Learning Key Steps to Attack Deep Reinforcement Learning Agents

Chien-Min Yu[1], Ming-Hsin Chen[1] and Hsuan-Tien Lin[1*]

[1]Department of Computer Science & Information Engineering, National Taiwan University.

*Corresponding author(s). E-mail(s): htlin@csie.ntu.edu.tw;
Contributing authors: r07922080@csie.ntu.edu.tw;
b08902020@csie.ntu.edu.tw;

Abstract

Deep reinforcement learning agents are vulnerable to adversarial attacks. In particular, recent studies have shown that attacking a few key steps can effectively decrease the agent's cumulative reward. However, all existing attacking methods define those key steps with human-designed heuristics, and it is not clear how more effective key steps can be identified. This paper introduces a novel reinforcement learning framework that learns key steps through interacting with the agent. The proposed framework does not require any human heuristics nor knowledge, and can be flexibly coupled with any white-box or black-box adversarial attack scenarios. Experiments on benchmark Atari games across different scenarios demonstrate that the proposed framework is superior to existing methods for identifying effective key steps. The results highlight the weakness of RL agents even under budgeted attacks.

Keywords: reinforcement learning, adversarial attacks

1 Introduction

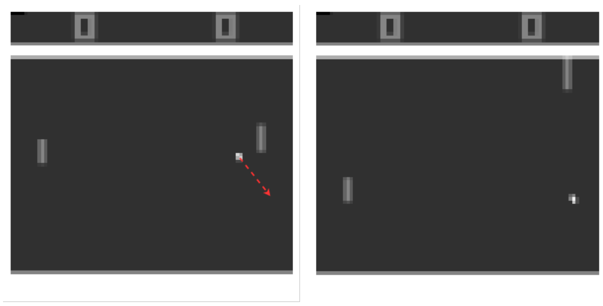
Reinforcement learning (RL) is a framework for sequential decision problems, where an agent interacts with an unknown environment and tries to maximize the total reward it receives. With the rapid development of deep learning, RL agents parametrized by neural networks, usually referred to as deep RL agents, are able to learn complex policies from raw inputs (Mnih et al, 2015). Deep RL agents have shown great success across various domains, such as achieving superhuman performance on games (Mnih et al, 2015; Silver et al, 2016, 2018), completing complex robotic tasks (Levine et al, 2016), optimizing patient treatments (Escandell-Montero et al, 2014; Raghu et al, 2017), and developing autonomous driving skills (Pan et al, 2017; Isele et al, 2018).

However, if we wish to deploy these RL agents into security-critical applications, we must take their reliability into consideration. It is discovered that neural network classifiers may make errors on deliberately crafted inputs, known as adversarial examples (Biggio et al, 2013; Szegedy et al, 2014; Goodfellow et al, 2015). Deep RL is no exception: Many studies have demonstrated that deep RL agents are also vulnerable when attacked by adversarial examples (Behzadan and Munir, 2017a; Huang et al, 2017), raising serious concerns on the reliability of these agents for security-critical applications. For example, there may be severe consequences if an RL-agent-driven autonomous vehicle is compromised by adversarial examples fed from a malicious attacker.

Given the sequential nature of RL, several existing studies argue that it is not necessary to attack at every time step to degrade the agent's performance significantly (Lin et al, 2017; Kos and Song, 2017). The reason is that not all decisions made by the agent are equally important. Some decisions may be critical to the agent, such as those for long-term planning or immediate reward gathering; some other decisions may not have much effect on the environment nor the rewards, and thus attacking those steps would not affect the agent's performance. Lin et al (2017) termed the critical decisions as *strategically-timed* while Kos and Song (2017) termed them as *crucial moments*, based on the corresponding attacking heuristics. In this work, we refer to all such critical steps, which decrease the cumulative reward more effectively when being attacked, as *key steps* for simplicity.

Figure 1 illustrates a key step and a non-key step in Atari Pong, where the main difference between them is whether the ball has already passed through the agent. Attackers aware of this difference can concentrate on attacking those key steps, and save efforts of generating, injecting, and hiding adversarial examples to attack non-key steps. Therefore, an intelligent attacker should prefer attacking key steps over non-key steps. From the perspective of making an RL agent more robust, we should also prefer identifying and understanding key steps as they reveal the Achilles' heel of the agent that needs to be better protected.

A natural question arises: How does an attacker identify the key steps to attack? As an initial attempt to address this question, Lin et al (2017) formulate the problem from an optimization perspective, and propose a heuristic that attacks the agent when it strongly prefers one action over others. Concurrently, Kos and Song (2017) propose another heuristic that attacks when the action appears rewarding to the agent. However, since these heuristics are designed based on human knowledge, it is unclear whether there exist more effective key steps. In this work, we study the possibilities of finding more



(a) Key step (b) Non-key step

Fig. 1: An example of the key step and non-key step in Atari Pong. (a) The ball is moving in the direction indicated by the red arrow. If the agent does not move down at this step or some earlier steps, the ball would pass through, making the agent lose a point. (b) The ball has already passed through the agent. Any decision at this step does not save the situation and does not affect the environment much.

effective key steps, and of finding them automatically without human knowledge. We confirm both possibilities by introducing an RL framework where the attacker *learns* the key steps *from scratch* through interacting with the agent.

In particular, we study the *key-step identification problem* proposed by Lin et al (2017). We prove that this constrained optimization problem can be converted into another form that matches the objective of RL. Then, we design a corresponding RL environment to train the attacker. Our contributions are summarized as follows:

- We formulate the key-step identification problem in an unconstrained form, and propose an RL framework that solves it directly without incorporating human knowledge. The proposed RL framework is independent of how adversarial examples are generated, and thus could be combined with white-box or black-box attacks.
- We justify the necessity of our proposed learning framework by constructing an example environment where existing human-designed heuristics fail to identify the key steps.
- We conduct thorough experiments on five benchmark Atari games to test the effectiveness of our proposed framework. The attacker trained by our framework can learn key steps that decrease the cumulative rewards of the RL agents more effectively than those steps found by existing human heuristics. The results confirm the possibility of finding more effective key steps automatically without human knowledge.

2 Background and Related Work

2.1 Reinforcement Learning

We begin by defining notations for the RL environment, namely the Markov Decision Process (MDP) (Sutton and Barto, 2018), which is a tuple $(\mathbb{S}, \mathbb{A}, P, R, P_0, \gamma)$. At each step t, an RL agent is in a state s_t from a finite set $\mathbb{S}$, and picks an action a_t from a finite set $\mathbb{A}$; then the environment returns a next state s_{t+1} with transition probability $P(s_{t+1} \mid s_t, a_t)$, and a bounded reward $r_t \in [-1, 1]$ sampled from a reward distribution $R(s_t, a_t)$. For convenience, we define $r(s_t, a_t) = \mathbb{E}_{r_t \sim R(s_t, a_t)}[r_t]$ to be the mean reward function. The state s_0 is drawn from an initial state distribution P_0, and $\gamma \in [0, 1]$ is a discount factor.

We work in the γ-discounted finite-horizon setting with a maximum step of T. Let Π be the set of all stationary stochastic policies that take actions in $\mathbb{A}$ given states in $\mathbb{S}$. The value function of a policy $\pi \in \Pi$ is defined to be the expected return:

$$V(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right],$$

where the expectation with respect to π denotes the expectation with respect to the trajectory it generates ($s_0 \sim P_0$, $a_t \sim \pi(s_t)$, and $s_{t+1} \sim P(s_t, a_t)$ for $t \geq 0$). Similarly, the Q-value function of a policy π starting from (s_t, a_t) is defined by

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t} \gamma^k r(s_{t+k}, a_{t+k}) \right].$$

To learn an optimal policy $\pi^* = \arg\max_{\pi \in \Pi} V(\pi)$, deep Q-network (DQN, Mnih et al 2015) first approximates the Q-value function by a neural network that takes a state as input and outputs the Q-values for all actions. The policy is then induced by picking the action with the largest Q-value at each state. Given a transition step (s_t, a_t, r_t, s_{t+1}), the network f_θ is trained by minimizing the Bellman squared error

$$\mathcal{L}_\theta = \left(\left(r_t + \gamma \max_a f_\theta(s_{t+1})_a \right) - f_\theta(s_t)_{a_t} \right)^2,$$

where we use $f_\theta(s)_a$ to denote the a-th entry of the Q-values $f_\theta(s)$.

2.2 Adversarial Attacks

In the context of adversarial attacks, an attacker tries to fool a target model by crafting adversarial examples that the target model would mis-classify (Biggio et al, 2013; Szegedy et al, 2014; Goodfellow et al, 2015). These adversarial examples are constructed through adding small perturbations to original examples. According to how much knowledge the attacker knows about the target model, adversarial attacks can be roughly divided into white-box ones and black-box ones.

White-box attacks assume that the attacker has full knowledge about the target model, including the target model's network parameters. For example, given a target model f_θ parametrized by θ and an image–label pair (x, y), the fast gradient sign method (FGSM, Goodfellow et al 2015) computes the perturbation as

$$\eta = \epsilon \, \mathrm{sign}(\nabla_x J(\theta, x, y)),$$

where ϵ is a scaling factor that controls the norm of the perturbation, and $J(\theta, x, y)$ is an objective function that depends on the attacker's goal. In untargetted attacks, the attacker aims to minimize the target model's classification accuracy, and thus the objective $J(\theta, x, y)$ is set to $D_{\mathrm{KL}}(e^{(y)} \| f_\theta(x))$, the Kullback-Leibler divergence between the label's one-hot encoding $e^{(y)}$ and the predicted probabilities $f_\theta(x)$. In targetted attacks, on the other hand, the attacker's goal is to make the target model predicts some other class $y' \neq y$, and thus the objective is set to $-D_{\mathrm{KL}}(e^{(y')} \| f_\theta(x))$.

The main disadvantage of white-box attacks is that their assumptions may not be satisfied in real-world tasks. In contrast, black-box attacks assume no access to network parameters and study other conditions instead, such as the ability to query model outputs (Chen et al, 2017). One of the most common black-box attacks is the substitute model approach (Papernot et al, 2016a, 2017), where the attacker computes perturbations using a substitute model that is trained to perform the same task as the target model. This approach is based on the finding that adversarial examples can be transferable; that is, if an adversarial example fools a model, it might also fool other models trained to perform the same task, regardless of the network architecture (Szegedy et al, 2014). Using this approach, once the attacker gains access to the training set, any white-box attack method can also be applied in a black-box setting.

2.3 Adversarial Attack on Deep RL Agents

As opposed to traditional adversarial attacks on classifiers, attacking an RL agent is a different task that comes with unique goals. When attacking a classifier, we typically aim to minimize the classification accuracy or to maximize the probability that the classifier predicts some given class. When attacking an RL agent, however, we usually do not care about individual actions that the agent picks (Behzadan and Munir, 2018; Xiao et al, 2019). Possible attacking goals include minimizing the agent's cumulative reward (Huang et al, 2017), luring the agent into a designated state (Lin et al, 2017), misguiding the agent to optimize an adversarial reward (Tretschk et al, 2018), or forcing the agent into executing a policy specified by the attacker (Rakhsha et al, 2020). The types of attacks against RL agents are also multifarious, ranging from testing-time attacks such as perturbation-based attacks (Huang et al, 2017) and action-based attacks (Lee et al, 2020), to training-time attacks such as reward-poisoning attacks (Zhang et al, 2020) and environment-poisoning attacks (Rakhsha et al, 2020). Learning adversarial policies in multi-agent systems has also been studied (Gleave et al, 2020).

In this work, we study the problem of minimizing the agent's cumulative reward through perturbation-based attacks at testing time (Lin et al, 2017; Kos and Song, 2017). The attacker needs to decide not only the way perturbations are generated, but also the key steps at which the attack would be effective, adding another dimension of complexity to the problem. Several studies discuss how perturbation-based attacks could be generated against RL agents. In the white-box setting, the FGSM has been used to craft adversarial examples that fool target agents into picking wrong actions (Behzadan and Munir, 2017a; Huang et al, 2017; Mandlekar et al, 2017; Pattanaik et al, 2018). Other white-box attacks such as the Jacobian saliency map algorithm (Papernot et al, 2016b) and the attack proposed by Carlini and Wagner (2017) are also shown to be effective against RL agents (Behzadan and Munir, 2017a; Lin et al, 2017). As for the black-box attacks, adversarial examples can be crafted with a substitute agent that is trained in the same environment as the target agent (Behzadan and Munir, 2017a; Huang et al, 2017) or in a learned environment (Inkawhich et al, 2019). Nevertheless, to the best of our knowledge, few studies have investigated the most effective steps to perform those attacks. We aim to study the underexplored aspect—identifying the key steps to attack. We shall treat the perturbation generation part as a procedure call in this work.

One reason to study attacks against RL agents is that a well-studied attacking scheme helps improve the agent's robustness. For instance, the agent's robustness to visual perturbations can be improved through adversarial training (Kos and Song, 2017; Mandlekar et al, 2017; Behzadan and Munir, 2017b; Pattanaik et al, 2018), a defense technique that adds adversarial examples into training sets (Goodfellow et al, 2015). On the other hand, both Pinto et al (2017) and Tessler et al (2019) obtain policies that are robust to environment changes by training the agent and the adversary in an alternating procedure, where the attacks are operated on the environment dynamics instead of visual perturbations. By proposing an effective attack method, we hope to stimulate the study of defense schemes and robust agents.

3 Learning Key Steps to Attack

3.1 The Key-Step Identification Problem

Suppose that an attacker would like to attack a target agent with policy π by adding perturbations to the states. We use η to denote an arbitrary attack method that maps from a state to a perturbation. Let B be the budget, the maximum number of attacks permitted in one episode. The key-step identification problem (Lin et al, 2017) can be formulated as follows:

$$\begin{aligned} \min_{b_0,\ldots,b_T \in \{0,1\}} \quad & \mathbb{E}\left[\sum_{t=0}^{T} r(s_t, a_t)\right] \\ \text{s.t.} \quad & s_0 \sim P_0, \\ & a_t \sim \pi(s_t + b_t \eta(s_t)),\ s_{t+1} \sim P(s_t, a_t), \\ & \sum_{t=0}^{T} b_t \leq B. \end{aligned} \tag{1}$$

In this problem, the attacker aims to minimize the expected return of the target agent under the budget constraint. The binary variable b_t indicates whether the perturbation $\eta(s_t)$ is added to the state s_t, and step t is a key step found by the attacker if $b_t = 1$.

Problem 1 is a difficult combinatorial problem with an exponentially large search space. Each decision of the attacker changes the subsequent steps. Moreover, the transition dynamics and reward distribution may be random. Even if the attacker manages to search the whole space by brute force, the collected experiences are merely samples; the attacker still needs to estimate and minimize the expected return. Due to these difficulties, previous studies (Lin et al, 2017; Kos and Song, 2017) rely on heuristics to find the key steps.

3.2 Existing Methods and Their Weakness

Suppose that the target agent is parametrized by a Q-network f_θ.[1] Kos and Song (2017) observe that steps with large Q-values are usually followed by large immediate reward. They hypothesize that an attack is effective at steps with large Q-values, and propose to set $b_t = 1$ if the maximum Q-value $\max_a f_\theta(s_t)_a$ is larger than a given threshold. On the other hand, Lin et al (2017) propose to add perturbations when the agent is confident about its action. They use the softmax function to convert the Q-values into a policy (i.e., $\pi = \mathrm{softmax} \circ f_\theta$), and set $b_t = 1$ if the probability gap $\max_a \pi(s_t)_a - \min_a \pi(s_t)_a$ is larger than a given threshold. We refer to these two heuristics as `large-value` (Kos and Song, 2017) and `large-prob-gap` (Lin et al, 2017) in the following.

However, these methods might not work well in general. An extreme case in the next proposition deliberately presents a scenario where existing heuristics are totally ineffective. In particular, the success of existing heuristics hinges on the goodness of the target agent, and thus the heuristics backfire when the target agent is far from optimal.

Proposition 1 *Let $Q : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ be the function satisfying*

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathbb{S}} P(s' \mid s, a) \min_{a' \in \mathbb{A}} Q(s', a').$$

[1] These heuristics can be applied to attack other value-based or policy-based agents in a similar way.

If the target agent estimates the Q-value function to be $-Q$ and follows the policy $\pi(s) = \arg\max_a -Q(s,a)$, then the expected return of π cannot decrease when the agent is attacked by the `large-value` *or* `large-prob-gap` *heuristics.*

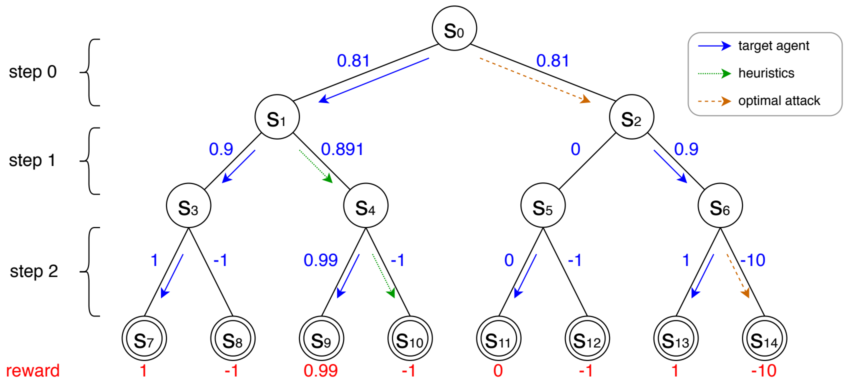


Fig. 2: An MDP example where prior methods fail. The MDP components are described as follows. A circle represents a state, and a double circle represents a terminal state. At each state, the agent can choose to go left or right, as shown by the lines. Red numbers under the terminal states represent the reward an agent gets if the agent reaches that state. The discount factor γ is set to 0.9. The target agent's estimated Q-values and policy are shown by the blue numbers and blue arrows, respectively. Let the budget constraint $B = 2$. To minimize the agent's return, a smart attacker should attack at steps 0 and 2, guiding the agent into state s_{14}. However, the `large-value` heuristic would attack at steps 1 and 2 since the maximum Q-value is larger at those steps. On the other hand, the `large-prob-gap` heuristic also attacks steps 1 and 2 since the gap of Q-values is larger at those steps (so the probability gap is larger too). As a result, both heuristics guide the agent into state s_{10}, failing to find the most effective key steps.

The proposition above reveals a case where a non-optimal target agent causes the heuristics to fail. Next, we demonstrate a carefully-crafted case where the heuristics fail on even an optimal target agent in Figure 2. In this example, the target agent under attack is of optimal Q-values and an optimal policy, but both heuristics are still unable to find the key steps. This toy example highlights the potential drawback of these heuristics. Since the attacker may change the agent's policy in future steps, the agent's Q-value estimates in the current step could be inaccurate. As a result, heuristics based on these Q-value estimates cannot find the most effective key steps in general.

A possible solution to fix the issue in Proposition 1 is to adopt the `large-absolute-value` heuristic that attacks when the absolute Q-values are the largest. Somehow the new heuristic suffers from other issues, such as being short-sighted to negative Q-values. The results for attacking the RL agent on an Atari game, Pong (Figure 6a), demonstrate that the `large-value` and `large-prob-gap` heuristics, as well as the new `large-absolute-value` one, are not as competitive as the proposed framework. The results justify that simple heuristics cannot always work well on identifying the key steps to attack RL agents.

The cases in this subsection show that existing heuristics can fail when targeted towards either an imperfect agent or a perfect one. From these cases it appears impossible to design an effective attack method without additional information. In the following, we propose a general RL framework that is able to learn key steps, by assuming additional interactions with the target agent.

3.3 Problem Transformation

The main obstacle of solving Problem 1 is the budget constraint. We first prove an equivalent formulation of Problem 1 based on the Lagrange relaxation technique, replacing the hard budget constraint with a soft penalty term, and then solve the problem through RL. For this purpose we introduce the following notations.

Given a cost function $c\colon \mathbb{S}\times\mathbb{A}\to\mathbb{R}$ and a vector-valued constraint function $g\colon \mathbb{S}\times\mathbb{A}\to\mathbb{R}^K$, we define the cost value function $C\colon \Pi\to\mathbb{R}$ and constraint value function $G\colon \Pi\to\mathbb{R}^K$ as

$$C(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t c(s_t,a_t)\right],$$
$$G(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t g(s_t,a_t)\right].$$

The next proposition states the equivalence between the constrained and regularized policy optimization problems.

Proposition 2 (A refinement of Proposition 2.1 in Le et al (2019)) *Consider the two policy optimization tasks:*

$$\texttt{Regularization:}\quad \min_{\pi\in\Pi} C(\pi)+\lambda^T G(\pi).$$
$$\texttt{Constraint:}\quad \min_{\pi\in\Pi} C(\pi)\quad s.t.\ G(\pi)\le\tau.$$

Assume that the constraint $G(\pi)\le\tau$ is feasible and cannot be removed without changing the optimal solution (i.e., $\inf_{\pi\in\Pi} C(\pi) < \inf_{\pi\in\Pi\colon G(\pi)\le\tau} C(\pi)$). Then for all $\lambda>0$, there exists τ, and vice versa, such that `Constraint` and `Regularization` share the same optimal solutions.

We provide a complete proof based on the occupancy measure (Puterman, 2014) in Appendix A. Proposition 2 states that the constrained policy optimization problem can be solved via an equivalent unconstrained form. With this proposition in hand we then have the following proposition.

Proposition 3 *Assume that the budget constraint in Problem 1 cannot be removed without changing the optimal solution. Consider the following problem:*

$$\begin{aligned} \min_{b_0,\dots,b_T\in\{0,1\}} \quad & \mathbb{E}\left[\sum_{t=0}^{T} r(s_t,a_t)\right] + \lambda\sum_{t=0}^{T} b_t \\ s.t. \quad & s_0 \sim P_0, \\ & a_t \sim \pi(s_t + b_t\eta(s_t)),\ s_{t+1}\sim P(s_t,a_t). \end{aligned} \tag{2}$$

For all $\lambda \geq 0$, there exists $B > 0$, and vice versa, such that Problem 1 and Problem 2 share the same optimal solutions.

This proposition is an instantiation of Proposition 2 and we also provide the proof in Appendix A. Intuitively, λ can be viewed as a parameter that controls the penalty for each attack. A larger penalty parameter λ corresponds to a smaller budget constraint B.

Proposition 3 states that we can transform the key-step identification problem into a sequential decision problem without additional constraints. Our next step is to solve this equivalent problem through RL by training an *attack policy* that learns to identify the key steps.

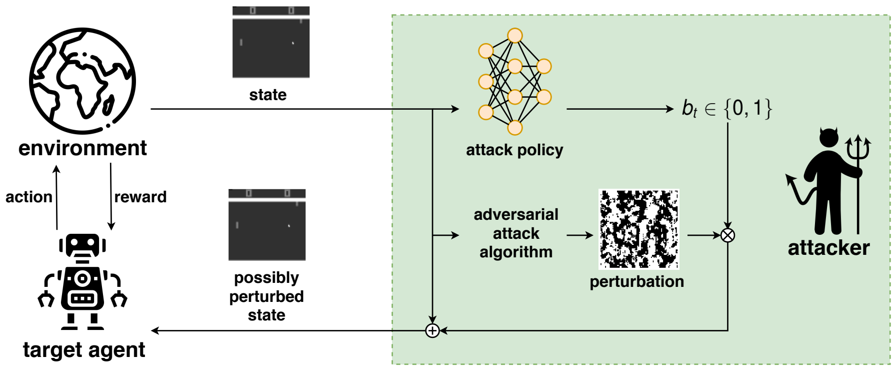


Fig. 3: Interactions among the environment, the target agent, and the attacker. If the attack policy outputs "1", the attacker would intercept the state from the environment and inject the perturbed state to the target agent; otherwise the state is left unperturbed. Different from prior methods, our method parametrizes the attack policy with a neural network and trains it by an RL framework.

3.4 An RL Framework for Learning Key Steps

Here we describe the proposed RL framework in detail. Suppose that the target agent with policy π is trained to maximize the expected return in an MDP $\mathcal{M} = (\mathbb{S}, \mathbb{A}, P, R, P_0, \gamma)$. We propose to train the attack policy π' in another MDP $\mathcal{M}' = (\mathbb{S}', \mathbb{A}', P', R', P_0', \gamma')$, where each component is defined as follows:

- $\mathbb{S}' = \mathbb{S}$, $\mathbb{A}' = \{0, 1\}$, $P_0' = P_0$, and $\gamma' = 1$.
- For all $s \in \mathbb{S}'$, $b \in \mathbb{A}'$, the transition dynamics $P'(s, b) = P(s, a)$, and the reward $R'(s, b) = -R(s, a) - b\lambda$, where $a \sim \pi(s + b\eta(s))$ is the target agent's action.

The new environment M' has the same state space as M, but reduces to binary action space with the action "1" representing an attack at that step and "0" otherwise. The attacker's reward is the negative of the target agent's reward, plus a penalty of $-\lambda$ for each attack. Therefore, maximizing the expected return in $\mathcal{M}'$ is equivalent to minimizing the objective in Problem 2. An illustration of the interactions among the environment, the target agent, and the attacker is given in Figure 3.

Our proposed RL framework is general. We make no assumption on the environment or on the target agent. In addition, our RL framework can be freely paired with any RL algorithm and with any attack method. Thus, our framework can be applied to white-box attack scenarios as well as black-box ones.

4 Experiments

We perform experiments on the toy example in Figure 2 and five Atari 2600 games in the Arcade Learning Environment (Bellemare et al, 2013). We aim to answer the following questions in our experiments: (1) whether our RL framework can learn more efficient key steps than prior methods, and (2) whether the empirical behavior is consistent with the theory.

4.1 Toy Example

As a sanity check, we test our framework on the toy example described in Figure 2. Since the target agent has only two possible actions, we simulate the attack by directly changing the target agent's action. We train the attack policy using DQN, with other details provided in Appendix B. The results are shown in Figure 4. When trained with the penalty parameter $\lambda = 10^1$, RL attack policy would perform no attacks since the penalty for each attack is too large. On the other hand, when trained with $\lambda \in \{10^0, 10^{-1}\}$, the attack policy is able to learn the most effective key steps (under a budget constraint $B = 2$), outperforming prior heuristics. As our analysis suggests, we can learn the effective key steps with appropriate values of λ.

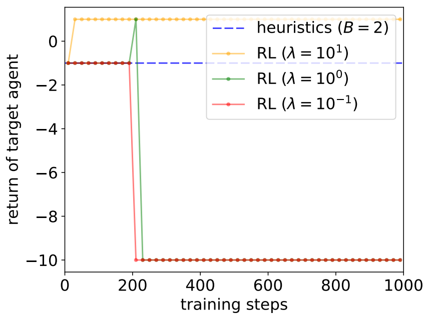


Fig. 4: Toy example results. The return of target agent is averaged over ten testing episodes.

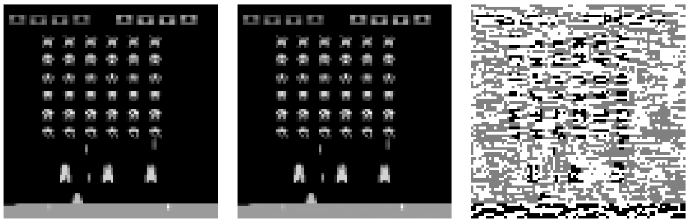


Fig. 5: An example of perturbation generated by the FGSM with $\epsilon = 0.01$. *Left:* Original frame. *Middle:* Perturbed frame. *Right:* Added perturbation (rescaled to $[0, 1]$ for visualization). The frames before and after perturbation look indistinguishable to human eyes.

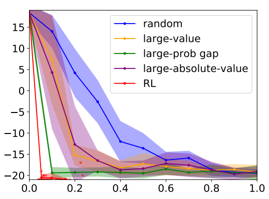
4.2 Atari

Next, we evaluate our framework on five Atari benchmark games (Pong, Space Invaders, Seaquest, Riverraid, Battlezone), which cover a variety of environments. The environment setting and preprocessing match the guidelines suggested by Machado et al (2018). In particular, we use a sticky-action probability of 0.25 in these environments. For the target agent, we take pretrained DQN agents from Dopamine (Castro et al, 2018), which have the same network architecture as Mnih et al (2015), and fix them thereafter. We also train attack policies using DQN with the same architecture. Other hyperparameters are reported in Appendix B.

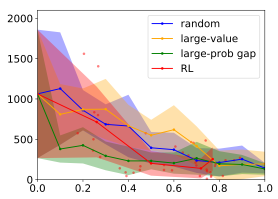
Throughout the experiments, we use the untargetted FGSM to generate perturbations due to its computation efficiency. We use Foolbox (Rauber et al, 2017) to generate the adversarial examples with the perturbation norm constraint ϵ set to 0.01. Figure 5 shows an example of the perturbation, which is imperceptible to human.

To make the scenario more realistic, we follow Hussenot et al (2020) and perturb the target agent's observation, rather than the whole state. That is, although the target agent stacks the latest four frames as its current state, the attacker is only allowed to add perturbation to the latest frame. We compare our method to a `random` baseline that attacks uniformly at random, and to two heuristics, `large-value` (Kos and Song, 2017) and `large-prob-gap` (Lin et al, 2017). For the Pong environment, we include `large-absolute-value` as another heuristic for comparison.

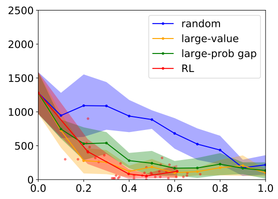
4.2.1 Performance Comparison with Small Training Budget



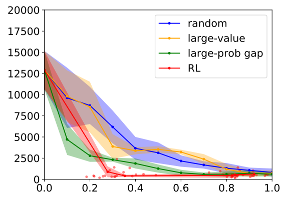
(a) Pong



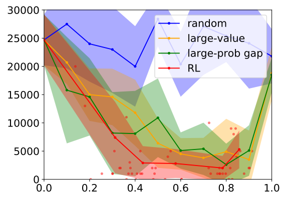
(b) Space Invaders



(c) Seaquest



(d) Riverraid



(e) Battlezone

Fig. 6: Performance comparison of attack policies. The vertical axis is the target agent's undiscounted return, and the horizontal axis is the attack ratio (the number of attacks divided by the number of steps in an episode). Under the same attack ratio, the lower the target agent's undiscounted return is, the better the attack method does. All reported scores are averaged over ten testing episodes and shown with one standard deviation. The results of the two heuristics are obtained by setting ten different thresholds on their corresponding criteria. For our RL method, we plot the score for each learned policy at 10M training frames, and show the testing episodes by red dots.

For each environment, we first train five attack policies with penalty parameter $\lambda \in \{10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}\}$ for 10M training frames, which is only 1/20 of the training cost of the target agent. Figure 6 plots the results in the white-box setting. Despite the small training cost, when there is enough budget on attacking, the attack policy learned in our RL framework achieves superior performance in Pong and Riverraid, and exhibits competitive performance to all competitors in other environments.

The results provide a positive answer to our first question on whether our RL framework can learn more efficient key steps than prior heuristics. Somehow it is worth noting some parts of the curve that our RL framework does not reach the best performance. One is when the budget of attacking is small, like less than 20% on Riverraid. In this case, intuitively there is less need to learn a strategic policy, and a simple heuristic seems to be strong enough. The other is when there is a huge variance about the effectiveness of attacks, such as on Space Invaders, Seaquest, and Battlezone. Even in those cases, we see that the RL framework usually stays competitive to other heuristics, while being often better than random attacks.

4.2.2 The Effect of the Penalty Parameter

Table 1: Comparison of our RL attack policy with different λ (10M training frames). Results are averaged over ten testing episodes with the standard deviation shown in parentheses.

		λ				
		10^1	10^0	10^{-1}	10^{-2}	10^{-3}
Pong	return	18.4 (1.9)	18.4 (1.9)	-20.5 (0.7)	-20.6 (1.2)	-20.9 (0.3)
	attack ratio (%)	0.0 (0.0)	0.0 (0.0)	5.3 (1.0)	9.7 (4.7)	15.8 (3.1)
Space Invaders	return	1063.0 (796.4)	714.0 (438.7)	201.5 (152.5)	139.5 (124.6)	254.0 (147.4)
	attack ratio (%)	0.0 (0.0)	26.0 (4.9)	50.0 (9.4)	71.9 (5.3)	77.1 (8.1)
Seaquest	return	1280.0 (308.4)	414.0 (183.3)	84.0 (72.6)	54.0 (34.7)	122.0 (94.0)
	attack ratio (%)	0.0 (0.0)	21.8 (4.6)	39.8 (7.7)	47.7 (5.2)	61.1 (5.2)
Riverraid	return	888.0 (636.1)	407.0 (127.3)	464.0 (295.9)	546.0 (113.6)	616.0 (328.7)
	attack ratio (%)	27.8 (5.8)	35.3 (7.0)	82.5 (2.6)	83.6 (5.0)	92.2 (5.6)
Battlezone	return	7400.0 (6327.7)	2900.0 (2981.6)	2800.0 (2561.2)	2000.0 (2144.8)	5300.0 (3348.1)
	attack ratio (%)	31.2 (8.1)	43.4 (7.5)	57.8 (11.1)	78.4 (4.5)	85.7 (4.2)

To investigate the penalty parameter λ, we summarize the performance of the learned attack policies with white-box attacks in Table 1. As λ increases, the learned attack policy tends to have a lower attack ratio. Also, training with too large values of λ would prevent the attack policy from launching any attacks. The tradeoff between the penalty parameter and budget constraint confirms the theory.

These results suggests that choosing an appropriate λ is important for learning effective key steps. Empirically, we observe that setting λ around the same order of magnitude as

$$\text{Average potential reward loss per step} = \frac{\text{Original return of target agent} - \text{Minimum return}}{\text{Number of steps per episode}} \tag{3}$$

would produce stable results (see Table 1, Appendix C.1, and Appendix C.2). We thus list Equation (3) to provide a stable λ as an initial value. A possible future direction is to allow the proposed RL framework to be combined with adaptive control or AutoML tools to get the right λ under a given attack budget in different environments.

4.2.3 Performance Comparison with Full Training Budget

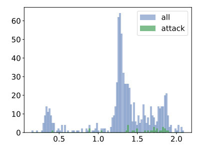
Table 2: Performance comparison under the same attack ratio with white-box attack. We train RL attack policies for 200M training frames with the chosen λ, and then we test the competitors using the same attack ratios as the learned attack policies. Results are averaged over ten testing episodes with the standard deviation shown in parentheses.

	λ	attack ratio (%)	return			
			RL	random	large-value	large-prob-gap
Pong	10^{-1}	3.9 (0.8)	**-21.0** (0.0)	15.4 (2.1)	15.7 (1.7)	-11.4 (3.9)
Space Invaders	10^{0}	10.3 (3.1)	**103.5** (117.4)	1292.0 (754.3)	1018.0 (578.6)	398.0 (223.7)
Seaquest	10^{0}	13.5 (3.5)	**362.0** (110.4)	978.0 (333.4)	662.0 (374.7)	746.0 (140.9)
Riverraid	10^{1}	17.9 (3.3)	**446.0** (62.6)	8981.0 (2142.1)	7014.0 (3770.7)	3495.0 (1800.5)

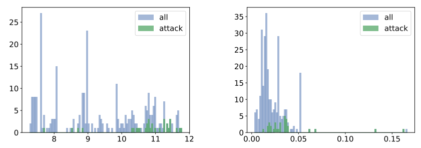
Next, we choose a value of λ that is close to the average potential reward loss in Equation (3) for each environment, and train the attack policy for 200M training frames, the same as the target agent's training budget. [2] The results are shown in Table 2. The RL attack policy improves with the full training budget and outperforms prior methods by a large margin. These results further validate that key steps are learnable, and that our method can learn more effective key steps automatically without any domain knowledge.

4.2.4 Behavior Comparison of the Attack Policies

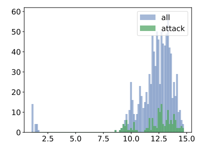
To understand to what extent the attack policy learned in our RL framework is different from human heuristics, we attack the target agent by the learned attack policy, and compute the statistics used by the two heuristics at each step. Figure 7 plots the histogram of those statistics in Pong and Space Invaders. The key steps found by our method spread across different intervals, rather than focusing on the largest intervals. This result suggests that our method does not simply mimic the heuristics, but learns unique key steps to attack.



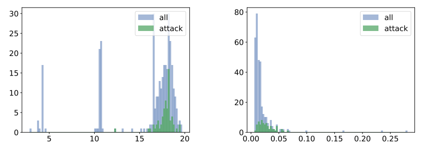
(a) Pong



(b) Space Invaders

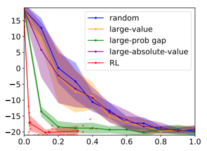


(c) Seaquest

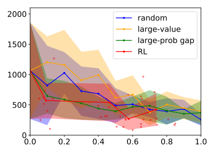


(d) Riverraid

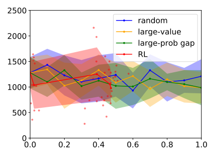
Fig. 7: Behavior comparison of our method to previous heuristics. *Left:* Histogram of the maximum Q-value computed by `large-value.` *Right:* Histogram of the probability gap computed by `large-prob-gap.` (*all:* number of all steps in one episode; *attack:* number of steps that our method attacks)



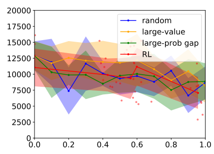
(a) Pong



(b) Space Invaders



(c) Seaquest



(d) Riverraid

Fig. 8: Performance comparison of attack policies with black-box attacks. This figure is the black-box version of Figure 6 (white-box). It can be observed that the RL-learned policy remains competitive on Pong.

4.2.5 Black-Box Scenario

We adopt the substitute model approach and take the substitute agent from Dopamine. The substitute agent has the same architecture as the target agent, but is trained with a different random seed and network initialization. This setting is the same as the experiments of "Transferability Across Policies" in Huang et al (2017). Perturbations for adversarial attacks and statistics for `large-value` and `large-prob-gap` are then computed with respect to the substitute agent.

Due to limited resources, we only train the attackers for 10M training frames. Results are shown in Figure 8 and the comparison of different λ can be

[2]Because of computational resource constraints, the following experiments are conducted on the first four environments only.

Table 3: Comparison of our RL policy with different λ with black-box attacks (10M training frames). Results are averaged over ten testing episodes with the standard deviation shown in parentheses.

		λ				
		10^{1}	10^{0}	10^{-1}	10^{-2}	10^{-3}
Pong	return	18.4 (1.9)	14.7 (2.2)	-17.2 (2.5)	-20.2 (1.0)	-19.7 (1.6)
	attack ratio (%)	0.0 (0.0)	0.1 (0.0)	3.1 (0.4)	12.4 (3.5)	31.5 (6.9)
Space Invaders	return	1063.0 (796.4)	571.0 (294.4)	539.5 (309.2)	269.5 (183.6)	422.5 (267.9)
	attack ratio (%)	0.0 (0.0)	9.3 (2.8)	46.6 (4.7)	57.7 (6.3)	73.9 (7.1)
Seaquest	return	1280.0 (308.4)	1038.0 (463.8)	1252.0 (524.2)	1056.0 (340.8)	978.0 (310.2)
	attack ratio (%)	0.0 (0.0)	2.3 (1.3)	39.0 (4.2)	46.6 (5.2)	47.2 (4.3)
Riverraid	return	11057.0 (2950.7)	9891.0 (2584.1)	9353.0 (3229.2)	11176.0 (2942.4)	7069.0 (2488.5)
	attack ratio (%)	0.0 (0.1)	42.8 (4.0)	56.2 (3.1)	59.9 (3.5)	95.7 (1.9)

found in Table 3. In Pong, our method improves over baseline and prior methods. In other environments, we observe higher rates of attack failures, which we hypothesize to be the reason that causes all methods to show similar performance. On the comparison with different λ, we observe similar trends as the white-box scenario, where larger λ corresponds to smaller attack ratio. Overall, even with a small training budget, our method is still able to outperform or perform comparably to previous methods in the black-box scenario.

5 Discussion

The experiment results on simulated environments demonstrate the potential of our proposed RL framework in being superior in attacking at the key steps than human-designed heuristics. The heuristics are admittedly competitive when the attack budget is low or when the efficacy of attacks is less certain; the proposed framework, however, is significantly more effective in the other scenarios, especially in the white-box setting.

The findings above can be useful to a willing attacker to launch attacks successfully. Suppose the attacker has enough budget and a competent method for producing effective attacks. In that case, the attacker can immediately employ our proposed framework to attack the target agent through interactions. The proposed framework spends the budget efficiently by selecting the key steps. Furthermore, the key steps offered by the framework are significantly different from those by a single heuristic, which makes it harder for the system running the target agent to detect the attempt to attack.

The usefulness discussed above, however, comes with two assumptions. The assumptions may restrict the feasibility of a willing attacker's application of the proposed framework. The first assumption is a competent method for producing effective attacks. As our experiments demonstrate, this assumption is easier to achieve in the white-box setting but only sometimes possible in the black-box setting. The white-box setting is a very strong restriction in the real world, as the target agent's model is typically not fully accessible. The second assumption is that the framework is executed with a reasonable λ. While we attempt to provide a reasonable λ through Equation (3) based on our experience in the simulated environments, it is unclear whether such a λ also applies to real-world environments. Furthermore, the relationship between λ and the budget B cannot be easily computed. Therefore, an attacker running on a strict budget may find it challenging to locate a good λ corresponding to the desired budget.

Suppose we switch from the attacker's viewpoint to the perspective of any safety-sensitive RL system. The framework is arguably useful for identifying the weakness of the system. Once the weakness, in the form of key steps, is located, we can apply techniques like adversarial training to strengthen the system's robustness. In addition, the framework is usually feasible in such a scenario, as the developers hold complete knowledge of the models within the system and can use multiple λ to stress-test the system. With its usefulness and feasibility, the proposed framework has the potential to help enhance the robustness of any safety-sensitive RL system.

6 Conclusion and Future Direction

We show that the key-step identification problem can be solved by training an attack policy through an RL framework. Compared to existing studies, our method learns unique key steps without any human knowledge and can be flexibly paired with white-box or black-box settings. Results on Atari benchmarks validate our belief that the proposed method can learn more effective key steps in the white-box setting. We also demonstrate the potential of the proposed framework in the black-box setting on Pong. Nevertheless, for Atari environments other than Pong, all the existing approaches cannot attack more effectively than random attacking in the black-box setting. The results suggest that while attacks may be transferable, the key steps are not easily transferable. We hope that this work could serve as a building stone towards more practical attacks and defenses of deep RL agents.

One interesting future direction is to blend the strength of RL and humans, such as using RL to learn a dynamic threshold on top of human heuristics. The dynamic thresholding can possibly be achieved by some specially-designed state space that encodes the human heuristics. Another direction is to study the condition on when key steps can be transferable, like Pong, which helps understand the nature of attacking at the key steps. The other direction is to couple the RL framework with adversarial training and alternating training techniques (Pinto et al, 2017; Tessler et al, 2019) to robustify the deep RL agent.

Declarations

- Funding: The authors did not receive support from any organization for the submitted work.

- Conflict of interest/Competing interests: The authors have no competing interests to declare that are relevant to the content of this article.
- Ethics approval: We will have read and agreed to follow the Committee on Publication Ethics guidelines.
- Consent to participate: Not applicable.
- Consent for publication: Not applicable.
- Availability of data and materials: All data/materials that are used to conduct experiments in this work are publicly available.
- Code availability: The source codes that are used to generate all experimental results can be downloaded from this url[3].
- Authors' contributions: All authors contributed to the design of this work. Initial idea, method design, experiment implementation, and the first draft of the manuscript were completed by Chien-Min Yu. Hsuan-Tien Lin provided numerous advises on idea formation, method design, experiment presentation, as well as proofreading and revising the draft. Additional experiments are implemented and carried out by Ming-Hsin Chen.

References

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

Behzadan V, Munir A (2017b) Whatever does not kill deep reinforcement learning, makes it stronger. arXiv preprint arXiv:171209344

Behzadan V, Munir A (2018) The faults in our pi stars: Security issues and open challenges in deep reinforcement learning. arXiv preprint arXiv:181010369

Bellemare MG, Naddaf Y, Veness J, et al (2013) The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research 47:253–279

Biggio B, Corona I, Maiorca D, et al (2013) Evasion attacks against machine learning at test time. In: Joint European conference on machine learning and knowledge discovery in databases, Springer, pp 387–402

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge university press

Carlini N, Wagner D (2017) Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), IEEE, pp 39–57

[3]https://drive.google.com/file/d/1Qdd64tmMB1IgrbOobBCu_j4X7zMO0QVf/view?usp=sharing

Castro PS, Moitra S, Gelada C, et al (2018) Dopamine: A research framework for deep reinforcement learning. arXiv preprint arXiv:181206110

Chen PY, Zhang H, Sharma Y, et al (2017) Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, ACM, pp 15–26

Escandell-Montero P, Chermisi M, Martinez-Martinez JM, et al (2014) Optimization of anemia treatment in hemodialysis patients via reinforcement learning. Artificial intelligence in medicine 62(1):47–60

Gleave A, Dennis M, Wild C, et al (2020) Adversarial policies: Attacking deep reinforcement learning. In: International Conference on Learning Representations

Goodfellow I, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. In: International Conference on Learning Representations

Huang S, Papernot N, Goodfellow I, et al (2017) Adversarial attacks on neural network policies. arXiv preprint arXiv:170202284

Hussenot L, Geist M, Pietquin O (2020) Copycat: Taking control of neural policies with constant attacks. In: International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)

Inkawhich M, Chen Y, Li H (2019) Snooping attacks on deep reinforcement learning. arXiv preprint arXiv:190511832

Isele D, Rahimi R, Cosgun A, et al (2018) Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 2034–2039

Kos J, Song D (2017) Delving into adversarial attacks on deep policies. arXiv preprint arXiv:170506452

Le H, Voloshin C, Yue Y (2019) Batch policy learning under constraints. In: Proceedings of Machine Learning Research, vol 97. PMLR, Long Beach, California, USA, pp 3703–3712

Lee XY, Ghadai S, Tan KL, et al (2020) Spatiotemporally constrained action space attacks on deep reinforcement learning agents. In: Association for the Advancement of Artificial Intelligence (AAAI)

Levine S, Finn C, Darrell T, et al (2016) End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research 17(1):1334–1373

Lin YC, Hong ZW, Liao YH, et al (2017) Tactics of adversarial attack on deep reinforcement learning agents. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, pp 3756–3762

Machado MC, Bellemare MG, Talvitie E, et al (2018) Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. Journal of Artificial Intelligence Research 61:523–562

Mandlekar A, Zhu Y, Garg A, et al (2017) Adversarially robust policy learning: Active construction of physically-plausible perturbations. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 3932–3939

Mnih V, Kavukcuoglu K, Silver D, et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529

Pan X, You Y, Wang Z, et al (2017) Virtual to real reinforcement learning for autonomous driving. In: Proceedings of the British Machine Vision Conference (BMVC)

Papernot N, McDaniel P, Goodfellow I (2016a) Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:160507277

Papernot N, McDaniel P, Jha S, et al (2016b) The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, pp 372–387

Papernot N, McDaniel P, Goodfellow I, et al (2017) Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia conference on computer and communications security, ACM, pp 506–519

Pattanaik A, Tang Z, Liu S, et al (2018) Robust deep reinforcement learning with adversarial attacks. In: International Conference on Autonomous Agents and MultiAgent Systems

Pinto L, Davidson J, Sukthankar R, et al (2017) Robust adversarial reinforcement learning. In: Proceedings of Machine Learning Research, vol 70. PMLR, International Convention Centre, Sydney, Australia, pp 2817–2826

Puterman ML (2014) Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons

Raghu A, Komorowski M, Ahmed I, et al (2017) Deep reinforcement learning for sepsis treatment. arXiv preprint arXiv:171109602

Rakhsha A, Radanovic G, Devidze R, et al (2020) Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In: Proceedings of the International Conference on Machine Learning

Rauber J, Brendel W, Bethge M (2017) Foolbox: A python toolbox to benchmark the robustness of machine learning models. arXiv preprint arXiv:170704131

Silver D, Huang A, Maddison CJ, et al (2016) Mastering the game of go with deep neural networks and tree search. nature 529(7587):484

Silver D, Hubert T, Schrittwieser J, et al (2018) A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 362(6419):1140–1144

Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press

Syed U, Bowling M, Schapire RE (2008) Apprenticeship learning using linear programming. In: Proceedings of the 25th International Conference on Machine learning, pp 1032–1039

Szegedy C, Zaremba W, Sutskever I, et al (2014) Intriguing properties of neural networks. In: International Conference on Learning Representations

Tessler C, Efroni Y, Mannor S (2019) Action robust reinforcement learning and applications in continuous control. In: Proceedings of Machine Learning Research, vol 97. PMLR, Long Beach, California, USA, pp 6215–6224

Tretschk E, Oh SJ, Fritz M (2018) Sequential attacks on agents for long-term adversarial goals. arXiv preprint arXiv:180512487

Xiao C, Pan X, He W, et al (2019) Characterizing attacks on deep reinforcement learning. arXiv preprint arXiv:190709470

Zhang X, Ma Y, Singla A, et al (2020) Adaptive reward-poisoning attacks against reinforcement learning. In: Proceedings of the International Conference on Machine Learning

Appendix A Proofs of Section 3

Here we provide proofs for the propositions.

Proposition 1 *Let $Q : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ be the function satisfying*

$$Q(s,a) = r(s,a) + \gamma \sum_{s' \in \mathbb{S}} P(s' \mid s,a) \min_{a' \in \mathbb{A}} Q(s',a').$$

If the target agent estimates the Q-value function to be $-Q$ and follows the policy $\pi(s) = \arg\max_a -Q(s,a)$, then the expected return of π cannot decrease when the agent is attacked by the `large-value` or `large-prob-gap` heuristics.

Proof The target agent has policy $\pi(s) = \arg\max_a -Q(s,a) = \arg\min_a Q(s,a)$. By definition of Q, this policy achieves minimum expected return: $\pi = \arg\min_{\pi\in\Pi} V(\pi)$. Therefore, its expected return cannot decrease. Furthermore, the expected return strictly increases if there is an attack that changes the target agent's action from a to a' at some state s with $Q(s,a) \neq Q(s,a')$. □

Proposition 2 (A refinement of Proposition 2.1 in Le et al (2019)) *Consider the two policy optimization tasks:*

$$\texttt{Regularization:} \quad \min_{\pi\in\Pi} C(\pi) + \lambda^T G(\pi).$$
$$\texttt{Constraint:} \quad \min_{\pi\in\Pi} C(\pi) \quad s.t.\ G(\pi) \le \tau.$$

Assume that the constraint $G(\pi) \le \tau$ is feasible and cannot be removed without changing the optimal solution (i.e., $\inf_{\pi\in\Pi} C(\pi) < \inf_{\pi\in\Pi:\ G(\pi)\le\tau} C(\pi)$). Then for all $\lambda > 0$, there exists τ, and vice versa, such that `Constraint` and `Regularization` share the same optimal solutions.

Proof Let π^* be an optimal policy in `Regularization` for some $\lambda > 0$, and let $\tau^* = G(\pi^*)$. Suppose that there exists $\pi \in \Pi$ such that $G(\pi) \le \tau^*$ and $C(\pi) < C(\pi^*)$. Then we have

$$C(\pi) + \lambda^T G(\pi) < C(\pi^*) + \lambda^T \tau^* = C(\pi^*) + \lambda^T G(\pi^*),$$

contradicting to the optimality of π^*. Thus, π^* is also optimal in `Constraint` with $\tau = \tau^*$.

To prove the other direction, let us first define for a policy $\pi \in \Pi$ its occupancy measure $\rho_\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ as

$$\pi(a \mid s) \sum_{t=0}^{T} \gamma^t Pr(s_t = s \mid \pi).$$

The occupancy measure can be viewed as an unnormalized distribution of state–action visitation, and for any function $c \colon \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ we have

$$C(\pi) := \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t c(s_t, a_t) \right] = \sum_{s,a} \rho_\pi(s,a) c(s,a).$$

The set $\mathbb{D} := \{\rho_\pi : \pi \in \Pi\}$ of valid occupancy measures can be written as a feasible set of affine constraints (Puterman, 2014):

$$\mathbb{D} = \left\{ \rho : \rho \geq 0 \text{ and } \forall s \in \mathbb{S}, \sum_a \rho(s,a) = P_0(s) + \gamma \sum_{s',a} P(s \mid s', a)\rho(s', a) \right\}.$$

Furthermore, we have a one-to-one correspondence between Π and $\mathbb{D}$ (Theorem 2 of Syed et al 2008): If $\rho \in \mathbb{D}$, then ρ is the occupancy measure for $\pi_\rho := \rho(s,a)/\sum_{a'} \rho(s,a')$, and π_ρ is the only policy whose occupancy measure is ρ.

Therefore, we can rewrite the two problems as linear programs:

$$\texttt{Regularization:} \quad \min_{\rho \in \mathbb{D}} \sum_{s,a} \rho(s,a) \left(c(s,a) + \lambda^T g(s,a) \right).$$

$$\texttt{Constraint:} \quad \min_{\rho \in \mathbb{D}} \sum_{s,a} \rho(s,a) c(s,a) \quad \text{s.t.} \sum_{s,a} \rho(s,a) g(s,a) \leq \tau.$$

Let ρ^* be an optimal solution of `Constraint` for some τ. The Lagrangian of `Constraint` is given by

$$L(\rho, \lambda) = \sum_{s,a} \rho(s,a) \left(c(s,a) + \lambda^T g(s,a) \right) - \lambda^T \tau,$$

and we have $\rho^* = \arg\min_{\rho \in \mathbb{D}} \max_{\lambda \geq 0} L(\rho, \lambda)$. Let $\lambda^* = \arg\max_{\lambda \geq 0} \min_{\rho \in \mathbb{D}} L(\rho, \lambda)$. Since strong duality holds for any linear program provided that the primal problem is feasible (Boyd and Vandenberghe, 2004), we have

$$\begin{aligned} \rho^* &= \arg\min_{\rho \in \mathbb{D}} \max_{\lambda \geq 0} L(\rho, \lambda) \\ &= \max_{\lambda \geq 0} \arg\min_{\rho \in \mathbb{D}} L(\rho, \lambda) \\ &= \arg\min_{\rho \in \mathbb{D}} L(\rho, \lambda^*) \\ &= \arg\min_{\rho \in \mathbb{D}} \sum_{s,a} \rho(s,a) \left(c(s,a) + (\lambda^*)^T g(s,a) \right) - (\lambda^*)^T \tau. \end{aligned}$$

Removing the constant $-(\lambda^*)^T \tau$, we have that ρ^* is optimal in `Regularization` with $\lambda = \lambda^*$. Finally, since the constraint cannot be removed without changing the optimal solution, $\rho^* \neq \arg\min_{\rho \in \mathbb{D}} \sum_{s,a} \rho(s,a) c(s,a)$, and we must have $\lambda^* > 0$.

$\square$

Proposition 3 *Assume that the budget constraint in Problem 1 cannot be removed without changing the optimal solution. Consider the following problem:*

$$\begin{aligned} \min_{b_0, \ldots, b_T \in \{0,1\}} \quad & \mathbb{E}\left[\sum_{t=0}^{T} r(s_t, a_t) \right] + \lambda \sum_{t=0}^{T} b_t \\ s.t. \quad & s_0 \sim P_0, \\ & a_t \sim \pi(s_t + b_t \eta(s_t)),\ s_{t+1} \sim P(s_t, a_t). \end{aligned} \tag{2}$$

For all $\lambda \geq 0$, there exists $B > 0$, and vice versa, such that Problem 1 and Problem 2 share the same optimal solutions.

Proof Suppose that the target agent with policy π is trained to maximize the expected return in an MDP $\mathcal{M} = (\mathbb{S}, \mathbb{A}, P, R, P_0, \gamma)$. We define the cost function $c\colon \mathbb{S} \times \{0,1\} \to \mathbb{R}$ as

$$c(s,b) = -\mathbb{E}_{a\sim\pi(s+b\eta(s))}\left[\mathbb{E}_{r\sim R(s,a)}[r]\right],$$

and the constraint function $g\colon \mathbb{S} \times \{0,1\} \to \mathbb{R}$ as $g(s,b) = b$. Let Π_b be the set of all stationary stochastic policies that take actions in $\{0,1\}$ given states in $\mathbb{S}$. Define the cost value function $C\colon \Pi_b \to \mathbb{R}$ and constraint value function $G\colon \Pi_b \to \mathbb{R}$ as

$$C(\pi_b) = \mathbb{E}_{\pi_b}\left[\sum_{t=0}^{T} c(s_t, b_t)\right],$$

$$G(\pi_b) = \mathbb{E}_{\pi_b}\left[\sum_{t=0}^{T} g(s_t, b_t)\right],$$

where $s_0 \sim P_0$, $b_t \sim \pi_b(s_t)$, $a_t \sim \pi(s + b\eta(s))$, and $s_{t+1} \sim P(s_t, a_t)$ for $t \geq 0$. Using these definitions, we can rewrite Problem 1 as

$$\min_{\pi\in\Pi_b} C(\pi) \quad \text{s.t. } G(\pi) \leq B,$$

and Problem 2 as

$$\min_{\pi\in\Pi_b} C(\pi) + \lambda G(\pi).$$

It remains to apply Proposition 2. □

Appendix B Experimental Settings

In the toy example, we use one-hot encoding to represent each state for all 15 states. The Q-network is composed of five fully-connected layers, with the dimensions for hidden layers being $(128, 128, 512, 512)$. For Atari games, the architecture of Q-network follows Mnih et al (2015). All hyperparameters are listed in Table B1 and Table B2.

Preliminary results comparing the effect of discount factor γ are provided in Table B3. The results do not clearly show which value is better for training attack policies. We choose to use $\gamma = 0.99$ throughout our experiments, as this value is the common choice when training Atari agents.

Table B1: Hyperparameters for the toy example.

Hyperparameter	Value
optimizer	Adam
learning rate	0.0001
batch size	32
discount factor	1
target network update frequency	10
replay buffer size	100
learning start step	0
learning frequency	1
exploration type	ϵ-greedy
epsilon decay type	linear decay
exploration decay horizon	300
minimum epsilon during training	0.01
epsilon during testing	0

Table B2: Hyperparameters for Atari games.

Hyperparameter	Value
optimizer	Adam
learning rate	0.0001
batch size	32
discount factor	0.99
target network update frequency	1000
replay buffer size	100000
learning start step	20000
learning frequency	4
exploration type	ϵ-greedy
epsilon decay type	linear decay
exploration decay horizon	250000
minimum epsilon during training	0.01
epsilon during testing	0.001
adversarial attack method	FGSM
maximum norm constraint for perturbation	0.01
perturbation searching intervals	100

Table B3: Comparison of discount factor γ used in training white-box attack policies (10M training frames). Results are averaged over ten testing episodes with the standard deviation shown in parentheses.

	γ		λ				
			10^{1}	10^{0}	10^{-1}	10^{-2}	10^{-3}
Pong	0.99	return	18.4 (1.9)	18.4 (1.9)	-20.5 (0.7)	-20.6 (1.2)	-20.9 (0.3)
		attack ratio (%)	0.0 (0.0)	0.0 (0.0)	5.3 (1.0)	9.7 (4.7)	15.8 (3.1)
Pong	1	return	18.4 (1.9)	18.4 (1.9)	-18.1 (2.5)	-19.7 (0.9)	-19.3 (1.1)
		attack ratio (%)	0.0 (0.0)	0.0 (0.0)	3.8 (0.7)	22.0 (3.1)	49.0 (3.4)
Space Invaders	0.99	return	1063.0 (796.4)	714.0 (438.7)	201.5 (152.5)	139.5 (124.6)	254.0 (147.4)
		attack ratio (%)	0.0 (0.0)	26.0 (4.9)	50.0 (9.4)	71.9 (5.3)	77.1 (8.1)
Space Invaders	1	return	1063.0 (796.4)	190.0 (108.7)	93.5 (59.5)	199.5 (193.6)	156.5 (118.0)
		attack ratio (%)	0.0 (0.0)	62.9 (10.7)	46.4 (11.4)	43.3 (13.7)	67.8 (12.5)

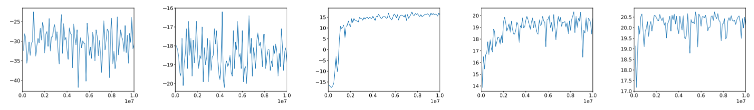
Appendix C Additional Experimental Results

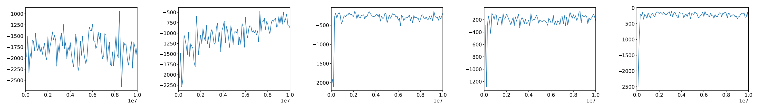
C.1 Statistics of the Target Agent

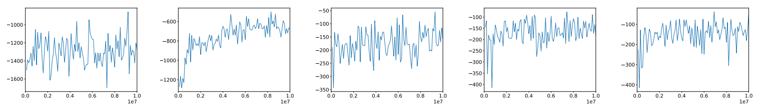
Table C4: Statistics of the target agents averaged over ten episodes.

	Original return of target agent	Minimum return	Number of steps per episode	Average potential reward loss (Equation 3)
Pong	18.4	-21.0	2498.9	1.58×10^{-2}
Space Invaders	1063.0	0.0	1196.5	8.88×10^{-1}
Seaquest	1280.0	0.0	1531.5	8.36×10^{-1}
Riverraid	12912.0	0.0	1772.6	7.28×10^{0}

C.2 Training Curves (White-Box)


(a) Pong


(b) Space Invaders


(c) Seaquest

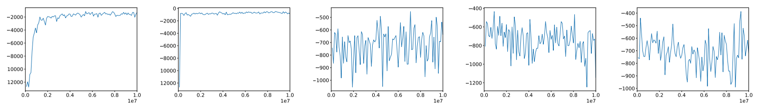
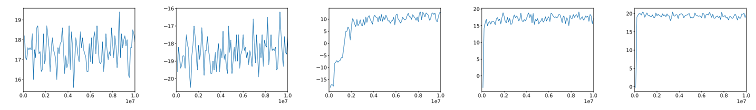
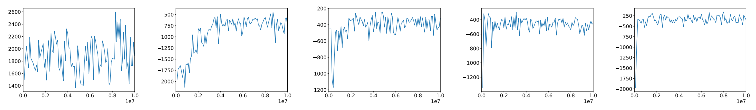

(d) Riverraid

Fig. C1: Attack policy's undiscounted return in MDP M' during training ($\lambda = 10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}$ from left to right). The attack policy is tested every 100000 steps and we report scores averaged over ten testing episodes.

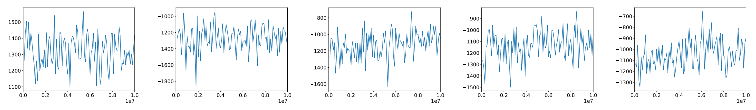
C.3 Training Curves (Black-Box)



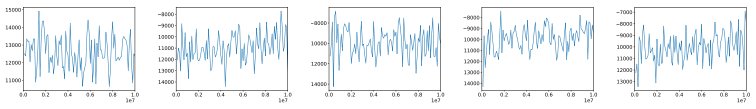
(a) Pong



(b) Space Invaders



(c) Seaquest



(d) Riverraid

Fig. C2: Attack policy's undiscounted return in MDP M' during training ($\lambda = 10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}$ from left to right) with black-box attacks. The attack policy is tested every 100000 steps and we report scores averaged over ten testing episodes.