theta_\mu)$ efficiently and automatically. Once the inner maximization problem is solved, we can add $\mathcal{R}_{\text{PPO}}$ as part of the policy optimization objective, and solve PPO using stochastic gradient descent (SGD) as usual.

Although Eq (5) looks similar to smoothness based regularizers in (semi-)supervised learning settings to avoid overfitting [45] and improve robustness [88], our regularizer is based on the foundations of SA-MDP. Our theory justifies the use of such a regularizer in reinforcement learning setting, while [45, 88] are developed for quite different settings not related to reinforcement learning.

3.3 State-Adversarial DRL for Deterministic Policies: A Case Study on DDPG

DDPG learns a deterministic policy $\pi(s):\mathcal{S}\to\mathcal{A}$, and in this situation, the total variation distance $\mathrm{D}_{TV}(\pi(\cdot|s),\pi(\cdot|\hat{s}))$ is malformed, as the densities at different states s and $\hat{s}$ are very likely to be completely non-overlapping. To address this issue, we define a smoothed version of policy, $\bar{\pi}(a|s)$ in DDPG, where we add independent Gaussian noise with variance σ^2 to each action: $\bar{\pi}(a|s)\sim\mathcal{N}(\pi(s),\sigma^2 I_{|\mathcal{A}|})$. Then we can compute $\mathrm{D}_{TV}(\bar{\pi}(\cdot|s),\bar{\pi}(\cdot|\hat{s}))$ using the following theorem:

Theorem 6. $\mathrm{D}_{TV}(\bar{\pi}(\cdot|s),\bar{\pi}(\cdot|\hat{s}))=\sqrt{2/\pi}\frac{d}{\sigma}+O(d^3)$, *where* $d=\|\pi(s)-\pi(\hat{s})\|_2$.

Thus, as long as we can penalize $\sqrt{2/\pi}\frac{d}{\sigma}$, the total variation distance between the two smoothed distributions can be bounded. In DDPG, we parameterize the policy as a policy network π_{θ_π}. Based on Theorem 5, the robust policy regularizer for DDPG is:

$$\mathcal{R}_{\text{DDPG}}(\theta_\pi)=\sqrt{2/\pi}(1/\sigma)\sum_s\max_{\hat{s}\in B(s)}\|\pi_{\theta_\pi}(s)-\pi_{\theta_\pi}(\hat{s})\|_2 \tag{6}$$

for each state s in a sampled batch of states, we need to solve a maximization problem, which can be done using SGLD or convex relaxations similarly as we have shown in Section 3.2. Note that the smoothing procedure can be done completely at test time, and during training time our goal is to keep $\max_{\hat{s}\in B(s)}\|\pi_{\theta_\pi}(s)-\pi_{\theta_\pi}(\hat{s})\|_2$ small. We show the full SA-DDPG algorithm in Appendix G.

3.4 State-Adversarial DRL for Q Learning: A Case Study on DQN

The action space for DQN is finite, and the deterministic action is determined by the max Q value: $\pi(a|s)=1$ when $a=\arg\max_{a'}Q(s,a')$ and 0 otherwise. The total variation distance in this case is

$$\mathrm{D}_{TV}(\pi(\cdot|s),\pi(\cdot|\hat{s}))=\begin{cases}0 & \arg\max_a\pi(a|s)=\arg\max_a\pi(a|\hat{s})\\ 1 & \text{otherwise.}\end{cases}$$

Thus, we want to make the top-1 action stay unchanged after perturbation, and we can use a hinge-like robust policy regularizer, where $a^*(s) = \arg\max_a Q_\theta(s,a)$ and c is a small positive constant:

$$\mathcal{R}_{\text{DQN}}(\theta) := \sum_s \max\{\max_{\hat{s}\in B(s)} \max_{a\neq a^*} Q_\theta(\hat{s},a) - Q_\theta(\hat{s},a^*(s)), -c\}. \tag{7}$$

The sum is over all s in a sampled batch. Other loss functions (e.g., cross-entropy) are also possible as long as the aim is to keep the top-1 action to stay unchanged after perturbation. This setting is similar to the robustness of classification tasks, if we treat $a^*(s)$ as the "correct" label, thus many robust classification techniques can be applied as in [43, 15]. The maximization can be solved using projected gradient descent (PGD) or convex relaxation of neural networks. Due to its similarity to classification, we defer the details on solving $\mathcal{R}_{\text{DQN}}(\theta)$ and full SA-DQN algorithm to Appendix H.

3.5 Robust Sarsa (RS) and Maximal Action Difference (MAD) Attacks

In this section we propose two strong adversarial attacks under Assumption 1 for continuous action tasks trained using PPO or DDPG. For this setting, Pattanaik et al. [50] and many follow-on works use the gradient of $Q(s,a)$ to provide the direction to update states adversarially in K steps:

$$s^{k+1} = s^k - \eta \cdot \text{proj}\left[\nabla_{s^k} Q(s^0, \pi(s^k))\right], \quad k = 0,\ldots,K-1, \text{and define } \hat{s} := s^K. \tag{8}$$

Here proj$[\cdot]$ is a projection to $B(s)$, η is the learning rate, and s^0 is the state under attack. It attempts to find a state $\hat{s}$ triggering an action $\pi(\hat{s})$ minimizing the action-value at state s^0. The formulation in [50] has a glitch that the gradient is evaluated as $\nabla_{s^k} Q(s^k, \pi(s^k))$ rather than $\nabla_{s^k} Q(s^0, \pi(s^k))$. We found that the corrected form (8) is more successful. If Q is a perfect action-value function, $\hat{s}$ leads to the worst action that minimizes the value at s^0. However, this attack has a few drawbacks:

- Attack strength strongly depends on critic quality; if Q is poorly learned, is not robust against small perturbations or has obfuscated gradients, the attack fails as no correct update direction is given.
- It relies on the Q function which is specific to the training process, but not used during roll-out.
- Not applicable to many actor-critic methods (e.g., TRPO and PPO) using a learned value function $V(s)$ instead of $Q(s,a)$. Finding $\hat{s} \in B(s)$ minimizing $V(s)$ does not correctly reflect the setting of perturbing observations, as $V(\hat{s})$ represents the value of $\hat{s}$ rather than the value of taking $\pi(\hat{s})$ at s^0.

When we evaluate the robustness of a policy, we desire it to be independent of a specific critic network to avoid these problems. We thus propose two novel *critic independent* attacks for DDPG and PPO.

Robust Sarsa (RS) attack. Since π is fixed during evaluation, we can learn its corresponding $Q^\pi(s,a)$ using on-policy temporal-difference (TD) algorithms similar to Sarsa [55] without knowing the critic network used during training. Additionally, we find that the robustness of $Q^\pi(s,a)$ is very important; if $Q^\pi(s,a)$ is not robust against small perturbations (e.g., given a state s_0, a small change in a will significantly reduce $Q^\pi(s_0,a)$ which does not reflect the true action-value), it cannot provide a good direction for attacks. Based on these, we learn $Q^\pi(s,a)$ (parameterized as an NN with parameters θ) with a TD loss as in Sarsa and an additional robustness objective to minimize:

$$L_{RS}(\theta) = \sum_{i\in[N]} \left[r_i + \gamma Q^\pi_{RS}(s_i', a_i') - Q^\pi_{RS}(s_i,a_i)\right]^2 + \lambda_{RS} \sum_{i\in[N]} \max_{\hat{a}\in B(a_i)} \left(Q^\pi_{RS}(s_i,\hat{a}) - Q^\pi_{RS}(s_i,a_i)\right)^2$$

N is the batch size and each batch contains N tuples of transitions (s,a,r,s',a') sampled from agent rollouts. The first summation is the TD-loss and the second summation is the robustness penalty with regularization λ_{RS}. $B(a_i)$ is a small set near action a_i (e.g., a ℓ_∞ ball of norm 0.05 when action is normalized between 0 to 1). The inner maximization can be solved using convex relaxation of neural networks as we have done in Section 3.3. Then, we use $Q^\pi_{\theta_{RS}}$ to perform critic-based attacks as in (8). This attack sometimes significantly outperforms the attack using the critic trained along with the policy network, as its attack strength does not depend on the quality of an existing critic. We give the detailed procedure for RS attack and show the importance of the robust objective in appendix D.

Maximal Action Difference (MAD) attack. We propose another simple yet very effective attack which does not depend on a critic. Following our Theorem 5 and 6, we can find an adversarial state $\hat{s}$ by *maximizing* $D_{\text{KL}}\left(\pi(\cdot|s)\|\pi(\cdot|\hat{s})\right)$. For actions parameterized by Gaussian mean $\pi_{\theta_\pi}(s)$ and covariance matrix Σ (independent of s), we minimize $L_{\text{MAD}}(\hat{s}) := -D_{\text{KL}}\left(\pi(\cdot|s)\|\pi(\cdot|\hat{s})\right)$ to find $\hat{s}$:

$$\arg\min_{\hat{s}\in B(s)} L_{\text{MAD}}(\hat{s}) = \arg\max_{\hat{s}\in B(s)} \left(\pi_{\theta_\pi}(s) - \pi_{\theta_\pi}(\hat{s})\right)^\top \Sigma^{-1} \left(\pi_{\theta_\pi}(s) - \pi_{\theta_\pi}(\hat{s})\right). \tag{9}$$

For DDPG we can simply set $\Sigma = I$. The objective can be optimized using SGLD to find a good $\hat{s}$.

Table 1: Average episode rewards ± standard deviation over 50 episodes on 3 baselines and SA-PPO. We report natural rewards (no attacks) and rewards under five adversarial attacks. In each row we bold the best (lowest) attack reward over all five attacks. The gray rows are the most robust agents.

Env.	ϵ	Method	Natural Reward	Attack Reward					Best Attack
				Critic	Random	MAD	RS	RS+MAD	
Hopper	0.075	PPO (vanilla)	3167.6± 541.6	1799.0± 935.2	2915.2±677.7	1505.2± 382.0	779.4± 33.2	**733.8± 44.6**	733
		PPO (adv. 50%)	174± 146	69 ±83	141± 128	**42± 46**	49 ±50	44± 43	42
		PPO (adv. 100%)	6.1± 2.6	4.4 ±1.8	6.1± 3.2	5.8± 2.7	3.8 ±0.9	**3.6 ±0.5**	3.6
		SA-PPO (SGLD)	3523.1±329.0	3665.5± 8.2	3080.2± 745.4	2996.6± 786.4	**1403.3± 55.0**	1415.4± 72.0	1403.3
		SA-PPO (Convex)	3704.1± 2.2	3698.4± 4.4	3708.7± 23.8	3443.1± 466.672	1235.8± 50.2	**1224.2± 47.8**	1224.2
Walker2d	0.05	PPO (vanilla)	4619.5± 38.2	4589.3± 12.4	4480.0±465.3	4469.1±715.6	**913.7± 54.3**	926.8±66.3	913.7
		PPO (adv. 50%)	-11 ± 0.9	-10.6 ± 0.86	-10.99 ± 0.95	-10.78 ± 0.89	**-11.55 ± 0.79**	-11.37 ± 0.87	-11.55
		PPO (adv. 100%)	-113 ± 4.14	-111.9 ± 4.13	-111 ± 4.27	-112 ± 4.08	-114.4 ± 4.0	**-114.5 ± 4.09**	-114.5
		SA-PPO (SGLD)	4911.8± 188.9	5019.0± 65.2	4894.8± 139.9	4755.7± 413.1	2605.6± 1255.7	**2468.4 ±1205**	2468.4
		SA-PPO (Convex)	4486.6± 60.7	4572.0± 52.3	4475.0± 48.7	4343.4± 329.4	2168.2± 665.4	**2076.1± 666.7**	2076.1
Humanoid	0.075	PPO (vanilla)	5270.6±1074.3	5494.7± 118.7	5648.3± 86.8	1140.3± 534.8	1036.0± 420.2	**884.1± 356.3**	884.1
		PPO (adv. 50%)	234± 28	198 ± 58	240 ± 19.4	148 ± 73	**98 ± 69**	101.5 ± 66.4	98
		PPO (adv. 100%)	141.4 ± 20.6	140.25 ± 16.6	142.13 ± 16	140.23 ± 34.5	113.2 ± 18.5	**112.6 ± 13.88**	112.6
		SA-PPO (SGLD)	6624.0± 25.5	6587.0± 23.1	6614.1± 21.4	6586.4± 23.5	6200.5± 818.1	**6073.8± 1108.1**	6073.8
		SA-PPO (Convex)	6400.6± 156.8	6397.9 ±35.6	6207.9± 783.3	6379.5± 30.5	4707.2± 1359.1	**4690.3± 1244.89**	4690.3

4 Experiments

In our experiments[2], the set of adversarial states $B(s)$ is defined as an ℓ_∞ norm ball around s with a radius ϵ: $B(s) := \{\hat{s} : \|s - \hat{s}\|_\infty \leq \epsilon\}$. Here ϵ is also referred to as the perturbation budget. In MuJoCo environments, the ℓ_∞ norm is applied on normalized state representations.

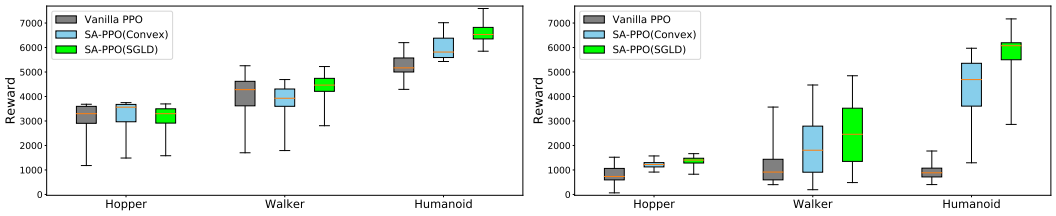
Evaluation of SA-PPO We use the PPO implementation from [14], which conducted hyperparameter search and published the optimal hyperparameters for PPO on three Mujoco environments in OpenAI Gym [7]. We use their optimal hyperparameters for PPO, and the same set of hyperparameters for SA-PPO without further tuning. We run Walker2d and Hopper 2×10^6 steps and Humanoid 1×10^7 steps to ensure convergence. Our vanilla PPO agents achieve similar or better performance than reported in the literature [14, 25, 22]. Detailed hyperparameters are in Appendix F. SA-PPO has one additional regularization parameter, κ_{PPO}, for the regularizer $\mathcal{R}_{\text{PPO}}$, which is chosen in {0.003, 0.01, 0.03, 0.1, 0.3, 1.0}. We solve the SA-PPO objective using both SGLD and convex relaxation methods. We include three baselines: vanilla PPO, and adversarially trained PPO [40, 50] with 50% and 100% training steps under critic attack [50]. The attack is conducted by finding $\hat{s} \in B(s)$ minimizing $V(\hat{s})$ instead of $Q(s, \pi(\hat{s}))$, as PPO does not learn a Q function during learning. We evaluate agents using 5 attacks, including our strong RS and MAD attacks, detailed in Appendix D.

In Table 1, naive adversarial training deteriorates performance and does not reliably improve robustness in all three environments. Our RS attack and MAD attacks are very effective in all environments and achieve significantly lower rewards than critic and random attacks; this shows the importance of evaluation using strong attacks. SA-PPO, solved either by SGLD or the convex relaxation objective, *significantly improves robustness* against strong attacks. Additionally, SA-PPO achieves natural performance (without attacks) similar to that of vanilla PPO in Walker2d and Hopper, and *significantly improves the reward in Humanoid environment*. Humanoid has a high state-space dimension (376) and is usually hard to train [22], and our results suggest that a robust objective can be helpful even in a non-adversarial setting. Because PPO training can have large performance variance across multiple runs, to show that our SA-PPO can consistently obtain a robust agent, we repeatedly train each environment using SA-PPO and vanilla PPO at least **15 times** and attack all agents obtained. In Figures 4a and 4b we show the box plot of the natural and best attack reward for these PPO and SA-PPO agents. We can see that the best attack rewards of most SA-PPO agents are consistently better than PPO agents (in terms of median, 25% and 75% percentile rewards over multiple repetitions).

Evaluation of SA-DDPG We use a high quality DDPG implementation [62] as our baseline, achieving similar or better performance on five Mujoco environments as in the literature [35, 17]. For SA-DDPG, we use the same set of hyperparameters as in DDPG [62] (detailed in Appendix G), except for the additional regularization term κ_{DDPG} for $\mathcal{R}_{\text{DDPG}}$ which is searched in $\{0.1, 0.3, 1.0, 3.0\}$ for InvertedPendulum and Reacher due to their low dimensionality and $\{30, 100, 300, 1000\}$ for other environments. We include vanilla DDPG, adversarially trained DDPG [50] (attacking 50% or 100% steps) as baselines. We use the same set of 5 attacks as in 1. In Table 2, we observe that naive adversarial training is not very effective in many environments. SA-DDPG (solved by SGLD or convex relaxations) significantly improves robustness under strong attacks in all 5 environments.

[2]Code and pretrained agents available at https://github.com/chenhongge/StateAdvDRL

Figure 4: Box plots of natural and attack rewards for PPO and SA-PPO. Each box is obtained from at least **15 agents** trained with the same hyperparameters as in agents reported in Table 1. The red lines inside the boxes are median rewards, and the upper and lower sides of the boxes show 25% and 75% percentile rewards of 30 agents. The line segments outside of the boxes show min or max rewards.



(a) Natural episode rewards (no attacks) (b) Rewards under the best (strongest) attacks

Table 2: Average episode rewards ± standard deviation over 50 episodes on DDPG, adversarial training [50] (50% and 100% steps) and SA-DDPG. Each number represents an agent with *median* reward under the best attack over 11 training runs with identical hyperparameters. Due to large variance in RL, it important to report median metrics. **Bold** numbers indicate the most robust agents. Full results of all five attacks are in Table 6 and statistics over multiple training runs are in Figure 12.

Environment		Ant	Hopper	Inverted Pendulum	Reacher	Walker2d
ℓ_∞ norm perturbation budget ϵ		0.2	0.075	0.3	1.5	0.05
DDPG (vanilla)	Natural Reward	1487 ± 850	3302 ± 762	1000 ± 0	-4.37 ± 1.54	1870 ± 1418
	Attack Reward (best)	142 ± 180	606 ± 124	92 ± 1	-27.87 ± 4.38	790 ± 985
DDPG (adv. 50%)	Natural Reward	1487 ± 850	3302 ± 762	1000 ± 0	-4.37 ± 1.54	1870 ± 1418
	Attack Reward (best)	31 ± 179	41 ± 105	39 ± 0	-25.81 ± 6.53	837 ± 722
DDPG (adv. 100%)	Natural Reward	1082 ± 574	973 ± 0	1000 ± 0	-5.71 ± 1.80	462 ± 569
	Attack Reward (best)	-52 ± 231	24 ± 15	82 ± 0	-27.44 ± 4.05	302 ± 260
SA-DDPG (SGLD)	Natural Reward	2186 ± 534	3068 ± 223	1000 ± 0	-5.38 ± 1.74	3318 ± 680
	Attack Reward (best)	$\mathbf{2007 \pm 686}$	$\mathbf{1609 \pm 676}$	423 ± 281	$\mathbf{-12.10 \pm 4.58}$	1210 ± 979
SA-DDPG (convex relax)	Natural Reward	2254 ± 430	3128 ± 453	1000 ± 0	-5.24 ± 2.06	4540 ± 1562
	Attack Reward (best)	1820 ± 635	1202 ± 402	$\mathbf{1000 \pm 0}$	-12.44 ± 3.77	$\mathbf{1986 \pm 1993}$

Similar to the observations on SA-PPO, SA-DDPG can improve natural agent performance in environments (Ant and Walker2d) with relatively high dimensional state space (111 and 17).

Evaluation of SA-DQN We implement Double DQN [73] and Prioritized Experience Replay [59] on four Atari games. We train Atari agents for 6 million frames for both vanilla DQN and SA-DQN. Detailed parameters and training procedures are in Appendix H. We normalize the pixel values to $[0, 1]$ and we add ℓ_∞ adversarial noise with norm $\epsilon = 1/255$. We include vanilla DQNs and adversarially trained DQNs with 50% of frames under attack [5] during training time as baselines, and we report results of robust imitation learning [15]. We evaluate all environments under 10-step untargeted PGD attacks, except that results from [15] were evaluated using a weaker 4-step PGD attack. For the most robust Atari agents (SA-DQN convex), we additionally attack them using 50-step PGD attacks, and find that the rewards do not further reduce. In Table 3, we see that our SA-DQN achieves much higher rewards under attacks in most environments, and naive adversarial training is mostly ineffective under strong attacks. We obtain better rewards than [15] in most environments, as we learn the agents directly rather than using two-step imitation learning.

Table 3: Average episode rewards ± std and *action certification rates* over 50 episodes on three baselines and SA-DQN. We report natural rewards (no attacks) and PGD attack rewards (under 10-step or 50-step PGD). Action certification rate is the proportion of the actions during rollout that are guaranteed unchanged by any attacks within the given ϵ. Training time is reported in Section H.

Environment		Pong	Freeway	BankHeist	RoadRunner
ℓ_∞ norm perturbation budget ϵ		1/255			
DQN (vanilla)	Natural Reward	21.0 ± 0.0	34.0 ± 0.2	1308.4 ± 24.1	45534.0 ± 7066.0
	PGD Attack Reward (10 steps)	-21.0±0.0	0.0±0.0	56.4±21.2	0.0±0.0
	Action Cert. Rate	0.0	0.0	0.0	0.0
DQN Adv. Training (attack 50% frames) Behzadan & Munir [5]	Natural Reward	10.1 ± 6.6	25.4±0.8	1126.0±70.9	22944.0±6532.5
	PGD Attack Reward (10 steps)	-21.0 ± 0.0	0.0±0.0	9.4±13.6	14.0±34.7
	Action Cert. Rate	0.0	0.0	0.0	0.0
Imitation learning Fischer et al. [15]	Natural Reward	19.73	32.93	238.66	12106.67
	PGD Attack Reward (4 steps)	18.13	**32.53**	190.67	5753.33
SA-DQN (PGD)	Natural Reward	21.0±0.0	33.9 ± 0.4	1245.2±14.5	34032.0±3845.0
	PGD Attack Reward (10 steps)	21.0±0.0	23.7 ± 2.3	1006.0±226.4	20402.0±7551.1
	Action Cert. Rate	0.0	0.0	0.0	0.0
SA-DQN (convex)	Natural Reward	21.0 ± 0.0	30.0±0.0	1235.4±9.8	44638.0±7367.0
	PGD Attack Reward (10 steps)	**21.0 ± 0.0**	30.0±0.0	**1232.4±16.2**	**44732.0±8059.5**
	PGD Attack Reward (50 steps)	**21.0 ± 0.0**	30.0±0.0	**1234.6±16.6**	**44678.0±6954.0**
	Action Cert. Rate	1.000	1.000	0.984	0.475

Robustness certificates. When our robust policy regularizer is trained using convex relaxations, we can obtain certain robustness certificates under observation perturbations. For a simple environment like Pong, we can guarantee actions do not change for all frames during rollouts, thus guarantee the cumulative rewards under perturbation. For SA-DDPG, the *upper bounds* on the maximal ℓ_2 difference in action changes is a few times smaller than baselines on all 5 environments (see Appendix I). Unfortunately, for most RL tasks, due to the complexity of environment dynamics and reward process, it is impossible to obtain a "certified reward" as the certified test error in supervised learning settings [80, 89]. We leave further discussions on these challenges in Appendix E.

Broader Impact

Reinforcement learning is a central part of modern artificial intelligence and is still under heavy development in recent years. Unlike supervised learning which has been widely deployed in many commercial and industrial applications, reinforcement learning has not been widely accepted and deployed in real-world settings. Thus, the study of reinforcement learning robustness under the adversarial attacks settings receives less attentions than the supervised learning counterparts.

However, with the recent success of reinforcement learning on many complex games such as Go [66], StartCraft [74] and Dota 2 [6], we will not be surprised if we will see reinforcement learning (especially, deep reinforcement learning) being used in everyday decision making tasks in near future. The potential social impacts of applying reinforcement learning agents thus must be investigated before its wide deployment. One important aspect is the trustworthiness of an agent, where robustness plays a crucial rule. The robustness considered in our paper is important for many realistic settings such as sensor noise, measurement errors, and man-in-the-middle (MITM) attacks for a DRL system. if the robustness of reinforcement learning can be established, it has the great potential to be applied into many mission-critical tasks such as autonomous driving [61, 57, 86] to achieve superhuman performance.

On the other hand, one obstacle for applying reinforcement learning to real situations (beyond games like Go and StarCraft) is the "reality gap": a well trained reinforcement learning agent in a simulation environment can easily fail in real-world experiments. One reason for this failure is the potential sensing errors in real-world settings; this was discussed as early as in Brooks [8] in 1992 and still remains an open challenge now. Although our experiments were done in simulated environments, we believe that a smoothness regularizer like the one proposed in our paper can also benefit agents tested in real-world settings, such as robot hand manipulation [2].

Acknowledgments and Disclosure of Funding

We acknowledge the support by NSF IIS-1901527, IIS-2008173, ARL-0011469453, and scholarship by IBM. The authors thank Ge Yang and Xiaocheng Tang for helpful discussions.

References

[1] Achiam, J., Held, D., Tamar, A., and Abbeel, P. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 22–31. JMLR. org, 2017.

[2] Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[3] Balunovic, M. and Vechev, M. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations*, 2019.

[4] Behzadan, V. and Munir, A. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pp. 262–275. Springer, 2017.

[5] Behzadan, V. and Munir, A. Whatever does not kill deep reinforcement learning, makes it stronger. *arXiv preprint arXiv:1712.09344*, 2017.

[6] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] Brooks, R. A. Artificial life and real robots. In *Proceedings of the First European Conference on artificial life*, pp. 3–10, 1992.

[9] Bu, L., Babu, R., De Schutter, B., et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[10] Bubeck, S., Eldan, R., and Lehec, J. Finite-time analysis of projected Langevin Monte Carlo. In *Advances in Neural Information Processing Systems*, pp. 1243–1251, 2015.

[11] Chen, T., Niu, W., Xiang, Y., Bai, X., Liu, J., Han, Z., and Li, G. Gradient band-based adversarial training for generalized attack immunity of A3C path finding. *arXiv preprint arXiv:1807.06752*, 2018.

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

[13] Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., and Kohli, P. A dual approach to scalable verification of deep networks. *UAI*, 2018.

[14] Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *arXiv preprint arXiv:2005.12729*, 2020.

[15] Fischer, M., Mirman, M., and Vechev, M. Online robustness training for deep reinforcement learning. *arXiv preprint arXiv:1911.00887*, 2019.

[16] Fu, J., Luo, K., and Levine, S. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[17] Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[18] Gelfand, S. B. and Mitter, S. K. Recursive stochastic algorithms for global optimization in $\mathbb{R}^d$. *SIAM Journal on Control and Optimization*, 29(5):999–1018, 1991.

[19] Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Mann, T., and Kohli, P. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[20] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, pp. 2829–2838, 2016.

[21] Gu, Z., Jia, Z., and Choset, H. Adversary A3C for robust reinforcement learning. *arXiv preprint arXiv:1912.00330*, 2019.

[22] Hämäläinen, P., Babadi, A., Ma, X., and Lehtinen, J. PPO-CMA: Proximal policy optimization with covariance matrix adaptation. *arXiv preprint arXiv:1810.02541*, 2018.

[23] Hasselt, H. V. Double q-learning. In *Advances in neural information processing systems*, pp. 2613–2621, 2010.

[24] Havens, A., Jiang, Z., and Sarkar, S. Online robust policy learning in the presence of unknown adversaries. In *Advances in Neural Information Processing Systems*, pp. 9916–9926, 2018.

[25] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[27] Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[28] Ilahi, I., Usama, M., Qadir, J., Janjua, M. U., Al-Fuqaha, A., Hoang, D. T., and Niyato, D. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *arXiv preprint arXiv:2001.09684*, 2020.

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

[30] Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pp. 267–274, 2002.

[31] Kos, J. and Song, D. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

[32] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[33] Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

[34] Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., and Russell, S. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4213–4220, 2019.

[35] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[36] Lin, Y.-C., Hong, Z.-W., Liao, Y.-H., Shih, M.-L., Liu, M.-Y., and Sun, M. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*, 2017.

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

[38] Lütjens, B., Everett, M., and How, J. P. Certified adversarial robustness for deep reinforcement learning. *arXiv preprint arXiv:1910.12908*, 2019.

[39] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.

[40] Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3932–3939. IEEE, 2017.

[41] Mankowitz, D. J., Mann, T. A., Bacon, P.-L., Precup, D., and Mannor, S. Learning robust options. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[42] Mankowitz, D. J., Levine, N., Jeong, R., Abdolmaleki, A., Springenberg, J. T., Mann, T., Hester, T., and Riedmiller, M. Robust reinforcement learning for continuous control with model misspecification. *arXiv preprint arXiv:1906.07516*, 2019.

[43] Mirman, M., Fischer, M., and Vechev, M. Distilled agent DQN for provable adversarial robustness, 2018. URL `https://openreview.net/forum?id=ryeAy3AqYm`.

[44] Mirman, M., Gehr, T., and Vechev, M. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pp. 3575–3583, 2018.

[45] Miyato, T., Maeda, S.-i., Koyama, M., Nakae, K., and Ishii, S. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*, 2015.

[46] 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. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[47] Nilim, A. and El Ghaoui, L. Robustness in Markov decision problems with uncertain transition matrices. In *Advances in Neural Information Processing Systems*, pp. 839–846, 2004.

[48] Osogami, T. Robust partially observable Markov decision process. In *International Conference on Machine Learning*, pp. 106–115, 2015.

[49] Pan, X., You, Y., Wang, Z., and Lu, C. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[50] Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., and Chowdhary, G. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2040–2042. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[51] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2817–2826. JMLR. org, 2017.

[52] Pirotta, M., Restelli, M., Pecorino, A., and Calandriello, D. Safe policy iteration. In *International Conference on Machine Learning*, pp. 307–315, 2013.

[53] Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[54] Raginsky, M., Rakhlin, A., and Telgarsky, M. Non-convex learning via stochastic gradient Langevin dynamics: a nonasymptotic analysis. *arXiv preprint arXiv:1702.03849*, 2017.

[55] Rummery, G. A. and Niranjan, M. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[56] Russo, A. and Proutiere, A. Optimal attacks on reinforcement learning policies. *arXiv preprint arXiv:1907.13548*, 2019.

[57] Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[58] Salman, H., Yang, G., Zhang, H., Hsieh, C.-J., and Zhang, P. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems 32*, pp. 9832–9842. Curran Associates, Inc., 2019.

[59] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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

[61] Shalev-Shwartz, S., Shammah, S., and Shashua, A. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[62] Shangtong, Z. Modularized implementation of deep RL algorithms in PyTorch. `https://github.com/ShangtongZhang/DeepRL`, 2018.

[63] Shashua, S. D.-C. and Mannor, S. Deep robust Kalman filter. *arXiv preprint arXiv:1703.02310*, 2017.

[64] Shen, Q., Li, Y., Jiang, H., Wang, Z., and Zhao, T. Deep reinforcement learning with smooth policy. *ICML*, 2020.

[65] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[66] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[67] Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pp. 10825–10836, 2018.

[68] Singh, G., Gehr, T., Püschel, M., and Vechev, M. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019.

[69] Sutton, R. S., Barto, A. G., et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[70] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *ICLR*, 2013.

[71] Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337, 1993.

[72] Tessler, C., Efroni, Y., and Mannor, S. Action robust reinforcement learning and applications in continuous control. *arXiv preprint arXiv:1901.09184*, 2019.

[73] Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[74] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[75] Voyage. Introducing voyage deepdrive -unlocking the potential of deep reinforcement learning. `https://news.voyage.auto/introducing-voyage-deepdrive-69b3cf0f0be6`, 2019.

[76] Wang, S., Chen, Y., Abdou, A., and Jana, S. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.

[77] Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pp. 6367–6377, 2018.

[78] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003, 2016.

[79] Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Daniel, L., Boning, D., and Dhillon, I. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning*, pp. 5273–5282, 2018.

[80] Wong, E. and Kolter, Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5283–5292, 2018.

[81] Wong, E., Schmidt, F., Metzen, J. H., and Kolter, J. Z. Scaling provable adversarial defenses. In *NIPS*, 2018.

[82] Xiao, C., Pan, X., He, W., Peng, J., Sun, M., Yi, J., Li, B., and Song, D. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470*, 2019.

[83] Xu, H. and Mannor, S. Distributionally robust markov decision processes. In *Advances in Neural Information Processing Systems*, pp. 2505–2513, 2010.

[84] Xu, K., Shi, Z., Zhang, H., Huang, M., Chang, K.-W., Kailkhura, B., Lin, X., and Hsieh, C.-J. Automatic perturbation analysis on general computational graphs. *arXiv preprint arXiv:2002.12920*, 2020.

[85] Xu, P., Chen, J., Zou, D., and Gu, Q. Global convergence of langevin dynamics based algorithms for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 3122–3133, 2018.

[86] You, C., Lu, J., Filev, D., and Tsiotras, P. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. *Robotics and Autonomous Systems*, 114:1–18, 2019.

[87] Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. Efficient neural network robustness certification with general activation functions. In *NIPS*, 2018.

[88] Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., and Jordan, M. I. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[89] Zhang, H., Chen, H., Xiao, C., Li, B., Boning, D., and Hsieh, C.-J. Towards stable and efficient training of verifiably robust neural networks. *ICLR*, 2020.

[90] Zhang, Y., Liang, P., and Charikar, M. A hitting time analysis of stochastic gradient Langevin dynamics. *arXiv preprint arXiv:1702.05575*, 2017.

Appendix

• Readers who are interested in SA-MDP can find an example of SA-MDP in Section A and complete proofs in Section B.

• Readers who are interested in adversarial attacks can find more details about our new attacks and existing attacks in Section D. Especially, we discussed how a robust critic can help in attacking RL, and show experiments on the improvements gained by the robustness objective during attack.

• Readers who want to know more details of optimization techniques to solve our state-adversarial robust regularizers can refer to Section C, including more background on convex relaxations of neural networks in Section C.1.

• We provide detailed algorithm and hyperparameters for SA-PPO in Section F. We provide details for SA-DDPG in Section G. We provide details for SA-DQN in Section H.

• We provide more empirical results in Section I. To demonstrate the convergence of our algorithm, we repeat each experiment at least 15 times and plot the convergence of rewards during multiple runs. We found that for some environments (like Humanoid) we can consistently improve baseline performance. We also evaluate some settings under multiple perturbation strength ϵ.

A An example of SA-MDP

We first show a simple environment and solve it under different settings of MDP and SA-MDP. The environment has three states $\mathcal{S} = \{S_1, S_2, S_3\}$ and 2 actions $\mathcal{A} = \{A_1, A_2\}$. The transition probabilities and rewards are defined as below (unmentioned probabilities and rewards are 0):

$$\begin{aligned}
&\Pr(s' = S_1 | s = S_1, a = A_1) = 1.0 \\
&\Pr(s' = S_2 | s = S_1, a = A_2) = 1.0 \\
&\Pr(s' = S_2 | s = S_2, a = A_2) = 1.0 \\
&\Pr(s' = S_3 | s = S_2, a = A_1) = 1.0 \\
&\Pr(s' = S_1 | s = S_3, a = A_2) = 1.0 \\
&\Pr(s' = S_2 | s = S_3, a = A_1) = 1.0 \\
&R(s = S_1, a = A_2, s' = S_2) = 1.0 \\
&R(s = S_2, a = A_1, s' = S_2) = 1.0 \\
&R(s = S_3, a = A_1, s' = S_3) = 1.0
\end{aligned}$$

The environment is illustrated in Figure 5. For the power of adversary, we allow ν to perturb one

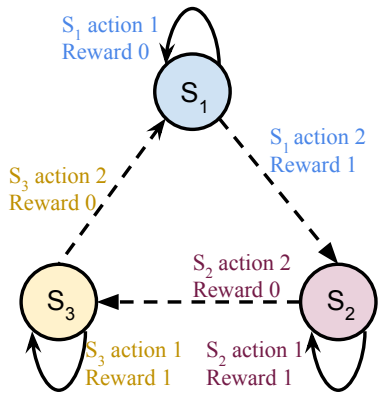


Figure 5: A simple 3-state toy environment.

state to any other two neighbouring states:

$$B_\nu(S_1) = B_\nu(S_2) = B_\nu(S_3) = \{S_1, S_2, S_3\}$$

Now we evaluate various policies for MDP and SA-MDP for this environment. We use $\gamma = 0.99$ as the discount factor. A stationary and Markovian policy in this environment can be described by 3 parameters p_{11}, p_{21}, p_{31} where $p_{ij} \in [0, 1]$ denotes the probability $\Pr(a = A_j | s = S_i)$. We denote the value function as V for MDP and $\tilde{V}$ for SA-MDP.

- **Optimal Policy for MDP.** For a regular MDP, the optimal solution is $p_{11} = 0$, $p_{21} = 1$, $p_{31} = 1$. We take A_2 to receive reward and leave S_1, and then keep doing A_1 in S_2 and S_3. The values for each state are $V(S_1) = V(S_2) = V(S_3) = \frac{1}{1-\gamma} = 100$, which is optimal. However, this policy obtains $\tilde{V}(S_1) = \tilde{V}(S_2) = \tilde{V}(S_3) = 0$ for SA-MDP, because we can set $\nu(S_1) = S_2$, $\nu(S_2) = S_1$, $\nu(S_3) = S_1$ and consequentially we always take the wrong action receiving 0 reward.
- **A Stochastic Policy for MDP and SA-MDP.** We consider a stochastic policy where $p_{11} = p_{21} = p_{31} = 0.5$. Under this policy, we randomly stay or move in each state, and has a 50% probability of receiving a reward. The adversary ν has no power because π is the same for all states. In this situation, $V(S_1) = \tilde{V}(S_1) = V(S_2) = \tilde{V}(S_2) = V(S_3) = \tilde{V}(S_3) = \frac{0.5}{1-0.99} = 50$ for both MDP and SA-MDP. This can also be seen as an extreme case of Theorem 5, where the policy does not change under adversary in all states, so there is no performance loss in SA-MDP.
- **Deterministic Policies for SA-MDP.** Now we consider all $2^3 = 8$ possible deterministic policies for SA-MDP. Note that if for any state S_i we have $p_{i1} = 0$ and another state S_j we have $p_{j1} = 1$, we always have $\tilde{V}(S_1) = \tilde{V}(S_2) = \tilde{V}(S_3) = 0$. This is because we can set $\nu(S_1) = S_j$, $\nu(S_2) = S_i$ and $\nu(S_3) = S_i$ and always receive a 0 reward. Thus the only two possible other policies are $p_{11} = p_{21} = p_{31} = 0$ and $p_{11} = p_{21} = p_{31} = 1$, respectively. For $p_{11} = p_{21} = p_{31} = 1$ we have $\tilde{V}(S_1) = 0, \tilde{V}(S_2) = \tilde{V}(S_3) = 100$ as we always take A_1 and never transit to other states; for $p_{11} = p_{21} = p_{31} = 0$, we circulate through all three states and only receive a reward when we leave A_1. We have $\tilde{V}(S_1) = \frac{1}{1-\gamma^3} \approx 33.67$, $\tilde{V}(S_2) = \frac{\gamma^2}{1-\gamma^3} \approx 33.00$ and $\tilde{V}(S_3) = \frac{\gamma}{1-\gamma^3} \approx 33.33$.

Figure 6, 7, 8 give the graphs of $\tilde{V}(S_1)$, $\tilde{V}(S_2)$ and $\tilde{V}(S_3)$ under three different settings of p_{11}. The figures are generated using Algorithm 1.

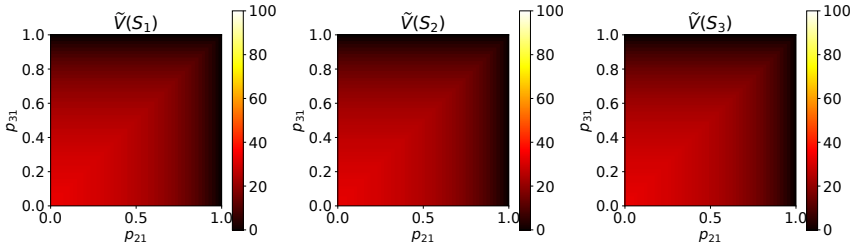


Figure 6: Value functions for SA-MDP when $p_{11} = 0$, with $p_{21} \in [0, 1]$, $p_{31} \in [0, 1]$

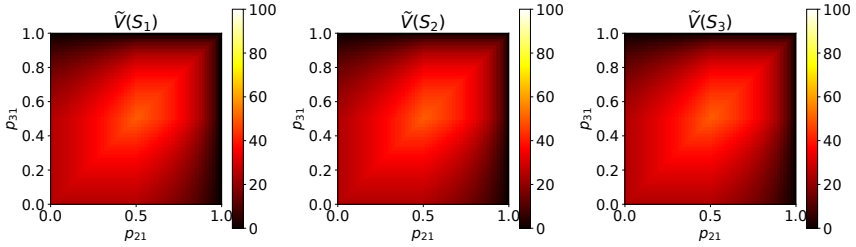


Figure 7: Value functions for SA-MDP when $p_{11} = 0.5$, with $p_{21} \in [0, 1]$, $p_{31} \in [0, 1]$

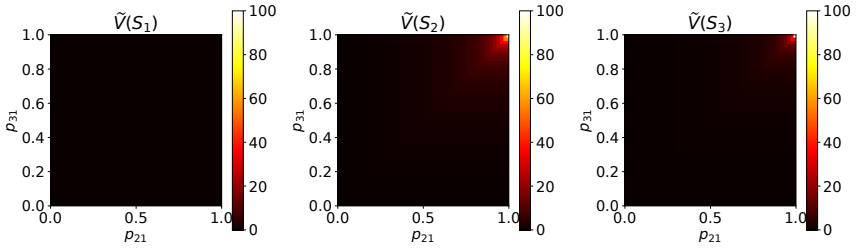


Figure 8: Value functions for SA-MDP when $p_{11} = 1.0$, with $p_{21} \in [0, 1]$, $p_{31} \in [0, 1]$

B Proofs for State-Adversarial Markov Decision Process

Theorem 1 (Bellman equations for fixed π and ν)**.** *Given $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ and $\nu : \mathcal{S} \to \mathcal{S}$, we have*

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

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

Proof. Based on the definition of $\tilde{V}_{\pi\circ\nu}(s)$:

$$\begin{aligned}
\tilde{V}_{\pi\circ\nu}(s) &= \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \\
&= \mathbb{E}_{\pi\circ\nu}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right] \\
&= \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[r_{t+1} + \gamma \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right]\right] \\
&= \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}_{\pi\circ\nu}(s')\right]
\end{aligned} \tag{10}$$

The recursion for $\tilde{Q}_{\pi\circ\nu}(s,a)$ can be derived similarly. Additionally, we note the following useful relationship between $\tilde{V}_{\pi\circ\nu}(s)$ and $\tilde{Q}_{\pi\circ\nu}(s,a)$:

$$\tilde{V}_{\pi\circ\nu}(s) = \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \tilde{Q}_{\pi\circ\nu}(s,a) \tag{11}$$

□

Before starting to prove Theorem 2, first we show that finding the optimal adversary ν^* given a fixed π for a SA-MDP can be cast into the problem of finding an optimal policy in a regular MDP.

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

Proof. For an SA-MDP $M = (\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$ and a fixed policy π, we define a regular MDP $\hat{M} = (\mathcal{S}, \hat{\mathcal{A}}, \hat{R}, \hat{p}, \gamma)$ such that $\hat{\mathcal{A}} = \mathcal{S}$, and ν is the policy for $\hat{M}$. To prove this lemma, we use a slight extension of a stochastic adversary, where $\nu : \mathcal{S} \to \mathcal{P}(\hat{A})$. At each state s, our policy ν gives a probability distribution $\nu(\cdot|s)$ indicating that we perturb a state s to $\hat{s}$ with probability $\nu(\hat{s}|s)$ in the SA-MDP M.

For $\hat{M}$, the reward function is defined as:

$$\hat{R}(s,\hat{a},s') = \begin{cases} -\frac{\sum_{a\in\mathcal{A}} \pi(a|\hat{a}) p(s'|s,a) R(s,a,s')}{\sum_{a\in\mathcal{A}} \pi(a|\hat{a}) p(s'|s,a)} & \text{for } s, s' \in \mathcal{S} \text{ and } \hat{a} \in B(s) \subset \hat{\mathcal{A}} = \mathcal{S}, \\ C & \text{for } s, s' \in \mathcal{S} \text{ and } \hat{a} \notin B(s). \end{cases} \tag{12}$$

The transition probability $\hat{p}$ is defined as

$$\hat{p}(s'|s,\hat{a}) = \sum_{a\in\mathcal{A}} \pi(a|\hat{a})p(s'|s,a) \quad \text{for } s,s' \in \mathcal{S} \text{ and } \hat{a} \in \hat{\mathcal{A}} = \mathcal{S}.$$

For the case of $\hat{a} \in B(s)$, the above reward function definition is based on the intuition that when the agent receives a reward r at a time step given s, a, s', the adversary's reward is $\hat{r} = -r$. Note that we consider r as a random variable given s, a, s'. To give the distribution of rewards for adversary $p(\hat{r}|s,\hat{a},s')$, we follow the conditional probability which marginalizes π:

$$\begin{aligned}
p(\hat{r}|s,\hat{a},s') &= \frac{p(\hat{r},s'|s,\hat{a})}{p(s'|s,\hat{a})} \\
&= \frac{\sum_a p(\hat{r},s'|a,s,\hat{a})\pi(a|s,\hat{a})}{\sum_a p(s'|a,s,\hat{a})\pi(a|s,\hat{a})} \\
&= \frac{\sum_a p(\hat{r},s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\
&= \frac{\sum_a p(\hat{r}|s',a,s)p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})}
\end{aligned} \tag{13}$$

Considering that $R(s,a,s') := \mathbb{E}[r|s',a,s] = -\mathbb{E}[\hat{r}|s',a,s]$, and taking an expectation in Eq. (13) over $\hat{r}$ yield the first case in (12):

$$\begin{aligned}
\hat{R}(s,\hat{a},s') &:= \mathbb{E}[\hat{r}|s,\hat{a},s'] \\
&= \sum_{\hat{r}} \hat{r} \frac{\sum_a p(\hat{r}|s',a,s)p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\
&= \frac{\sum_a \left[\sum_{\hat{r}} \hat{r}p(\hat{r}|s',a,s)\right] p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\
&= \frac{\sum_a \mathbb{E}[\hat{r}|s',a,s]p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\
&= -\frac{\sum_a R(s,a,s')p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})}
\end{aligned} \tag{14}$$

The reward for adversary's actions outside $B(s)$ is a constant C such that

$$C < \min\Big\{ -\overline{M}, \quad \frac{\gamma}{(1-\gamma)}\underline{M} - \frac{1}{(1-\gamma)}\overline{M}\Big\},$$

where $\underline{M} := \min_{s,a,s'} R(s,a,s')$ and $\overline{M} := \max_{s,a,s'} R(s,a,s')$. We have for $\forall(s,\hat{a},s')$,

$$C < \hat{R}(s,\hat{a},s') \leq -\underline{M},$$

and for $\forall \hat{a} \in B(s)$, according to Eq. (14),

$$-\overline{M} \leq \hat{R}(s,\hat{a},s') \leq -\underline{M}.$$

According basic properties of MDP [53, 69], we know that the $\hat{M}$ has an optimal policy ν^*, which satisfies $\hat{V}_{\pi\circ\nu^*}(s) \geq \hat{V}_{\pi\circ\nu}(s)$ for $\forall s, \forall \nu$. We also know that this ν^* is deterministic and assigns a unit mass probability for the optimal action of each s.

We define $\mathfrak{N} := \{\nu : \forall s,\ \exists \hat{a} \in B(s),\ \nu(\hat{a}|s) = 1\}$ which restricts the adversary from taking an action not in $B(s)$, and claim that $\nu^* \in \mathfrak{N}$. If this is not true for a state s^0, we have

$$\begin{aligned}
\hat{V}_{\pi\circ\nu^*}(s^0) &= \mathbb{E}_{\hat{p},\nu^*}\Big[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s^0\Big] \\
&= C + \mathbb{E}_{\hat{p},\nu^*}\left[\sum_{k=1}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s^0\right] \\
&\leq C - \frac{\gamma}{1-\gamma}\underline{M} \\
&< -\frac{1}{1-\gamma}\overline{M} \\
&\leq \mathbb{E}_{\hat{p},\nu'}\Big[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s^0\Big] = \hat{V}_{\pi\circ\nu'}(s^0),
\end{aligned}$$

where the second equality holds because ν^* is deterministic, and the last inequality holds for any $\nu' \in \mathfrak{N}$. This contradicts the assumption that ν^* is optimal. So from now on in this proof we only study policies in $\mathfrak{N}$.

For any policy $\nu \in \mathfrak{N}$:

$$\begin{aligned}
\hat{V}_{\pi\circ\nu}(s) &= \mathbb{E}_{\hat{p},\nu}\Big[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s\Big] \\
&= \mathbb{E}_{\hat{p},\nu}\left[\hat{r}_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+2}|s_t = s\right] \\
&= \sum_{\hat{a}\in\mathcal{S}}\nu(\hat{a}|s)\sum_{s'\in\mathcal{S}}\hat{p}(s'|s,\hat{a})\left[\hat{R}(s,\hat{a},s') + \gamma\mathbb{E}_{\hat{p},\nu}\left[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+2}|s_{t+1} = s'\right]\right] \\
&= \sum_{\hat{a}\in\mathcal{S}}\nu(\hat{a}|s)\sum_{s'\in\mathcal{S}}\hat{p}(s'|s,\hat{a})\left[\hat{R}(s,\hat{a},s') + \gamma\hat{V}_{\pi\circ\nu}(s')\right] \qquad (15)
\end{aligned}$$

Note that all policies in $\mathfrak{N}$ are deterministic and this class of policies consists ν^*. Also, $\mathfrak{N}$ is consistent with the class of policies studied in Theorem 1. We denote the deterministic action $\hat{a}$ chosen by a $\nu \in \mathfrak{N}$ at s as $\nu(s)$. Then for $\forall\nu \in \mathfrak{N}$, we have

$$\begin{aligned}
\hat{V}_{\pi\circ\nu}(s) &= \sum_{s'\in\mathcal{S}}\hat{p}(s'|s,\nu(s))\left[\hat{R}(s,\hat{a},s') + \gamma\hat{V}_{\pi\circ\nu}(s')\right] \\
&= \sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}}\pi(a|\hat{a})p(s'|s,a)\left[-\frac{\sum_{a\in\mathcal{A}}\pi(a|\hat{a})p(s'|s,a)R(s,a,s')}{\sum_{a\in\mathcal{A}}\pi(a|\hat{a})p(s'|s,a)} + \gamma\hat{V}_{\pi\circ\nu}(s')\right] \\
&= \sum_{a\in\mathcal{A}}\pi(a|\nu(s))\sum_{s'\in\mathcal{S}}p(s'|s,a)\left[-R(s,a,s') + \gamma\hat{V}_{\pi\circ\nu}(s')\right], \qquad (16)
\end{aligned}$$

or

$$-\hat{V}_{\pi\circ\nu}(s) = \sum_{a\in\mathcal{A}}\pi(a|\nu(s))\sum_{s'\in\mathcal{S}}p(s'|s,a)\left[R(s,a,s') + \gamma(-\hat{V}_{\pi\circ\nu}(s'))\right]. \qquad (17)$$

Comparing (17) and (10), we know that $-\hat{V}_{\pi\circ\nu} = \tilde{V}_{\pi\circ\nu}$ for any $\nu \in \mathfrak{N}$. The optimal value function $\hat{V}_{\pi\circ\nu^*}$ satisfies:

$$\begin{aligned}
\hat{V}_{\pi\circ\nu^*}(s) &= \max_{\hat{a}\in B(s)}\sum_{s'\in\mathcal{S}}\hat{p}(s'|s,\hat{a})\left[\hat{R}(s,\hat{a},s') + \gamma\hat{V}_{\pi\circ\nu}(s')\right] \\
&= \max_{s_\nu\in B(s)}\sum_{a\in\mathcal{A}}\pi(a|s_\nu)\sum_{s'\in\mathcal{S}}p(s'|s,a)\left[-R(s,a,s') + \gamma\hat{V}_{\pi\circ\nu^*}(s')\right], \qquad (18)
\end{aligned}$$

where we denote the action $\hat{a}$ taken at s as s_ν. So for ν^*, since $-\hat{V}_{\pi\circ\nu^*} = \tilde{V}_{\pi\circ\nu^*}$, we have

$$\tilde{V}_{\pi\circ\nu^*}(s) = \min_{\hat{a}\in B(s)} \sum_{a\in\mathcal{A}} \pi(a|\hat{a}) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}_{\pi\circ\nu^*}(s')\right], \tag{19}$$

and $\tilde{V}_{\pi\circ\nu^*}(s) \leq \tilde{V}_{\pi\circ\nu}(s)$ for $\forall s, \forall \nu \in \mathfrak{N}$. Hence ν^* is also the optimal ν for $\tilde{V}_{\pi\circ\nu}$. □

Lemma 1 gives many good properties for the optimal adversary. First, an optimal adversary always exists under the regularity conditions where an optimal policy exists for a MDP. Second, we do not need to consider stochastic adversaries as there always exists an optimal deterministic adversary. Additionally, showing Bellman contraction for finding the optimal adversary can be done similarly as in obtaining the optimal policy in a regular MDP, as shown in the proof of Theorem 2.

Theorem 2 (Bellman contraction for optimal adversary)**.** *Define Bellman operator* $\mathscr{L} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$,

$$(\mathscr{L}\tilde{V})(s) = \min_{s_\nu\in B(s)} \sum_{a\in\mathcal{A}} \pi(a|s_\nu) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}(s')\right]. \tag{20}$$

The Bellman equation for optimal adversary ν^* *can then be written as:* $\tilde{V}_{\pi\circ\nu^*} = \mathscr{L}\tilde{V}_{\pi\circ\nu^*}$*. Additionally,* $\mathscr{L}$ *is a contraction that converges to* $\tilde{V}_{\pi\circ\nu^*}$*.*

Proof. Based on Lemma 1, this proof is technically similar to the proof of "optimal Bellman equation" in regular MDPs, where $\max$ over π is replaced by $\min$ over ν. By the definition of $\tilde{V}_{\pi\circ\nu^*}(s)$,

$$\begin{aligned}
\tilde{V}_{\pi\circ\nu^*}(s) &= \min_\nu \tilde{V}_{\pi\circ\nu}(s) \\
&= \min_\nu \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \\
&= \min_\nu \mathbb{E}_{\pi\circ\nu}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right] \\
&= \min_\nu \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r_{t+1} + \gamma \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right]\right] \\
&= \min_{s_\nu\in B_\nu(s)} \sum_{a\in\mathcal{A}} \pi(a|s_\nu) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r_{t+1} + \gamma \min_\nu \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right]\right] \\
&= \min_{s_\nu\in B_\nu(s)} \sum_{a\in\mathcal{A}} \pi(a|s_\nu) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r_{t+1} + \gamma \tilde{V}_{\pi\circ\nu^*}(s')\right]
\end{aligned}$$

This is the Bellman equation for the optimal adversary ν^*; ν^* is a fixed point of the Bellman operator $\mathscr{L}$.

Now we show the Bellman operator is a contraction. We have, if $\mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) \geq \mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s)$,

$$\begin{aligned}
&\mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) - \mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s) \\
&\leq \max_{s_\nu\in B_\nu(s)} \Big\{ \sum_{a\in\mathcal{A}} \pi(a|s_\nu) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[R(s,a,s') + \gamma \tilde{V}_{\pi\circ\nu_1}(s')\right] \\
&- \sum_{a\in\mathcal{A}} \pi(a|s_\nu) \sum_{s'\in\mathcal{S}} p(s'|s,a)\left[R(s,a,s') + \gamma \tilde{V}_{\pi\circ\nu_2}(s')\right] \Big\} \\
&= \gamma \max_{s_\nu\in B_\nu(s)} \sum_{a\in\mathcal{A}} \pi(a|s_\nu) \sum_{s'\in\mathcal{S}} p(s'|s,a)[\tilde{V}_{\pi\circ\nu_1}(s') - \tilde{V}_{\pi\circ\nu_2}(s')] \\
&\leq \gamma \max_{s_\nu\in B_\nu(s)} \sum_{a\in\mathcal{A}} \pi(a|s_\nu) \sum_{s'\in\mathcal{S}} p(s'|s,a) \|\tilde{V}_{\pi\circ\nu_1} - \tilde{V}_{\pi\circ\nu_2}\|_\infty \\
&= \gamma \|\tilde{V}_{\pi\circ\nu_1} - \tilde{V}_{\pi\circ\nu_2}\|_\infty
\end{aligned}$$

The first inequality comes from the fact that

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

where $x_2^* = \arg\min_{x_2} g(x_2)$. Similarly, we can prove $\mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s) - \mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) \leq \|\tilde{V}_{\pi\circ\nu_1} - \tilde{V}_{\pi\circ\nu_2}\|_\infty$ if $\mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s) > \mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s)$. Hence

$$\|\mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) - \mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s)\|_\infty = \max_s |\mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) - \mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s)| \leq \gamma\|\tilde{V}_{\pi\circ\nu_1} - \tilde{V}_{\pi\circ\nu_2}\|_\infty.$$

Then according to the Banach fixed-point theorem, since $0 < \gamma < 1$, $\tilde{V}_{\pi\circ\nu}$ converges to a unique fixed point, and this fixed point is $\tilde{V}_{\pi\circ\nu^*}$.

□

Algorithm 1 Policy Evaluation for an SA-MDP $(\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$

Input: Policy π, convergence threshold ε
Output: Values for policy π, detnoted as $\tilde{V}_{\pi\circ\nu^*}(s)$

```
Initialize array V(s) ← 0 for all s ∈ S
repeat
  Δ ← 0
  for all s ∈ S do
    v ← ∞, v_0 ← V(s)
    for all s_ν ∈ B(s) do
      v' ← Σ_{a∈A} π(a|s_ν) Σ_{s'∈S} p(s'|s,a) · [R(s,a,s') + γV(s')]
      v ← min(v, v')
    end for
    V(s) ← v
    Δ ← max(Δ, |v_0 − V(s)|)
  end for
until Δ < ε
Ṽ_{π∘ν*}(s) ← V(s)
```

A direct consequence of Theorem 2 is the policy evaluation algorithm (Algorithm 1) for SA-MDP, which obtains the values for each state under *optimal* adversary for a fixed policy π. For both Lemma 1 and Theorem 2, we only consider a fixed policy π, and in this setting finding an optimal adversary is not difficult. However, finding an optimal π under the optimal adversary is more challenging, as we can see in Section A, given the white-box attack setting where the adversary knows π and can choose optimal perturbations accordingly, an optimal policy for MDP can only receive zero rewards under optimal adversary. We now show two intriguing properties for optimal policies in SA-MDP:

Theorem 3. *There exists an SA-MDP and some stochastic policy $\pi \in \Pi_{MR}$ such that we cannot find a better deterministic policy $\pi' \in \Pi_{MD}$ satisfying $\tilde{V}_{\pi'\circ\nu^*(\pi')}(s) \geq \tilde{V}_{\pi\circ\nu^*(\pi)}(s)$ for all $s \in \mathcal{S}$.*

Proof. Proof by giving a counter example that no deterministic policy can be better than a random policy. The SA-MDP example in section A provided such a counter example: all 8 possible deterministic policies are no better than the stochastic policy $p_{11} = p_{21} = p_{31} = 0.5$. □

Theorem 4. *Under the optimal ν^*, an optimal policy $\pi^* \in \Pi_{MR}$ does not always exist for SA-MDP.*

Proof. We will show that the SA-MDP example in section A does not have an optimal policy. First, for π_1 where $p_{11} = p_{21} = p_{31} = 1$ we have $\tilde{V}_{\pi_1\circ\nu^*(\pi_1)}(S_1) = 0, \tilde{V}_{\pi_1\circ\nu^*(\pi_1)}(S_2) = \tilde{V}_{\pi_1\circ\nu^*(\pi_1)}(S_3) = 100$. This policy is not an optimal policy since we have π_2 where $p_{11} = p_{21} = p_{31} = 0.5$ that can achieve $\tilde{V}_{\pi_2\circ\nu^*(\pi_2)}(S_1) = \tilde{V}_{\pi_2\circ\nu^*(\pi_2)}(S_2) = \tilde{V}_{\pi_2\circ\nu^*(\pi_2)}(S_3) = 50$ and $\tilde{V}_{\pi_2\circ\nu^*(\pi_2)}(S_1) > \tilde{V}_{\pi_1\circ\nu^*(\pi_1)}(S_1)$.

An optimal policy π, if exists, must be better than π_1 and have $\tilde{V}_{\pi\circ\nu^*(\pi)}(S_1) > 0, V_{\pi\circ\nu^*(\pi)}(S_2) = V_{\pi\circ\nu^*(\pi)}(S_3) = 100$. In order to achieve $V_{\pi\circ\nu^*(\pi)}(S_2) = V_{\pi\circ\nu^*(\pi)}(S_3) = 100$, we must set $p_{21} = p_{31} = 1$ since it is the only possible way to start from S_2 and S_3 and receive +1 reward for every step. We can still change p_{11} to probabilities other than 1, however if $p_{11} < 1$ the adversary can set $\nu(S_2) = \nu(S_3) = S_1$ and reduce $V_{\pi\circ\nu^*(\pi)}(S_2)$ and $V_{\pi\circ\nu^*(\pi)}(S_3)$. Thus, no policy better than π_1 exists, and since π_1 is not an optimal policy, no optimal policy exists. □

Theorem 3 and Theorem 4 show that the classic definition of optimality is probably not suitable for SA-MDP. Further works can study how to obtain optimal policies for SA-MDP under some alternative definition of optimality, or using a more complex policy class (e.g., history dependent policies).

Theorem 5. *Given a policy π for a non-adversarial MDP and its value function is $V_\pi(s)$. Under the optimal adversary ν in SA-MDP, for all $s \in \mathcal{S}$ we have*

$$\max_{s\in\mathcal{S}} \left\{ V_\pi(s) - \tilde{V}_{\pi\circ\nu^*(\pi)}(s) \right\} \le \alpha \max_{s\in\mathcal{S}} \max_{\hat{s}\in B(s)} \mathrm{D_{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s})) \tag{21}$$

where $\mathrm{D_{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$ is the total variation distance between $\pi(\cdot|s)$ and $\pi(\cdot|\hat{s})$, and $\alpha := 2[1 + \frac{\gamma}{(1-\gamma)^2}] \max_{(s,a,s')\in\mathcal{S}\times\mathcal{A}\times\mathcal{S}} |R(s,a,s')|$ is a constant that does not depend on π.

Proof. Our proof is based on Theorem 1 in Achiam et al. [1]. In fact, many works in the literature have proved similar results under different scenarios [30, 52]. For an arbitrary starting state s_0 and two arbitrary policies π and π', Theorem 1 in Achiam et al. [1] gives an upper bound of $V_\pi(s_0) - V_{\pi'}(s_0)$. The bound is given by

$$\begin{aligned} V_\pi(s_0) - V_{\pi'}(s_0) \le & -\mathbb{E}_{\substack{s\sim d^\pi_{s_0} \\ a\sim\pi(\cdot|s) \\ s'\sim p(\cdot|a,s)}} \left[\left(\frac{\pi'(a|s)}{\pi(a|s)} - 1\right) R(s,a,s') \right] \\ & + \frac{2\gamma}{(1-\gamma)^2} \max_s \left\{ \mathbb{E}_{\substack{a\sim\pi'(\cdot|s) \\ s'\sim p(\cdot|a,s)}} \left[R(s,a,s') \right] \right\} \mathbb{E}_{s\sim d^\pi_{s_0}} \left[\mathrm{D}_{TV}(\pi(\cdot|s), \pi'(\cdot|s)) \right], \end{aligned} \tag{22}$$

where $d^\pi_{s_0}$ is the discounted future state distribution from s_0, defined as

$$d^\pi_{s_0}(s) := (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s | \pi, s_0). \tag{23}$$

Note that in Theorem 1 of Achiam et al. [1], the author proved a general form with an arbitrary function f and we assume $f \equiv 0$ in our proof. We also assume the starting state is deterministic, so J^π in Achiam et al. [1] is replaced by $V^\pi(s_0)$. Then we simply need to bound both terms on the right hand side of (22).

For the first term we know that

$$\begin{aligned} -\mathbb{E}_{\substack{s\sim d^\pi_{s_0} \\ a\sim\pi(\cdot|s) \\ s'\sim p(\cdot|a,s)}} \left[\left(\frac{\pi'(a|s)}{\pi(a|s)} - 1\right) R(s,a,s') \right] &= \sum_s d^\pi_{s_0}(s) \sum_a \left[\pi(a|s) - \pi'(a|s) \right] \sum_{s'} p(s'|s,a) R(s,a,s') \\ &\le \sum_s d^\pi_{s_0}(s) \sum_a \left| \pi(a|s) - \pi'(a|s) \right| \left| \sum_{s'} p(s'|s,a) R(s,a,s') \right| \\ &\le \max_{s,a,s'} |R(s,a,s')| \max_s \left\{ \sum_a \left| \pi(a|s) - \pi'(a|s) \right| \right\} \\ &= 2 \max_{s,a,s'} |R(s,a,s')| \max_s \mathrm{D}_{TV}(\pi(\cdot|s), \pi'(\cdot|s)) \end{aligned} \tag{24}$$

The second term is bounded by

$$\begin{aligned} & \frac{2\gamma}{(1-\gamma)^2} \max_s \left\{ \mathbb{E}_{\substack{a\sim\pi'(\cdot|s) \\ s'\sim p(\cdot|a,s)}} \left[R(s,a,s') \right] \right\} \mathbb{E}_{s\sim d^\pi_{s_0}} \left[\mathrm{D}_{TV}(\pi(\cdot|s), \pi'(\cdot|s)) \right] \\ & \le \frac{2\gamma}{(1-\gamma)^2} \max_{s,a,s'} |R(s,a,s')| \max_s \mathrm{D}_{TV}(\pi(\cdot|s), \pi'(\cdot|s)) \end{aligned} \tag{25}$$

Therefore, the RHS of (22) is bounded by $\alpha \max_s \mathrm{D}_{TV}(\pi(\cdot|s), \pi'(\cdot|s))$, where

$$\alpha = 2[1 + \frac{\gamma}{(1-\gamma)^2}] \max_{s,a,s'} |R(s,a,s')| \tag{26}$$

Finally, we simply let $\pi'(\cdot|s) := \pi(\cdot|\nu^*(s))$ and the proof is complete. □

Before proving Theorem 6 we first give a technical lemma about the total variation distance between two multi-variate Gaussian distributions with the same variance.

Lemma 2. *Given two multi-variate Gaussian distributions* $X_1 \sim \mathcal{N}(\mu_1, \sigma^2 I_n)$ *and* $X_2 \sim \mathcal{N}(\mu_2, \sigma^2 I_n)$*,* $\mu_1, \mu_2 \in \mathbb{R}^n$*, define* $d = \|\mu_2 - \mu_1\|_2$*. We have* $\mathrm{D}_{TV}(X_1, X_2) = \sqrt{\frac{2}{\pi}}\frac{d}{\sigma} + O(d^3)$.

Proof. Denote probability density of X_1 and X_2 as f_1 and f_2, and denote $a = \frac{\mu_2 - \mu_1}{d}$ as the normal vector of the perpendicular bisector line between μ_1 and μ_2. Due to the symmetry of Gaussian distribution, $f_1(x) - f_2(x)$ is positive for all x where $a^\top x - a^\top \mu_1 - \frac{d}{2} > 0$ and negative for all x on the other symmetric side. When $a^\top x - a^\top \mu_1 - \frac{d}{2} > 0$, $\int_{x \in \mathbb{R}^n} [f_1(x) - f_2(x)] \mathrm{d}x = \Phi(\frac{d}{2\sigma}) - (1 - \Phi(\frac{d}{2\sigma})) = 2\Phi(\frac{d}{2\sigma}) - 1$. Thus,

$$\begin{aligned} D_{TV}(X_1, X_2) &= \int_{x \in \mathbb{R}^n} |f_1(x) - f_2(x)| \mathrm{d}x \\ &= 2 \int_{a^\top x - a^\top \mu_1 - \frac{d}{2} > 0} (f_1(x) - f_2(x)) \mathrm{d}x \\ &= 2(\Phi(\frac{d}{2\sigma}) - (1 - \Phi(\frac{d}{2\sigma}))) \\ &= 2(2\Phi(\frac{d}{2\sigma}) - 1) \end{aligned}$$

Then we use the Taylor series for $\Phi(x)$ at $x = 0$:

$$\Phi(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{2^n n!(2n+1)}$$

Since we consider the case where d is small, we only keep the first order term and obtain:

$$D_{TV}(X_1, X_2) = \sqrt{\frac{2}{\pi}}\frac{d}{\sigma} + O(d^3)$$

□

Theorem 6. $\mathrm{D}_{TV}(\bar{\pi}(\cdot|s), \bar{\pi}(\cdot|\hat{s})) = \sqrt{2/\pi}\frac{d}{\sigma} + O(d^3)$*, where* $d = \|\pi(s) - \pi(\hat{s})\|_2$.

Proof. This theorem is a special case of Lemma 2 where $X_1 = \bar{\pi}(\cdot|s)$, $X_2 = \bar{\pi}(\cdot|s')$ and $X_1 \sim \mathcal{N}(\pi(s), \sigma^2 I)$, $X_2 \sim \mathcal{N}(\pi(s'), \sigma^2 I)$. □

C Optimization Techniques

C.1 More Backgrounds for Convex Relaxation of Neural Networks

In our work, we frequently need to solve a minimax problem:

$$\min_{\theta} \max_{\phi \in \mathbb{S}} g(\theta, \phi) \tag{27}$$

One approach we will discuss is to first solve the inner maximization problem (approximately) using an optimizer like SGLD. However, due to the non-convexity of π_θ, we cannot solve the inner maximization to global maxima, and the gap between local maxima and global maxima can be large. Using convex relaxations of neural networks, we can instead find an upper bound of $\max_{\phi \in \mathbb{S}} g(\theta, \phi)$:

$$\overline{g}(\theta) \geq \max_{\phi \in \mathbb{S}} g(\theta, \phi)$$

Thus we can minimize an upper bound instead, which can guarantee the original objective (27) is minimized.

As an illustration on how to find $\overline{g}(\theta)$ using convex relaxations, following Salman et al. [58] we consider a simple L-layer MLP network $f(\theta, x)$ with parameters $\theta = \{(W^{(i)}, b^{(i)}), i \in \{1, \cdots, L\}\}$ and activation function σ. We denote $x^{(0)} = x$ as the input, $x^{(i)}$ as the post-activation value for layer i, $z^{(i)}$ as the pre-activation value for layer i. $i \in \{1, \cdots, L\}$. The output of the network $f(\theta, x)$ is $z^{(L)}$. Then, we consider the following optimization problem:

$$\max_{x \in \mathbb{S}} f(\theta, x), \quad \text{where } \mathbb{S} \text{ is the set of perturbations}$$

which is equivalent to the following optimization problem:

$$\begin{aligned} \max \quad & z^{(L)} \\ \text{s.t.} \quad & z^{(l)} = W^{(l)} x^{(l-1)} + b^{(l)}, l \in [L], \\ & x^{(l)} = \sigma(z^{(l)}), l \in [L-1], \\ & x^{(0)} \in \mathbb{S} \end{aligned} \tag{28}$$

In this constrained optimization problem (28), assuming $\mathbb{S}$ is a convex set, the constraint on $z^{(l)}$ is convex (linear) and the only non-convex constraints are those for $x^{(l)}, l = \{1, \cdots, L-1\}$, where a non-linear activation function is involved. Note that activation function $\sigma(z)$ itself can be a convex function, but when used as an equality constraint, the feasible solution is constrained to the *graph* of $\sigma(z)$, which is non-convex.

Previous works [80, 87, 58] propose to use convex relaxations of non-linear units to relax the non-convex constraint $x^{(l)} = \sigma(z^{(l)})$ with a convex one, $x^{(l)} = \text{convex}(\sigma(z^{(l)}))$, such that (28) can be solved efficiently. We can then obtain an *upper bound* of $f(\theta, x)$ since the constraints are relaxed.

Zhang et al. [87] gave several concrete examples (e.g., ReLU, tanh, sigmoid) on how these relaxations are formed. In the special case where linear relaxations are used, (28) can be solved efficiently and automatically (without manual derivation and implementation) for general computational graphs [84]. Generally, using the framework from Xu et al. [84] we can access an oracle function ConvexRelaxUB defined as below:

Definition 2. *Given a neural network function $f(\mathbf{X})$ where $\mathbf{X}$ is any input for this function, and $\mathbf{X} \in \mathbb{S}$ where $\mathbb{S}$ is the set of perturbations, the oracle function* ConvexRelaxUB *provided by an automatic neural network convex relaxation tool returns an upper bound $\overline{f}$, which satisfies:*

$$\overline{f} \geq \max_{\mathbf{X} \in \mathbb{S}} f(\mathbf{X})$$

Note that in the above definition, $\mathbf{X}$ can by *any* input for this computation (e.g., $\mathbf{X}$ can be s, a, or θ for a $Q_\theta(s, a)$ function). In the special case of our paper, for simplicity we define the notation ConvexRelaxUB$(f, \theta, s \in B(s))$ which returns an upper bound function $\overline{f}(\theta)$ for $\max_{s \in B(s)} f(\theta, s)$.

Computational cost Many kinds of convex relaxation based methods exist [58], where the expensive ones (which give a tighter upper bound) can be a few magnitudes slower than forward propagation. The cheapest method is interval bound propagation (IBP), which only incurs twice more costs as forward propagation; however, IBP base training has been reported unstable and hard to reproduce as its bounds are very loose [89, 3]. To avoid potential issues with IBP, in all our environments, we use the IBP+Backward relaxation scheme following [89, 84], which produces considerably tighter bounds, while being only a few times slower than forward propagation (e.g., 3 times slower than forward propagation when loss fusion [84] is implemented). In fact, Xu et al. [84] used the same relaxation for training downscaled ImageNet dataset on very large vision models. For DRL the policy neural networks are typically small and can be handled quite efficiently. In our paper, we use convex relaxation as a blackbox tool (provided by the `auto_LiRPA` library [84]), and any new development for improving its efficiency can benefit us.

C.2 Solving the Robust Policy Regularizer using SGLD

Stochastic gradient Langevin dynamics (SGLD) [18] can escape saddle points and shallow local optima in non-convex optimization problems [54, 90, 10, 85], and can be used to solve the inner

maximization with zero gradient at $\hat{s} = s$. SGLD uses the following update rule to find $\hat{s}^K$ to maximize $\mathcal{R}_s(\hat{s}, \theta_\mu)$:

$$\hat{s}^{k+1} \leftarrow \text{proj}\left(\hat{s}^k - \eta_k \nabla_{\hat{s}^k} \mathcal{R}_s(\hat{s}^k, \theta_\mu) + \sqrt{2\eta_k/\beta_k}\xi\right), \quad \hat{s}^0 = s, \quad k = 0, \cdots, K-1$$

where η_k is step size, ξ is an i.i.d. standard Gaussian random variable in $\mathbb{R}^{|\mathcal{S}|}$, β_k is an inverse temperature hyperparameter, and $\text{proj}(\cdot)$ projects the update back into $B(s)$. We find that SGLD is sufficient to escape the stationary point at $\hat{s} = s$. However, due to the non-convexity of $\mu_{\theta_\mu}(\hat{s}, \theta_\mu)$, this approach only provides a lower bound $\mathcal{R}_s(\hat{s}^K, \theta_\mu)$ of $\max_{\hat{s} \in B(s)} \mathcal{R}_s(\hat{s}, \theta_\mu)$. Unlike the convex relaxation based approach, minimizing this lower bound does not guarantee to minimize (5), as the gap between $\max_{\hat{s} \in B(s)} \mathcal{R}_s(\hat{s}, \theta_\mu)$ and $\mathcal{R}_s(\hat{s}^K, \theta_\mu)$ can be large.

Computational Cost In SGLD, we first need to solve the inner maximization problem (such as Eq. (5)). The additional time cost depends on the number of SGLD steps. In our experiments for PPO and DDPG, we find that using 10 steps are sufficient. However, the total training cost does not grow by 10 times, as in many environments the majority of time was spent on environment simulation steps, rather than optimizing a small policy network.

D Additional details for adversarial attacks on state observations

D.1 More details on the Critic based attack

In Section 3.5 we discuss the critic based attack [50] as a baseline. This attack requires a Q function $Q(s, a)$ to find the best perturbed state. In Algorithm 2 we present our "corrected" critic based attack based on [50]:

Algorithm 2 Critic based attack [50]

Input: A policy function π under attack, a corresponding $Q(s, a)$ network, and a initial state s^0, K is the number of attack steps, η is the step size, $\underline{s}$ and $\overline{s}$ are valid lower and upper range of s (assuming a ℓ_∞ norm-like threat model).

for $k = 1$ to K **do**

$g^k = \nabla_{s^{k-1}} Q(s_0, \pi(s^{k-1})) = \frac{\partial Q}{\partial \pi} \frac{\partial \pi}{\partial s^{k-1}}$

$g^k \leftarrow \text{proj}(g^k)$ ▷project g^k according to norm constraint of s; for ℓ_∞ norm simply take the sign

$s^k \leftarrow s^{k-1} - \eta g^k$

$s^k \leftarrow \min(\max(s^k, \underline{s}), \overline{s})$ ▷only needed for ℓ_∞ norm threat model

end for

Output: An adversarial state $\hat{s} := s^K$

Note that in Algorithm 4 of [50], given a state s^0 under attack, they use the gradient $\nabla_s Q(s, \pi(s)) = \frac{\partial Q}{\partial s} + \frac{\partial Q}{\partial \pi} \frac{\partial \pi}{\partial s}$ which essentially attempts to minimize $Q(\hat{s}, \pi(\hat{s}))$, but they then sample randomly along this gradient direction to find the best $\hat{s}$ that minimizes $Q(s^0, \pi(\hat{s}))$. Our corrected formulation directly minimizes $Q(s^0, \pi(\hat{s}))$ using this gradient instead $\nabla_s Q(s^0, \pi(s)) = \frac{\partial Q}{\partial \pi} \frac{\partial \pi}{\partial s}$.

For PPO, since there is no $Q(s, a)$ available during training, we extend [50] to perform attack relying on $V(s)$: we find a state $\hat{s}$ that minimizes $V(\hat{s})$. Unfortunately, it does not match our setting of perturbing state observations; it looks for a state $\hat{s}$ that has the worst value (i.e., taking action $\pi(\hat{s})$ in state $\hat{s}$ is bad), but taking the action $\pi(\hat{s})$ at state s^0 does not necessarily trigger a low reward action, because $V(\hat{s}) = \max_a Q(\hat{s}, a) \neq \max_a Q(s^0, a)$. Thus, in Table 1 we can observe that critic based attack typically does not work very well for PPO agents.

D.2 More details on the Maximal Action Difference (MAD) attack

We present the full algorithm of MAD attack in Algorithm 3. It is a relatively simple attack by directly maximizing a KL-divergence using SGLD, yet it usually outperforms random attack and critic attack on many environments (e.g., see Figure 10).

Algorithm 3 Maximal Action Difference (MAD) Attack (a critic-independent attack)

Input: A policy function π under attack, and a initial state s_0, T is the number of attack steps, η is the step size, β is the (inverse) temperature parameter for SGLD, $\underline{s}$ and $\overline{s}$ are valid lower and upper range of s.
Define loss function $L_{\text{MAD}}(s) = -D_{\text{KL}}(\pi(\cdot|s_0)\|\pi(\cdot|s))$
for $t = 1$ to T **do**
Sample $\xi \sim \mathcal{N}(0, 1)$
$g_t = \nabla L_{\text{MAD}}(s_{t-1}) + \sqrt{\frac{2}{\beta\eta}}\xi$
$g_t \leftarrow \text{proj}(g_t)$ ▷project g_t according to norm constraint of s; for ℓ_∞ norm simply take the sign
$s_t \leftarrow s_{t-1} - \eta g_t$
$s_t \leftarrow \min(\max(s_t, \underline{s}), \overline{s})$
end for
Output: An adversarial state $\hat{s} := s_T$

D.3 More details on the Robust Sarsa attack

Algorithm 4 gives the full procedure of the Robust Sarsa attack. We collect trajectories of the agents and then optimize the ordinary temporal difference (TD) loss along with a robust objective $L_{\text{robust}}(\theta)$. $L_{\text{robust}}(\theta)$ constrains that when an input action a is slightly changed, the value $Q^\pi_{\text{RS}}(s, a)$ should not change significantly. We set the perturbation set $B_p(a, \epsilon)$ to be a ℓ_p norm ball with radius ϵ around an action a. We gradually increase ϵ from 0 to ϵ_{max} during training to learn a critic that is increasingly more robust. The inner maximization of $L_{\text{robust}}(\theta)$ is upper bounded by convex relaxations of neural networks, which we introduced in section C.1. Once the inner maximization is eliminated, we solve the final objective using regular first order optimization methods. In our attacks to DDPG and PPO, we try multiple regularization parameter λ_{RS} to find the best Sarsa model that achieves *lowest* attack rewards.

Algorithm 4 Train a robust value function for critic-independent attack (Robust Sarsa attack)

Input: Any policy function π under attack, T is the number of training steps, and an epsilon schedule ϵ_t
Initialize $Q^\pi_{\text{RS}}(s, a)$ to be a random network
for $t = 1$ to T **do**
Run the agent with policy π and collect a batch of N steps: $\{s_i, a_i, r_i, s'_i, a'_i\}, i \in [N]$
$L_{\text{TD}}(\theta) = \sum_{i\in[N]} \left[r_i + \gamma Q^\pi_{\text{RS}}(s'_i, a'_i) - Q^\pi_{\text{RS}}(s_i, a_i)\right]^2$
$L_{\text{robust}}(\theta) = \sum_{i\in[N]} \max_{\hat{a}\in B_p(a_i,\epsilon_t)} (Q^\pi_{\text{RS}}(s_i, \hat{a}) - Q^\pi_{\text{RS}}(s_i, a_i))^2$
$\overline{L}_{\text{robust}} = \text{ConvexRelaxUB}(L_{\text{robust}}, \theta, B_p(a_i, \epsilon_t))$, where $L_{\text{robust}}(\theta) \le \overline{L}_{\text{robust}}(\theta)$ ▷Solving the inner maximization by upper bounding L_{robust} using an automatic NN convex relaxation tool
Minimize $L_{\text{RS}}(\theta) = L_{\text{TD}}(\theta) + \lambda_{\text{RS}}\overline{L}_{\text{robust}}(\theta)$ using any gradient based optimizer (e.g., Adam)
end for
Output: A robust critic function Q^π_{RS} that can be used for Algorithm 2.

Although it is beyond the scope of this paper, RS attack can also be used as a blackbox attack when perturbing the actions rather than state observations, as $Q^\pi_{\theta_{RS}}$ can be learned by observing the environment and the agent without any internal information of the agent. Then, using the robust critic we learned, black-box attacks can be performed on action space by solving $\min Q^\pi_{\theta_{RS}}(s, a)$ with a norm constrained a.

For a practical implementation, to improve convergence and reduce instability, two $Q^\pi_{\text{RS}}(s, a)$ functions can be also used similarly as in double Q learning [23]. In our case, since the policy is not being updated and stable, we find that using a single Q function is also sufficient for most settings and usually converges faster.

We provide some empirical justifications for the necessity of using a robust objective. For both PPO and DDPG, we conduct attacks using a Sarsa network trained with and without the robustness objective, in Table 4 and Table 5, respectively. We observe that the robust objective can decrease reward further more in most settings.

Table 4: Comparison between Non-robust Sarsa attack (without the robustness objective $L_{\text{robust}}(\theta)$) and robust Sarsa attack on PPO and SA-PPO agents in Table 1. The Robust Sarsa Attack Reward column is the same result presented in RS column of Table 1. We report mean reward $\pm$ standard deviation over 50 attack episodes.

Env.	ℓ_∞ norm perturbation budget ϵ	Method	Non-robust Sarsa Attack Reward	Robust Sarsa Attack Reward
Hopper	0.05	PPO (vanilla)	2757.0±604.2	**779.4±33.2**
		PPO (adv. 50%)	276 ±140	**49 ± 50**
		PPO (adv. 100%)	14.4± 4.20	**3.8 ± 0.9**
		SA-PPO (SGLD)	3642.9±4.0	**1403.3±55.0**
		SA-PPO (Convex)	3014.9±656.1	**1235.8±50.2**
Walker2d	0.05	PPO (vanilla)	2224.7±1438.7	**913.7±54.3**
		PPO (adv. 50%)	-10.79 ± 0.93	**-11.55 ± 0.79**
		PPO (adv. 100%)	-111.9± 4.5	**-114.4 ± 4.0**
		SA-PPO (SGLD)	4777.1±305.5	**2605.6±1255.7**
		SA-PPO (Convex)	3701.1±1013.3	**2168.2± 665.4**
Humanoid	0.075	PPO (vanilla)	**716.4±166.1**	1036.0±420.2
		PPO (adv. 50%)	166± 78	**98 ± 69**
		PPO (adv. 100%)	122.6± 15.9	**113.2 ± 18.5**
		SA-PPO (SGLD)	6115.4±783.2	**6200.5±818.1**
		SA-PPO (Convex)	6241.2±540.8	**4707.2±1359.1**

Table 5: Comparison between Non-robust Sarsa attack (without the robustness objective) and robust Sarsa attack on DDPG and SA-DDPG agents in Table 2. The Robust Sarsa Attack Reward column presents the same results as presented in the RS attack rows of Table 6. We report mean reward $\pm$ standard deviation over 50 attack episodes.

Env.	ℓ_∞ norm perturbation budget ϵ	Method	Non-robust Sarsa Attack Reward	Robust Sarsa Attack Reward
Ant	0.2	DDPG (vanilla)	700 ± 305	$\mathbf{336 \pm 283}$
		SA-DDPG (Convex)	2380 ± 142	$\mathbf{1820 \pm 635}$
Hopper	0.075	DDPG (vanilla)	1362 ± 1468	$\mathbf{606 \pm 124}$
		SA-DDPG (Convex)	1323 ± 491	$\mathbf{1258 \pm 561}$
InvertedPendulum	0.3	DDPG (vanilla)	1000 ± 0	$\mathbf{92 \pm 1}$
		SA-DDPG (Convex)	1000 ± 0	1000 ± 0
Reacher	1.5	DDPG (vanilla)	-24.11 ± 7.19	$\mathbf{-21.74 \pm 5.14}$
		SA-DDPG (Convex)	$\mathbf{-11.67 \pm 3.57}$	-11.40 ± 3.56
Walker2d	0.05	DDPG (vanilla)	951 ± 1146	959 ± 1001
		SA-DDPG (Convex)	3200 ± 1939	$\mathbf{1986 \pm 1993}$

D.4 Hybrid RS+MAD attack

We find that RS and MAD attack can achieve the best results (lowest attack reward) in many cases. We also consider combining them to form a hybrid attack, which minimizes the robust critic predicted value and in the meanwhile maximizes action differences. It can be conducted by minimizing this combined loss function to find an adversarial state $\hat{s} \in B(s)$:

$$L_{\text{Hybrid}}(\hat{s}) = \alpha_{\text{RS-MAD}} Q_{\theta_Q}(s, \pi_{\theta_{RS}}(\hat{s})) + (1 - \alpha_{\text{RS-MAD}}) L_{\text{MAD}}(\hat{s})$$

For a practical implementation, it is important to choose $\alpha_{\text{RS-MAD}}$ so that the two parts of the loss are roughly balanced. The value of Q_{θ_Q} depends on environment reward (if reward is not normalized), and might be much larger in magnitudes than RS-MAD, so typically $\alpha_{\text{RS-MAD}}$ is close to 1.

We try different values of $\alpha_{\text{RS-MAD}}$ and report the lowest reward as the final reward under this attack.

D.5 Projected Gradient Decent (PGD) Attack for DQN

For DQN, we use the regular untargeted Projected Gradient Decent (PGD) attack in the literature [36, 50, 82]. The untargeted PGD attack with K iterations updates the state K times as follows:

$$\begin{aligned} s^{k+1} &= s^k + \eta \text{proj}[\nabla_{s^k} \mathcal{H}(Q_\theta(s^k, \cdot), a^*)], \\ s^0 &= s, \quad k = 0, \ldots, K-1 \end{aligned} \tag{29}$$

where $\mathcal{H}(Q_\theta(s^k, \cdot), a^*)$ is the cross-entropy loss between the output logits of $Q_\theta(s^k, \cdot)$ and the onehot-encoded distribution of $a^* := \arg\max_a Q_\theta(s, a)$. proj$[\cdot]$ is a projection operator depending on the norm constraint of $B(s)$ and η is the learning rate. A successful untargeted PGD attack will then perturb the state to lead the Q network to output an action other than the optimal action a^* chosen at the original state s. To guarantee that the final state obtained by the attack is within an ℓ_∞ ball around s ($B_\epsilon(s) = \{\hat{s} : s - \epsilon \le \hat{s} \le s + \epsilon\}$), the projection proj$[\cdot]$ is a sign operator and η is typically set to $\eta = \frac{\epsilon}{K}$.

E Robustness Certificates for Deep Reinforcement Learning

If we use the convex relaxation in Section C.1 to train our networks, it can produce robustness certificates [80, 44, 89] for our task. However in some RL tasks the certificates have interpretations different from classification tasks, as discussed in detail below.

Robustness Certificates for DQN. In DQN, the action space is finite, so we have a robustness certificate on the actions taken at each state. More specifically, at each state s, policy π's action is certified if its corresponding Q function satisfies

$$\arg\max_a Q_\theta(s,a) = \arg\max_a Q_\theta(\hat{s},a) = a^*, \text{for all } \hat{s} \in B(s). \tag{30}$$

Given a states s, we can use neural network convex relaxations to compute an upper bound $u_{Q_\theta,a^*,a}(s)$ such that

$$Q_\theta(\hat{s},a) - Q_\theta(\hat{s},a^*) \leq u_{Q_\theta,a^*,a}(s)$$

holds for all $\hat{s} \in B(s)$. So if $u_{Q_\theta,a^*,a}(s) \leq 0$ for all $a \in \mathcal{A}$, we have

$$Q_\theta(\hat{s},a) - Q_\theta(\hat{s},a^*) \leq 0 \tag{31}$$

is guaranteed for all $\hat{s} \in B(s)$, which means that the agent's action will not change when the state observation is in $B(s)$. When the agent's action is not changed under an adversarial perturbation, its reward and transition at current step will not change in the DQN setting, either.

In some settings, we find that 100% of the actions are guaranteed to be unchanged (e.g., the Pong environment in Table 3). In that case, we can in fact also certify that the accumulated reward is not changed given the specific initial conditions for testing. Otherwise, if some steps during the roll-out do not have this certificate, or have a weaker certificate that more than one actions are possible given $\hat{s} \in B(s)$, all the possible actions have to be explored as the next action input to the environment. When there are n states which are not certified to have unchanged actions, each with m possible actions, we need to run n^m trajectories to find the worst case cumulative reward. This is impractical for typical settings.

However, even in the 100% certificate rate setting like Pong, it can still be challenging to certify that the agent is robust under *any* starting condition. Since the agent is started with a random initialization, it is impractical to enumerate all possible initializations and guarantee all generated trajectories are certified. Similarly, in the classification setting, many existing certified defenses [81, 44, 19, 89] can only practically guarantee robustness on a specific test set (by computing a "verified test error"), rather than on *any* input image.

Robustness Certificates for PPO and DDPG. In DDPG and PPO, the action space is continuous, hence it is not possible to certify that actions do not change under adversary. We instead seek for a different type of guarantee, where we can upper bound the change in action given a norm bounded input perturbation:

$$U_s \geq \max_{\hat{s} \in B(s)} \|\pi_{\theta_\pi}(\hat{s}) - \pi_{\theta_\pi}(s)\| \tag{32}$$

Given a state s, we can use convex relaxations to compute an upper bound U_s. Generally speaking, if $B(s)$ is small, a robust policy desires to have a small U_s, otherwise it can be possible to find an adversarial state perturbation that greatly changes $\pi_{\theta_\pi}(\hat{s})$ and causes the agent to misbehave. However, giving certificates on cumulative rewards is still challenging, as it requires to bound reward $r(s,a)$ given a fixed state s, and a perturbed and bounded action a (bounded via (32)). Since the environment dynamics can be quite complex in practice (except for the simplest environment like InvertedPendulum), it is hard to bound reward changes given a bounded action. We leave this part as a future direction for exploration and we believe the robustness certificates in (32) can be useful for future works.

F Additional details for SA-PPO

Algorithm We present the full SA-PPO algorithm in Algorithm 5. Compared to vanilla PPO, we add a robust state-adversarial regularizer which constrains the KL divergence on state perturbations. We highlighted these changes in Algorithm 5. The regularizer $\mathcal{R}_{\text{PPO}}(\theta_\pi)$ can be solved using SGLD or convex relaxations of neural networks. We define the perturbation set $B(s)$ to be an ℓ_p norm ball around state s with radius ϵ: $B_p(s,\epsilon) := \{s' | \|s'-s\|_p \leq \epsilon\}$. We use a ϵ-schedule during training, where the perturbation budget is slowly increasing dduring each epoch t as ϵ_t until reaching ϵ.

Algorithm 5 State-Adversarial Proximal Policy Optimization (SA-PPO). We highlight its differences compared to vanilla PPO in brown.

Input: Number of iterations T, a ϵ schedule ϵ_t

1: Initialize actor network $\pi(a|s)$ and critic network $V(s)$ with parameter θ_π and θ_V,

2: **for** $t = 1$ to T **do**

3: Run π_{θ_π} to collect a set of trajectories $\mathcal{D} = \{\tau_k\}$ containing $|\mathcal{D}|$ episodes, each τ_k is a trajectory contain $|\tau_k|$ samples, $\tau_k := \{(s_{k,i}, a_{k,i}, r_{k,i}, s_{k,i+1})\}, i \in [|\tau_k|]$

4: Compute cumulative reward $\hat{R}_{k,i}$ for each step i in every episode k using the trajectories and discount factor γ

5: Update value function by minimizing the mean-square error:

$$\theta_V \leftarrow \arg\min_{\theta_V} \frac{1}{\sum_k |\tau_k|} \sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \left(V(s_{k,i}) - \hat{R}_{k,i} \right)^2$$

6: Estimate advantage $\hat{A}_{k,i}$ for each step i in every episode k using generalized advantage estimation (GAE) and value function $V_{\theta_V}(s)$

7: Define the state-adversarial policy regularier:

$$\mathcal{R}_{\text{PPO}}(\theta_\pi) := \sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \max_{\bar{s}_{k,i} \in B_p(s_{k,i}, \epsilon_t)} \mathrm{D}_{\mathrm{KL}} \left(\pi(a|s_{k,i}) \| \pi(a|\bar{s}_{k,i}) \right)$$

8: Option 1: Solve $\mathcal{R}_{\text{PPO}}(\theta_\pi)$ using SGLD:

9: find $\hat{s}_{k,i} = \arg\max_{\bar{s}_{k,i} \in B_p(s_{k,i}, \epsilon_t)} \mathrm{D}_{\mathrm{KL}}(\pi(a|s_{k,i}) \| \pi(a|\bar{s}_{k,i}))$ using SGLD optimization for all k, i (the objective can be solved in a batch)

10: set $\overline{\mathcal{R}}_{\text{PPO}}(\theta_\pi) := \sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \mathrm{D}_{\mathrm{KL}}(\pi(a|s_{k,i}) \| \pi(a|\hat{s}_{k,i}))$

11: Option 2: Solve $\mathcal{R}_{\text{PPO}}(\theta_\pi)$ using convex relaxations:

12: $\overline{\mathcal{R}}_{\text{PPO}}(\theta_\pi) := \text{ConvexRelaxUB}(\mathcal{R}_{\text{PPO}}, \theta_\pi, \bar{s}_{k,i} \in B_p(s_{k,i}, \epsilon_t))$

13: Update the policy by minimizing the SA-PPO objective (the minimization is solved using ADAM):

$$\theta_\pi \leftarrow \arg\min_{\theta'_\pi} \frac{1}{\sum_k |\tau_k|} \left[\sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \min \left(r_{\theta'_\pi}(a_{k,i}|s_{k,i}) \hat{A}_{k,i}, g(r_{\theta'_\pi}(a_{k,i}|s_{k,i})) \hat{A}_{k,i} \right) + \kappa_{\text{PPO}} \overline{\mathcal{R}}_{\text{PPO}}(\theta'_\pi) \right]$$

where $r_{\theta'_\pi}(a_{k,i}|s_{k,i}) := \frac{\pi_{\theta'_\pi}(a_{k,i}|s_{k,i})}{\pi_{\theta_\pi}(a_{k,i}|s_{k,i})}$, $g(r) := \text{clip}(r_{\theta'_\pi}(a_{k,i}|s_{k,i}), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}})$

14: **end for**

Hyperparameters for Regular PPO Training We use the optimal hyperparameters in [14] which were found using a grid search for vanilla PPO. However, we found that their parameters are not optimal for Humanoid and achieves a cumulative reward of only about 2000 after 1×10^7 steps. Thus we redo hyperparameter search on Humanoid and change learning rate for actor to 5×10^{-5} and critic to 1×10^{-5}. This new set of hyperemeters allows us to obtain Humanoid reward about 5000 for vanilla PPO. Note that even under the original, non-optimal set of hyperemeters by [14], our SA-PPO variants still achieve high rewards similarly to those reported in our paper. Our hyperparameter change only significantly improves the performance of vanilla PPO baseline.

We run 2048 simulation steps per iteration, and run policy optimization of 10 epochs with a minibatch size of 64 using Adam optimizer with learning rate 3×10^{-4}, 4×10^{-4} and 5×10^{-5} for Walker, Hopper and Humanoid, respectively. The value network is also trained in 10 epochs per iteration with a minibatch size of 64, using Adam optimizer with learning rate 0.00025, 3×10^{-4}, and 1×10^{-5} for Walker, Hopper and Humanoid environments, respectively (the same as in [14] without further tuning, except for Humanoid as discussed above). Both networks are 3-layer MLPs with $[64, 64]$ hidden neurons. The clipping value ϵ for PPO is 0.2. We clip rewards to $[-10, 10]$ and states to $[-10, 10]$. The discount factor γ for reward is 0.99 and the discount factor used in generalized advantage estimation (GAE) is 0.95. We found that in [14] the agent rewards are still improving when training finishes, thus in our experiments we run the agents longer for better convergence: we run

Walker2d and Hopper 2×10^6 steps (976 iterations) and Humanoid 1×10^7 steps (4882 iterations) to ensure convergence.

Hyperparameter for SA-PPO Training For SA-PPO, we use the same set of hyperparameters as in PPO. Note that the hyperparameters are tuned for PPO but not specifically for SA-PPO. The additional regularization parameter κ_{PPO} for the regularizer $\mathcal{R}_{\text{PPO}}$ is chosen in $\{0.003, 0.01, 0.03, 0.1, 0.3, 1.0\}$. We linearly increase ϵ_t, the norm of ℓ_∞ perturbation on normalized states, from 0 to the target value (ϵ for evaluation, reported in Table 1) during the first $3/4$ iterations, and keep $\epsilon_t = \epsilon$ for the reset iterations. The same ϵ schedule is used for both SGLD and convex relaxation training. For SGLD, we run 10 iterations with step size $\frac{\epsilon_t}{10}$ and set the temperature parameter $\beta = 1 \times 10^{-5}$. For convex relaxations, we use the efficient IBP+Backward scheme [84], and we use a training schedule similar to [89] by mixing the IBP bounds and backward mode perturbation analysis bounds.

G Additional Details for SA-DDPG

Algorithm We present the SA-DDPG training algorithm in Algorithm 6. The main difference between DDPG and SA-DDPG is the additional loss term $\mathcal{R}_{\text{DDPG}}(\theta_\pi)$, which provides an upper bound on $\max_{s \in B(s_i)} \|\pi(s) - \pi(s_i)\|_2^2$. We highlighted these changes in Algorithm 6. We define the perturbation set $B(s)$ to be a ℓ_p norm ball around s with radius ϵ: $B_p(s, \epsilon) := \{s' | \|s' - s\|_p \leq \epsilon\}$. We use a ϵ-schedule during training, where the perturbation budget is slowly increasing during training as ϵ_t until reaching ϵ.

Algorithm 6 State-Adversarial Deep Deterministic Policy Gradient (SA-DDPG). We highlight its differences compared to vanilla DDPG in brown.

Initialize actor network $\pi(s)$ and critic network $Q(s, a)$ with parameter θ_π and θ_Q
Initialize target network $\pi'(s)$ and critic network $Q'(s, a)$ with weights $\theta_{\pi'} \leftarrow \theta_\pi$ and $\theta_{Q'} \leftarrow \theta_Q$
Initial replay buffer $\mathcal{B}$
for $t = 1$ to T **do**
  Initial a random process $\mathcal{N}$ for action exploration
  Choose action $a_t \sim \pi(s_t) + \epsilon$, $\epsilon \sim \mathcal{N}$
  Observe reward r_t, next state s_{t+1} from environment
  Store transition $\{s_t, a_t, r_t, s_{t+1}\}$ into $\mathcal{B}$
  Sample a mini-batch of N samples $\{s_i, a_i, r_i, s_i'\}$ from $\mathcal{B}$
  $y_i \leftarrow r_i + \gamma Q'(s_i', \pi'(s_i'))$ for all $i \in [N]$
  Update θ_Q by minimizing loss $L(\theta_Q) = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i))^2$
  $\mathcal{R}_{\text{DDPG}}(\theta_\pi, \bar{s}_i) := \sum_i \max_{\bar{s}_i \in B_p(s_i, \epsilon_t)} \|\pi_{\theta_\pi}(s_i) - \pi_{\theta_\pi}(\bar{s}_i)\|_2$
  Option 1: Solve $\mathcal{R}_{\text{DDPG}}(\theta_\pi)$ using SGLD:
    find $\hat{s}_i = \arg\max_{\bar{s}_i \in B_p(s_i, \epsilon_t)} \|\pi_{\theta_\pi}(s_i) - \pi_{\theta_\pi}(\bar{s}_i)\|_2$ for all i (solved in a batch using SGLD)
    set $\overline{\mathcal{R}}_{\text{DDPG}}(\theta_\pi) := \sum_i \|\pi_{\theta_\pi}(s_i) - \pi_{\theta_\pi}(\hat{s}_i)\|_2$
  Option 2: Solve $\mathcal{R}_{\text{DDPG}}(\theta_\pi)$ using convex relaxations:
    $\overline{\mathcal{R}}_{\text{DDPG}}(\theta_\pi) := \text{ConvexRelaxUB}(\mathcal{R}_{\text{DDPG}}, \theta_\pi, \bar{s}_i \in B_p(s_i, \epsilon_t))$
  Update θ_π using deterministic policy gradient and gradient of $\overline{\mathcal{R}}_{\text{DDPG}}$:
  $\nabla_{\theta_\pi} J(\theta_\pi) = \frac{1}{N} \sum_i \left[\nabla_a Q(s, a)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta_\pi} \pi(s)|_{s=s_i} + \kappa_{\text{DDPG}} \nabla_{\theta_\pi} \overline{\mathcal{R}}_{\text{DDPG}}\right]$
  Update Target Network:
  $\theta_{Q'} \leftarrow \tau\theta_Q + (1 - \tau)\theta_{Q'}$
  $\theta_{\pi'} \leftarrow \tau\theta_\pi + (1 - \tau)\theta_{\pi'}$
end for

Hyperparameters for Regular DDPG Training. Our hyperparameters are from [62]. Both actor and critic networks are 3-layer MLPs with $[400, 300]$ hidden neurons. We run each environment for 2×10^6 steps. Actor network learning rate is 1×10^{-4} and critic network learning rate is 1×10^{-3} (except that for Hopper-v2 and Ant-v2 the critic learning rate is reduced to 1×10^{-4} due to the larger values of rewards); both networks are optimized using Adam optimizer. No reward scaling is used, and discount factor is set to 0.99. We use a replay buffer with a capacity of 1×10^6 items and we do

not use prioritized replay buffer sampling. For the random process $\mathcal{N}$ used for exploration, we use a Ornstein-Uhlenbeck process with $\theta = 0.15$ and $\sigma = 0.2$. The mixing parameter of current and target actor and critic networks is set to $\tau = 0.001$.

Hyperparameters for SA-DDPG Training. SA-DDPG uses the same hyperparameters as in DDPG training. For the additional regularization parameter κ for $\pi(s)$, we choose in $\{0.1, 0.3, 1.0, 3.0\}$ for InvertedPendulum and Reacher due to their low dimensionality and $\{30, 100, 300, 1000\}$ for other environments.. We train the actor network without state-adversarial regularization for the first 1×10^6 steps, then increase ϵ_t from 0 to the target value in 5×10^5 steps, and then keep training at the target ϵ for 5×10^5 steps. The same ϵ schedule is used for both SGLD and convex relaxation. For SGLD, we run 5 iterations with step size $\frac{\epsilon_t}{5}$ and set the temperature parameter $\beta = 1 \times 10^{-5}$. For convex relaxations, we use the efficient IBP+Backward scheme [84], and a training schedule similar to [89] by mixing the IBP bounds and backward mode perturbation analysis bounds. The total number of training steps is thus 2×10^6, which is the same as the regular DDPG training. The target ϵ values for each task is the same as ϵ listed in Table 2 for evaluation. Note that we apply perturbation on normalized environment states. The normalization factors are the standard deviations calculated using data collected on the baseline policy (vanilla DDPG) without adversaries.

H Additional Details for SA-DQN

Algorithm We present the SA-DQN training algorithm in Algorithm 7. The main difference between SA-DQN and DQN is the additional state-adversarial regularizer $\mathcal{R}_{\text{DQN}}(\theta)$, which encourages the network not to change its output under perturbations on the state observation. We highlighted these changes in Algorithm 7. Note that the use of hinge loss is not required; other loss functions (e.g., cross-entropy loss) may also be used.

Algorithm 7 State-Adversarial Deep Q-Learning (SA-DQN). We highlight its differences compared to vanilla DQN in brown.

Initialize current Q network $Q(s, a)$ with parameters θ.
Initialize target Q network $Q'(s, a)$ with parameters $\theta' \leftarrow \theta$.
Initial replay buffer $\mathcal{B}$
for $t = 1$ to T **do**
  With probability ϵ_t select a random action at a_t, otherwise select $a_t = \arg\max_a Q_\theta(s_t, a; \theta)$
  Execute action a_t in environment and observe reward r_t and state s_{t+1}
  Store transition $\{s_t, a_t, r_t, s_{t+1}\}$ in $\mathcal{B}$.
  Randomly sample a minibatch of N samples $\{s_i, a_i, r_i, s'_i\}$ from $\mathcal{B}$.
  For all s_i, compute $a^*_i = \arg\max_a Q_\theta(s_i, a; \theta)$.
  Set $y_i = r_i + \gamma \max_{a'} Q'_{\theta'}(s'_i, a'; \theta)$ for non-terminal s_i, and $y_i = r_i$ for terminal s_i.
  Compute TD-loss for each transition: TD-$L(s_i, a_i, s'_i; \theta) = \text{Huber}(y_i - Q_\theta(s_i, a_i; \theta))$
  Define $\mathcal{R}_{\text{DQN}}(\theta) := \sum_i \max\left\{ \max_{\hat{s}_i \in B(s)} \max_{a \neq a^*_i} Q_\theta(\hat{s}_i, a; \theta) - Q_\theta(\hat{s}_i, a^*_i; \theta), -c \right\}$.
  Option 1: Use projected gradient descent (PGD) to solve $\mathcal{R}_{\text{DQN}}(\theta)$.
    Run PGD to solve: $\hat{s}_i = \arg\max_{\hat{s}_i \in B(s_i)} \max_{a \neq a^*_i} Q_\theta(\hat{s}_i, a; \theta) - Q_\theta(\hat{s}_i, a^*_i; \theta)$.
    Compute the sum of hinge loss of each s_i:
    $\overline{\mathcal{R}}_{\text{DQN}}(\theta) = \sum_i \max\{\max_{a \neq a^*_i} Q_\theta(\hat{s}_i, a; \theta) - Q_\theta(\hat{s}_i, a^*_i), -c\}$.
  Option 2: Use convex relaxations of neural networks to solve a surrogate loss of $\mathcal{R}_{\text{DQN}}(\theta)$.
    For all s_i and all $a \neq a^*_i$, obtain upper bounds on $Q_\theta(s, a; \theta) - Q_\theta(s, a^*_i; \theta)$:
    $u_{a^*_i, a}(s_i; \theta) = \text{ConvexRelaxUB}(Q_\theta(s, a; \theta) - Q_\theta(s, a^*_i; \theta), \theta, s \in B(s_i))$
    Compute a surrogate loss for the hinge loss:
    $\overline{\mathcal{R}}_{\text{DQN}}(\theta) = \sum_i \max\left\{ \max_{a \neq a^*_i} \{u_{a^*_i, a}(s_i)\}, -c \right\}$
  Perform a gradient descent step to minimize $\frac{1}{N}[\sum_i \text{TD-}L(s_i, a_i, s'_i; \theta) + \kappa_{\text{DQN}} \overline{\mathcal{R}}_{\text{DQN}}(\theta)]$.
  Update Target Network every M steps: $\theta' \leftarrow \theta$.
end for

Hyperparameters for Vanilla DQN training. For Atari games, the deep Q networks have 3 CNN layers followed by 2 fully connected layers (following [78]). The first CNN layer has 32 channels,

a kernel size of 8, and stride 4. The second CNN layer has 64 channels, a kernel size of 4, and stride 2. The third CNN layer has 64 channels, a kernel size of 3, and stride 1. The fully connected layers have 512 hidden neurons for both value and advantage heads. We run each environment for 6×10^6 steps without framestack. We set learning rate as 6.25×10^{-5} (following [26]) for Pong, Freeway and RoadRunner; for BankHeist our implementation cannot reliably converge within 6 million steps, so we reduce learning rate to 1×10^{-5}. For all Atari environments, we clip reward to $-1, +1$ (following [46]) and use a replay buffer with a capacity of 2×10^5.

We set discount factor set to 0.99. Prioritized replay buffer sampling is used with $\alpha = 0.5$ and β increased from 0.4 to 1 linearly through the end of training. A batch size of 32 is used in training. Same as in [46], we choose Huber loss as the TD-loss. We update the target network every 2k steps for all environments.

Hyperparameters for SA-DQN training. SA-DQN uses the same network structure and hyperparameters as in DQN training. The total number of SA-DQN training steps in all environments are the same as those in DQN (6 million). We update the target network every 2k steps for all environments except that the target network is updated every 32k steps for RoadRunner's SA-DQN, which improves convergence for our short training schedule of 6 million frames. For the additional state-adversarial regularization parameter κ for robustness, we choose $\kappa \in \{0.005, 0.01, 0.02\}$. For all 4 Atari environments, we train the Q network without regularization for the first 1.5×10^6 steps, then increase ϵ from 0 to the target value in 4×10^6 steps, and then keep training at the target ϵ for the rest 5×10^5 steps.

Training Time As Atari training is expensive, we train DQN and SA-DQN only 6 million frames; the rewards reported in most DQN paper (e.g., [46, 78, 26]) are obtained by training 20 million frames. Thus, the rewards (without attacks) reported maybe lower than some baselines. The training time for vanilla DQN, SA-DQN (SGLD) and SA-DQN (convex) are roughly 15 hours, 40 hours and 50 hours on a single 1080 Ti GPU, respectively. The training time of each environment varies but is very close.

Note that the training time for convex relaxation based method can be further reduced when using an more efficient relaxation. The fastest relaxation is interval bound propagation (IBP), however it is too inaccurate and can make training unstable and hard to tune [89]. We use the tighter IBP+Backward relaxation, and its complexity can be further improved to the same level as IBP with the recently developed loss fusion technique [84], while providing a much better relaxation than IBP. Our work simply uses convex relaxations as a blackbox tool and we leave further improvements on convex relaxation based methods as a future work.

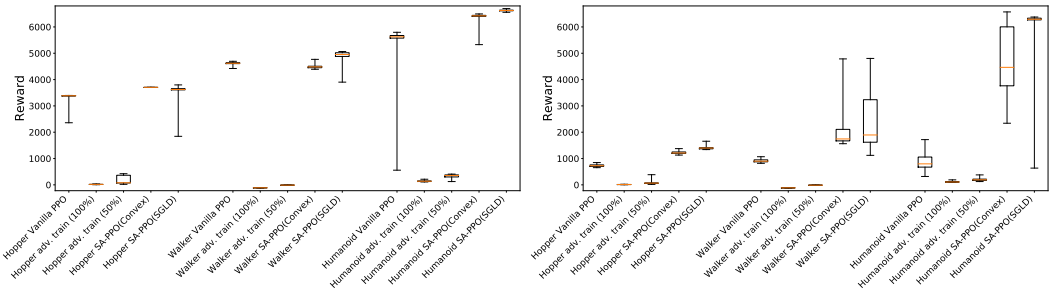
I Additional Experimental Results

I.1 More results on SA-PPO

Box plots of rewards for SA-PPO agents In Table 1, we report the mean and standard deviation of rewards for agents under attack. However, since the distribution of cumulative rewards can be non-Gaussian, in this section we include box plots of rewards for each task in Figure 9. We can observe that the rewards (median, 25% and 75% percentiles) under the strongest attacks (Figure 9b) significantly improve.

Evaluation using multiple ϵ In Figure 10 we show the attack rewards of PPO and SA-PPO agents with different perturbation budget ϵ. We can see that the lowest attack rewards of SA-PPO agents are higher than those of PPO under all ϵ values. Additionally, Robust Sarsa (RS) attacks and RS+MAD attacks are typically stronger than other attacks. On vanilla PPO agents, the MAD attack is also competitive.

Convergence of PPO and SA-PPO agents We want to confirm that our better performing Humanoid agents under state-adversarial regularization are not just by chance. We train each environment using SA-PPO and PPO *at least 15 times*, and collect rewards during training. We plot the median, 25% and 75% percentile of rewards during the training process for all these runs in Figure 11.



(a) Natural episode rewards (no attacks)

(b) Rewards under the best (strongest) attacks

Figure 9: Box plots of natural rewards and rewards under the strongest (best) attacks for PPO, adversarially trained PPO and SA-PPO agents corresponding to the results presented in Table 1 (Table 1 only reports mean and standard deviation). Each box shows the distribution of cumulated rewards collected from 50 episodes of a single agent. The red lines inside the boxes are median rewards, and the upper and lower sides of the boxes show 25% and 75% percentile rewards of 50 episodes. The line segments outside of the boxes show min or max rewards.

We can see that our SA-PPO agents consistently outperform vanilla PPO agents in Humanoid. Since we also present the 25% and 75% percentile of the rewards among 15 agents, we believe this improvement is not because of cherry-picking. For Hopper and Walker environments, SA-PPO has almost no performance drop compared to vanilla PPO.

I.2 More results on SA-DDPG

Reproducibility over multiple training runs. To show that our SA-DDPG can consistently obtain a robust agent and we do not cherry-pick good results, we repeatedly train all 5 environments using SA-DDPG and DDPG **11 times** each and attack all agents. We report the median, minimum, 25% and 75% rewards of 11 agents in box plots. The results are shown in Figure 12. We can observe that SA-DDPG is able to consistently improve the robustness: the median, 25% and 75% percentile rewards under attacks are significantly and consistently better than vanilla DDPG over all 5 environments.

Full attack results In Table 6 we present attack rewards on all of our DDPG agents. In the main text, we only report the strongest (lowest) attack rewards since the lowest reward determines the true agent robustness.

I.3 Robustness Certificates

We report robustness certificates for SA-DQN in Table 3. As discussed in section E, for DQN we can guarantee that an action does not change under bounded adversarial noise. In Table 3, the "Action Cert. Rate" is the ratio of actions that does not change under any ℓ_∞ norm bounded noise. In some settings, we find that 100% of the actions are guaranteed to be unchanged (e.g., the Pong environment in Table 3). In that case, we can in fact also certify that the cumulative reward is not changed given the specific initial conditions for testing.

In SA-DDPG, we can obtain robustness certificates that give bounds on actions in the presence of bounded perturbation on state inputs. Given an input state s, we use convex relaxations of neural networks to obtain the upper and lower bounds for each action: $l_i(s) \leq \pi_i(\hat{s}) \leq u_i(s), \forall \hat{s} \in B(s)$. We consider the following certificates on $\pi(s)$: the average output range $\frac{\|u(s)-l(s)\|_1}{|\mathcal{A}|}$ which reflect the tightness of bounds, and the ℓ_2 distance. Note that bounds on other ℓ_p norms can also be computed given $l_i(s)$ and $u_i(s)$. Since the action space is normalized within $[-1, 1]$, the worst case output range is 2. We report both certificates for all five environments in Table 7. DDPG without our robust regularizer usually cannot obtain non-vacuous certificates (range is close to 2). SA-DDPG can provide robustness certificates (bounded inputs guarantee bounded outputs). We include some discussions on these certificates in Section E.

For SA-PPO, since the action follows a Gaussian policy, we can upper bound its KL-divergence under state perturbations. The results are shown in Table 8. Note that, by increasing the regularization parameter κ, it is possible to obtain an even tighter certificate at the cost of model performance.

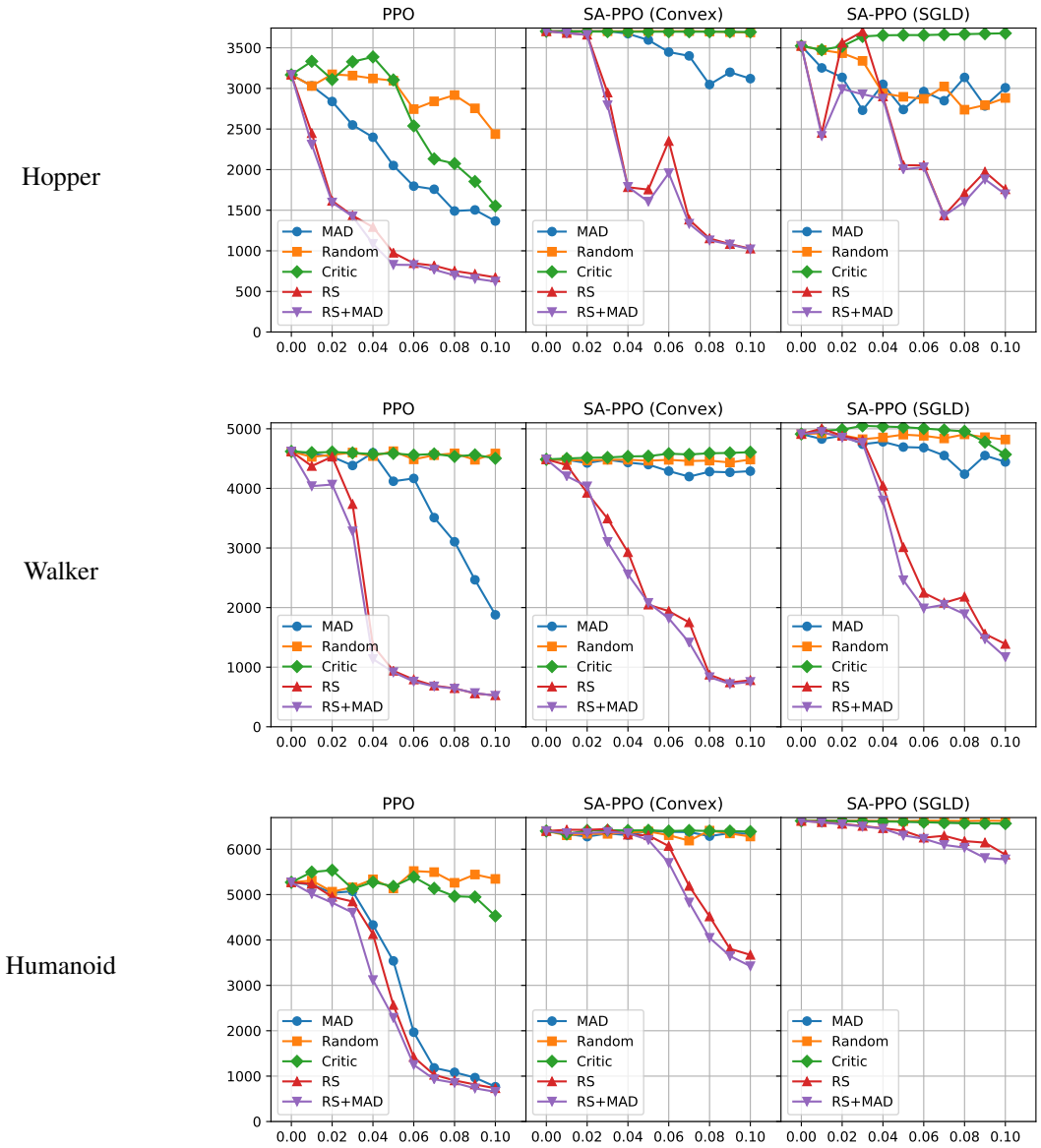
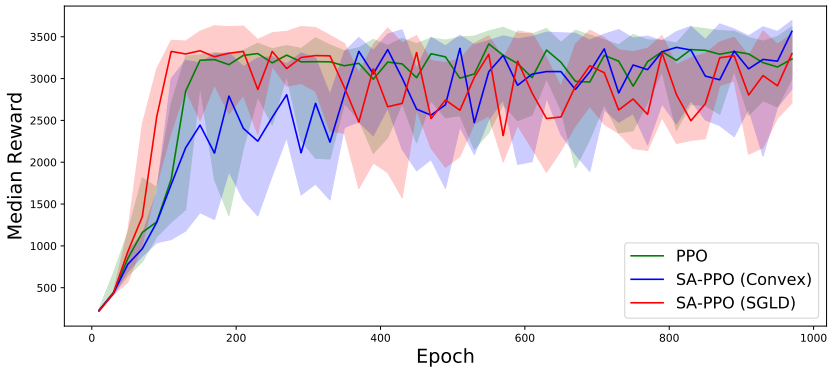
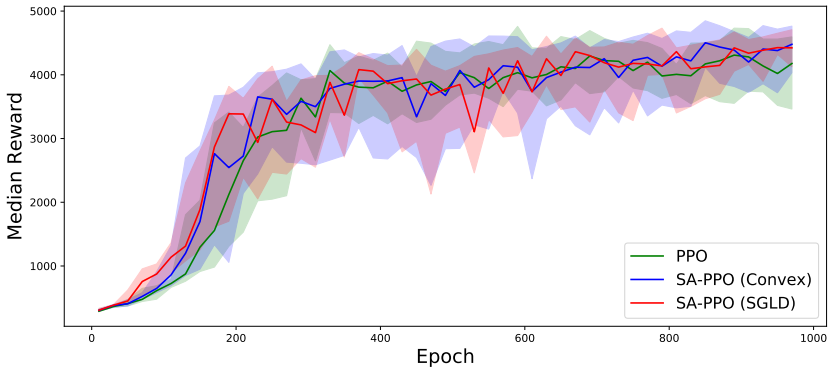


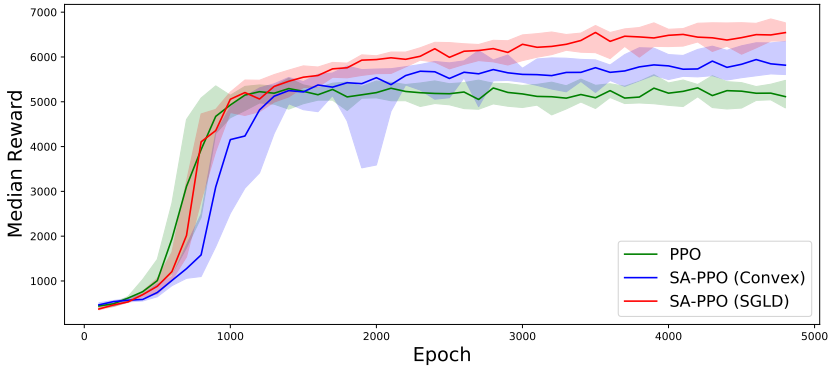
Figure 10: Attacking PPO agents under different ϵ values using 5 attacks. Each data point reported in this figure is an average of 50 episodes.



(a) Hopper

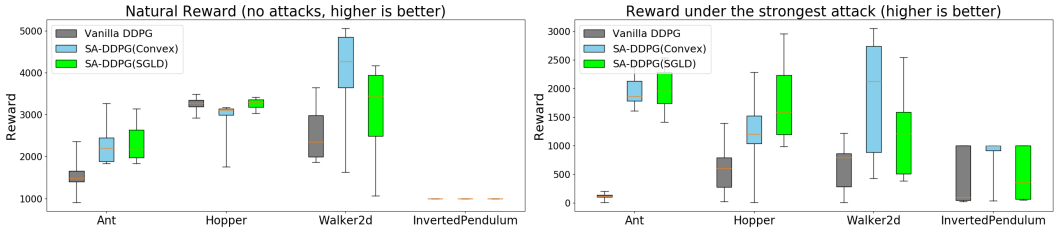


(b) Walker



(c) Humanoid

Figure 11: The median, 25% and 75% percentile episode reward of at least 15 PPO and 15 SA-PPO agents during training. The region of the shaded colors (light blue: SA-PPO solved with SGLD; light green: SA-PPO solved with convex relaxations; light red: vanilla PPO) represent the interval between 25% and 75% percentile rewards over these 15 different training runs, and the solid line is the median rewards over these runs.



(a) Natural episode rewards (no attacks)

(b) Rewards under the best (strongest) attacks

Figure 12: Box plots of natural and attack rewards for DDPG and SA-DDPG. Each box is obtained from **11 agents** trained with the same hyerparameters as the agents reported in Table 2 and tested for 50 episodes (each sample of the box is an average reward over 50 episodes). The red lines inside the boxes are median rewards, and the upper and lower sides of the boxes show 25% and 75% percentile rewards. The line segments outside of the boxes show min or max rewards.

The robustness certificates for SA-DDPG and SA-PPO are computed using interval bound propagation (IBP). For vanilla DDPG and PPO, we use CROWN [87], a much tighter convex relaxation to obtain the certificates, but they are often still vacuous.

Environment		Ant	Hopper	Inverted Pendulum	Reacher	Walker2d
ϵ		0.2	0.075	0.3	1.5	0.05
State Space		111	11	4	11	17
Vanilla DDPG	Natural Reward	1487 ± 850	3302 ± 762	1000 ± 0	-4.37 ± 1.54	1870 ± 1418
	Critic Attack	187 ± 157	2504 ± 1207	1000 ± 0	-24.35 ± 5.10	1301 ± 1229
	Random Attack	1473 ± 795	3086 ± 1006	1000 ± 0	-8.71 ± 2.42	1828 ± 1456
	MAD Attack	180 ± 200	2745 ± 1073	1000 ± 0	-27.67 ± 5.32	1564 ± 1405
	RS Attack	336 ± 283	606 ± 124	92 ± 1	-21.74 ± 5.14	959 ± 1001
	RS+MAD	142 ± 180	2056 ± 1225	1000 ± 0	-27.87 ± 4.38	790 ± 985
	Best Attack	142	606	92	-27.87	790
DDPG with adv. training (50% steps) Pattanaik et al. [50]	Natural Reward	1522 ± 831	2694 ± 497	1000 ± 0	-5.20 ± 1.70	1818 ± 1187
	Critic Attack	222 ± 299	1789 ± 1143	703 ± 373	-23.88 ± 5.05	1391 ± 1083
	Random Attack	1389 ± 785	2316 ± 741	1000 ± 0	-9.09 ± 2.42	1793 ± 955
	MAD Attack	92 ± 240	1497 ± 839	238 ± 240	-25.81 ± 6.53	1680 ± 1106
	RS Attack	129 ± 156	41 ± 105	39 ± 0	-25.45 ± 6.70	837 ± 722
	RS+MAD	31 ± 179	1503 ± 851	116 ± 90	-25.81 ± 6.53	1120 ± 859
	Best Attack	31	41	39	-25.81	837
DDPG with adv. training (100% steps) Pattanaik et al. [50]	Natural Reward	1082 ± 574	973 ± 0	1000 ± 0	-5.71 ± 1.80	462 ± 569
	Critic Attack	126 ± 148	62 ± 34	174 ± 66	-21.91 ± 3.52	809 ± 525
	Random Attack	832 ± 545	577 ± 431	998 ± 5	-9.60 ± 2.56	751 ± 568
	MAD Attack	43 ± 165	56 ± 50	121 ± 19	-26.47 ± 4.19	699 ± 484
	RS Attack	115 ± 286	24 ± 15	82 ± 0	-22.17 ± 4.46	302 ± 260
	RS+MAD	-52 ± 231	56 ± 50	110 ± 26	-27.44 ± 4.05	488 ± 406
	Best Attack	-52	24	82	-27.44	302
SA-DDPG solved by SGLD	Natural Reward	2186 ± 534	3068 ± 223	1000 ± 0	-5 ± 1	3318 ± 680
	Critic Attack	2076 ± 556	2899 ± 439	423 ± 281	-12.10 ± 4.58	1210 ± 979
	Random Attack	2162 ± 524	3071 ± 196	1000 ± 0	-11.41 ± 4.96	3058 ± 848
	MAD Attack	2128 ± 482	3093 ± 17	733 ± 284	-11.94 ± 4.79	3252 ± 689
	RS Attack	2038 ± 401	1729 ± 792	832 ± 328	-11.69 ± 4.80	2224 ± 1050
	RS+MAD	2007 ± 686	1609 ± 676	724 ± 322	-12.01 ± 4.84	1933 ± 1055
	Best Attack	**2007**	**1609**	423	$\mathbf{-12.10}$	1210
SA-DDPG solved by convex relaxations	Natural Reward	2254 ± 430	3128 ± 453	1000 ± 0	-5.24 ± 2.06	4540 ± 1562
	Critic Attack	1826 ± 568	2546 ± 843	1000 ± 0	-11.51 ± 3.80	2245 ± 1881
	Random Attack	2249 ± 491	3036 ± 593	1000 ± 0	-9.87 ± 3.95	4216 ± 1616
	MAD Attack	2106 ± 573	2959 ± 663	1000 ± 0	-12.43 ± 3.76	4135 ± 1884
	RS Attack	1820 ± 635	1258 ± 561	1000 ± 0	-11.40 ± 3.56	1986 ± 1993
	RS+MAD	2005 ± 699	1202 ± 402	1000 ± 0	-12.44 ± 3.77	2315 ± 2127
	Best Attack	1820	1202	1000	-12.44	**1986**

Table 6: Average episode rewards on 5 MuJoCo environments using policies trained by DDPG and SA-DDPG. Natural reward is the reward in clean environment without adversarial attacks. The "Best Attack" rows report the lowest reward over all five attacks (representing the strongest attack), and this lowest reward is used for robustness evaluation.

Table 7: Robustness certificates on bounded action changes under bounded state perturbations for DDPG agents. Results are averaged over 50 episodes. A smaller number is better. A vanilla DDPG agent typically cannot provide non-vacuous robustness guarantees.

Settings		Ant	Hopper	InvertedPendulum	Reacher	Walker2d
Certificates (ℓ_2 upper bound)	SA-DDPG (Convex)	0.181	0.050	0.787	0.202	0.169
	DDPG (vanilla)	3.972	2.612	0.992	1.491	2.484
Certificates (ℓ_1 upper bound)	SA-DDPG (Convex)	0.454	0.074	0.787	0.283	0.301
	DDPG (vanilla)	11.087	4.345	0.992	2.107	4.923
Certificates (ℓ_∞ upper bound)	SA-DDPG (Convex)	0.104	0.041	0.787	0.157	0.131
	DDPG (vanilla)	1.734	1.794	0.992	1.073	1.570
Certificates (Range)	SA-DDPG (Convex)	0.057	0.025	0.787	0.142	0.050
	DDPG (vanilla)	1.386	1.448	0.992	1.054	0.821

Table 8: Upper bound on KL-divergence $\mathrm{D}_{\mathrm{KL}}(\pi(a|s)\|\pi(a|\hat{s}))$ for three PPO environments. A smaller number is better. SA-PPO can reduce this upper bound significantly especially for high dimensional environments like Humanoid.

Settings		Hopper	Walker2d	Humanoid
Certificates (KL upper bound)	SA-PPO (Convex)	0.1232	0.09831	3.529
	PPO (vanilla)	32.16	31.56	925140