Tactics of Adversarial Attack on Deep Reinforcement Learning Agents

Yen-Chen Lin[*], Zhang-Wei Hong[*], Yuan-Hong Liao[*], Meng-Li Shih[*], Ming-Yu Liu[†], and Min Sun[*]
National Tsing Hua University, Hsinchu, Taiwan[*]
Nvidia, Santa Clara, CA, USA[†]
{yenchenlin1994,sean.mingyu.liu}@gmail.com
{williamd4112@gapp,s102061137@m102,shihsml@gapp,sunmin@ee}.nthu.edu.tw

Abstract

We introduce two novel tactics for adversarial attack on deep reinforcement learning (RL) agents: strategically-timed and enchanting attack. For strategically-timed attack, our method selectively forces the deep RL agent to take the least likely action. For enchanting attack, our method lures the agent to a target state by staging a sequence of adversarial attacks. We show that DQN and A3C agents are vulnerable to both tactics. Future work on defending is discussed in App. C.

1 Introduction

The existence of adversarial examples to deep reinforcement learning (RL) agents is largely unexplored. In a contemporary paper, Huang et al. (2017) proposed the uniform attack, which attacks a deep RL agent with adversarial examples at every time step in an episode for reducing the reward of the agent. We argue that it does not consider several important aspects of RL. First of all, the spirit of adversarial attack is to perform a minimal perturbation of the observation to avoid being detected. In RL, this should be considered both in the spatial and temporal domains. Therefore, we propose the *strategically-timed attack* which reduces the reward while limiting the number of times an adversarial attack is launched. Our adversary attacks a deep RL agent to take the worse action when it believes the attack is effective (e.g., s_{84} is more effective than s_{25} in Fig. 1). In our experiments, we show that this attack significantly reduces reward on state-of-the-art deep RL agents by attacking a small portion of the episode (on average 25% of the time steps in an episode). Another important characteristic of RL is that it's a closed loop problem since agent's action will affect its later inputs. Motivated by this aspect, we propose the enchanting attack for maliciously luring a deep RL agent to a specified state. In the real world, the enchanting attack may be used to mislead a self-driving car controlled by a deep RL agent to hit a certain obstacle. We implement the enchanting attack using a planning algorithm and a deep generative model for predicting future states of the environment. To the best of our knowledge, this is the first planning-based adversarial attack on a deep RL agent. Our experiment results show that the enchanting attack has a 70% or more success rate in attacking state-of-the-art deep RL agents. Next, we describe both attack tactics in detail.

1.1 Strategically-Timed Attack

In an episode, a deep RL agent observes a sequence of states $\mathcal{S} = \{s_1, ..., s_L\}$. Instead of attacking at every time step in an episode, the strategically-timed attack selects a subset of time steps to attack the deep RL agent. Let $\Delta = \{\delta_1, ..., \delta_L\}$ be a sequence of perturbations. Let $\mathcal{R}_1$ be the expected return at the first time step. We can formulate the above intuition as an optimization problem as follows:

$$\begin{aligned} \min_{b_1,...,b_L,\delta_1,...,\delta_L} \quad & R_1(\bar{s}_1, ..., \bar{s}_L) \\ & \bar{s}_t = s_t + b_t\delta_t \quad \text{for all } t = 1, ..., L; \\ & b_t \in \{0,1\}, \quad \text{for all } t = 1, ..., L; \quad \sum_t b_t \leq \Gamma \end{aligned} \tag{1}$$

The binary variables $b_1, ..., b_L$ denote when an attack is launched. If $b_t = 1$, the perturbation δ_t is applied. Otherwise, we do not alter the state. The total number of attacks is limited by the constant

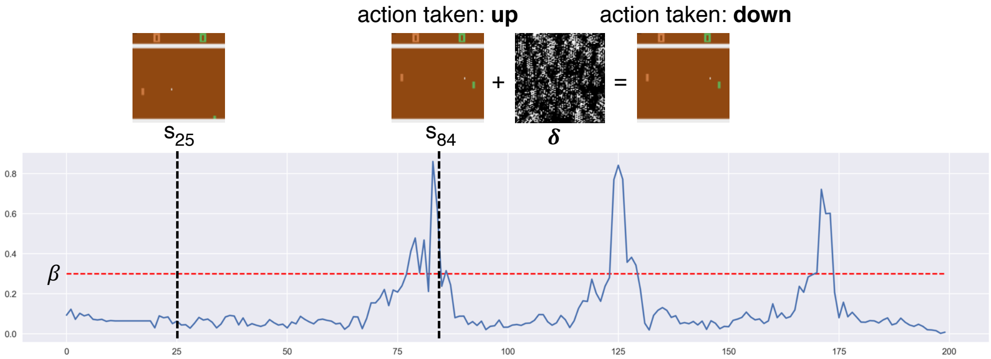


Figure 1: Illustration of the strategically-timed attack on Pong. We use a function c to compute the preference of the agent in taking the most preferred action over the least preferred action at the current state s_t. A large preference value implies an immediate reward. In the bottom panel, we plot $c(s_t)$. Our proposed strategically-timed attack launch an attack to a deep RL agent when the preference is greater than a threshold, $c(s_t) > \beta$ (red-dash line). When a small perturbation is added to the observation at s_{84}, the agent changes its action from up to down and eventually misses the ball. But when the perturbation is added to the observation at s_{25}, there is no impact to the reward. Note that the perturbation δ is enlarged to 250 times for visualization.

Γ. In words, the adversary minimizes the expected accumulated reward by strategically attacking less than $\Gamma < L$ time steps. When the adversary decide to attack the agent, it craft an adversarial example to mislead the agent. The optimization problem in (1) is a mixed integer programming problem, which is difficult to solve in general. We bypass this limitation and propose a heuristic algorithm to compute $\{b_1, ..., b_L\}$ (solving the when-to-attack problem) and $\{\delta_1, ..., \delta_L\}$ (solving the how-to-attack problem), respectively.

When to attack. For policy gradient-based methods such as A3C, when an well-trained agent's action distribution is rather uniform at state s, it implies that performing any action is equally good. In contrast, when an agent is rather determined to take a specific action, it implies that it's critical to perform the action with the highest probability. For value-based methods such as DQN, the same intuition applies, since we can convert the computed Q-values of actions into probability distribution over actions using softmax with temperature (similar to Huang et al. (2017)). In detail, we measure the preference of the agent in taking the most preferred action over the least preferred action with function $c(s)$ at state s using the probability of the best action minus the probability of the worse action. When attacking online, we craft an adversarial example (i.e., $b_t = 1$) when $c(s)$ is larger than a threshold β, which is related to Γ.

How to attack. To craft an adversarial example at time step δ_t, we search for a perturbation pattern to be added to the observation that can change the preferred action of the agent from the originally (before applying the perturbation) most preferred one to the originally least preferred one. We use the attack method introduced in Carlini & Wagner (2016) where we treat the least-preferred action as the misclassification target. This approach allows us to leverage the behavior of a trained deep RL agent to craft an adversarial example.

1.2 Enchanting Attack

The goal of enchanting attack is to lure the agent from current state s_t to a specified target state s_g after H steps. First, the task is reduced to planning a sequence of actions for reaching the target state s_g (see Appendix B). Next, we craft an adversarial example $s_t + \delta_t$ (Carlini & Wagner (2016)) to lure the agent to take the first planned action. After the agent observes the adversarial example and takes an action, the environment will return a new state s_{t+1}. We progressively craft $s_{t+1} + \delta_{t+1}, ..., s_{t+H} + \delta_{t+H}$, one at a time, using the same procedure described in (Fig. 2).

2 Experiment

We evaluate our novel tactics on 5 Atari 2600 games using OpenAI Gym(Brockman et al. (2016)) (i.e., Ms.Pacman, Pong, Seaquest, Qbert, and Chopper Command). We choose them as a balanced collection of games with super-human level (e.g. Pong) and below human level (e.g., Ms.Pacman).

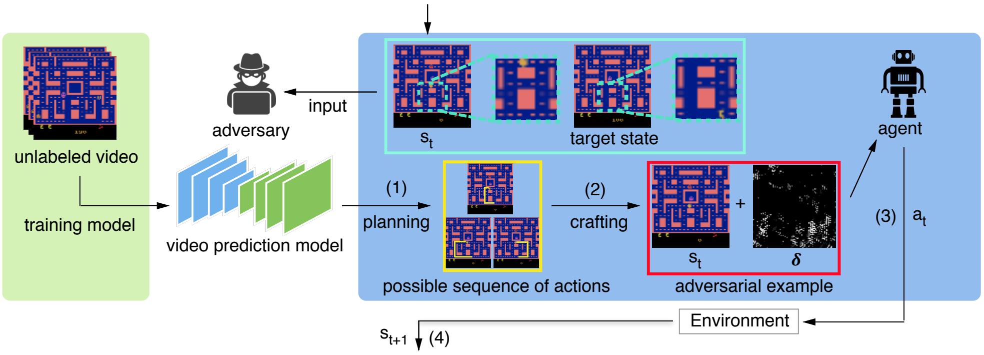
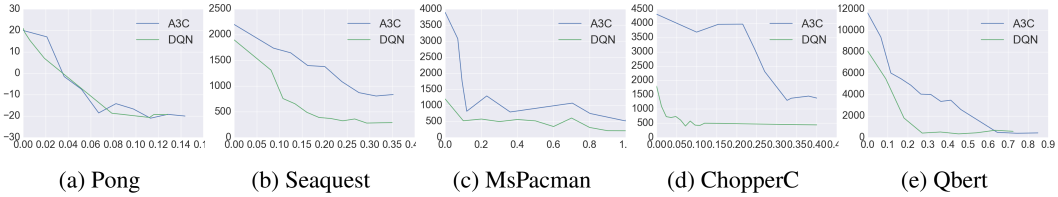
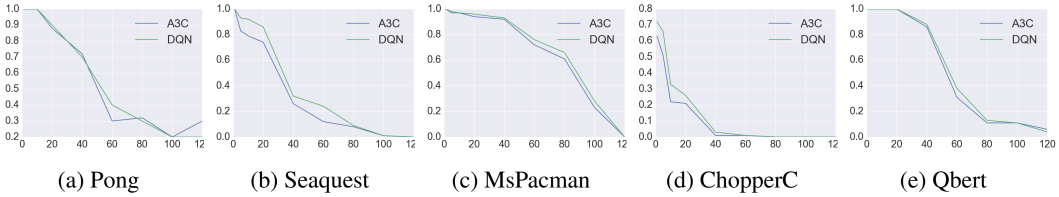


Figure 2: Illustration of Enchanting Attack on Ms.Pacman. Blue panel on the right shows the flow of the attack starting at s_t: (1) action sequence planning, (2) crafting an adversarial example with a target-action, (3) the agent takes an action, and (4) environment generates the next state s_{t+1}. Green panel at the left depicts that the video prediction model is trained from unlabeled video. White panel in the middle depicts the adversary starts at s_t and utilize the prediction model to plan the attack. Note that the perturbation δ is enlarged to 250 times for visualization.



(a) Pong (b) Seaquest (c) MsPacman (d) ChopperC (e) Qbert

Figure 3: Accumulated reward (y-axis) v.s. Attack rate (x-axis) of Strategically-timed Attack in 5 games. Blue and green curves correspond to results of A3C and DQN, respectively. A larger reward means the deep RL agent is more robust to the attack. CopperC denotes CopperCommand.



(a) Pong (b) Seaquest (c) MsPacman (d) ChopperC (e) Qbert

Figure 4: Success rate (y-axis) v.s. H steps in the future (x-axis) for Enchanting Attack in 5 games. Blue and green curves correspond to results of A3C and DQN, respectively. A lower rate means that the deep RL agent is more robust to the attack. CopperC denotes CopperCommand.

Our experimental setup is described in Appendix A. Our results are presented below and typical examples are shown in `https://goo.gl/WC14vQ`.

Strategically-timed attack. For each game and for the agents trained by the DQN and A3C algorithms, we launched the strategically-timed attack using different β values. Each β value rendered a different attack rate (i.e., portion of attacked time steps in an episode). We computed the collected rewards by the agents under different attack rates. The results are shown in Fig. 3 where the y-axis is the accumulated reward and the x-axis is the attack rate. From the figure, we found that on average the strategically-timed attack can reach the same effect of the uniform attack by attacking 25% of the time steps in an episode. We also found an agent trained using the DQN algorithm was more vulnerable than an agent trained with the A3C algorithm in most games except Pong.

Enchanting attack. We considered an attack was successful if the final state had a normalized Euclidean distance to the target state within a tolerance value of 1. To make sure the evaluation was not affected by different stages of the game, we set 10 initial time step t equals to $[0.0, 0.1, ..., 0.9] \times L$, where L was the average length of the game episode played by the RL agents 10 times. For each initial time step, we evaluated different $H = [1, 5, 10, 20, 40, 60, 80, 100, 120]$. Then, for each H, we computed the success rate. In Fig. 4, we show the success rate (y-axis) as a function of H in 5 games. We found that the agents trained by both the A3C and DQN algorithms were enchanted. When $H < 40$, the success rate was more than 70% in 3 out of 5 games.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *https://arxiv.org/abs/1608.04644*, 2016.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *https://arxiv.org/abs/1702.02284*, 2017.

Volodymyr Mnih, Adria Puigdomenech Badia, and Mehdi Mirza. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *NIPS*, pp. 2863–2871, 2015.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2016a.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*, 2016b.

Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

A Experimental Setup

For each game, the deep RL agents were trained using the state-of-the-art deep RL algorithms: A3C, and DQN. For the A3C agents, we used the same pre-processing steps and neural network architecture as in Mnih et al. (2016). We also used the same network architecture for the Q-function trained by DQN. The input to the neural network at time t was a concatenation of the last 4 images, resized to 84×84. Values of each pixel were rescaled to be from 0 to 1. The output of the policy was a distribution over possible actions for A3C, and an estimate of Q values for DQN.

Although several existing methods can be used to craft an adversarial example (e.g., the fast gradient sign method (Goodfellow et al. (2015)), and Jacobian-based saliency map attack (Papernot et al. (2016a))), anti-adversarial attack measures were also discovered to limit their impact Goodfellow et al. (2015); Papernot et al. (2016b). Therefore, we decided to adopt a recent method proposed by Carlini & Wagner (2016), which has been shown to break several existing anti-adversarial attack methods. We early stopped the optimizer when $\mathcal{D}(s, s+\delta) < \epsilon$, where ϵ is a small value. Throughout our experiments, we set $\epsilon = 0.007$.

For testing the enchanting attack, we synthesized target states randomly in order to avoid the bias of defining target states manually. Firstly, we let the agent to run using its policy by t steps to reach an initial state s_t and saved this state into a snapshot. Secondly, we randomly sampled a sequence of actions of length H and let the agent performed these actions to reach a state s_{t+H}. We consider s_{t+H} as a synthesized target state s_g. After recording the target state, we restored the snapshot and run the enchanting attack on the agent and compared the normalized Euclidean distance between the target state s_g and the final reached state s_{t+H}.

B Planning

We describe how to plan a sequence of actions to reach a specified target state s_g from current state s_t after a finite horizon H steps.

Future state prediction and evaluation. Based on the work of Oh et al. (2015), which use a generative model to predict a video frame in the future, we train a video prediction model M to predict a future state given a sequence of actions as follows,

$$s_{t+H}^{M} = M(s_t, A_{t:t+H}) \ , \tag{2}$$

where $A_{t:t+H} = \{a_t, ..., a_{t+H}\}$ is the given sequence of H future actions beginning at step t, s_t is the current state, and s_{t+H}^{M} is the predicted future state. For more details about the video prediction model M, please refer to the original paper.

The series of actions $A_{t:t+H} = \{a_t, ..., a_{t+H}\}$ take the agent to reach the state s_{t+H}^{M}. Since the goal of the enchanting attack is to reach the target state s_g, we can evaluate the success of the attack based on the similarity between s_g and s_{t+H}^{M}, which is given by $D(s_g, s_{t+H}^{M})$. The similarity D is realized using the normalized Euclidean distance in our experiments while other metrics can be applied as well. We also note that the state is given by the observed images by the agent in our experiments.

Sampling-based action planning. We use a sampling-based approach to compute a sequence of actions to steer the RL agent toward our target state. Our approach is a cross-entropy method (CEM) (Rubinstein & Kroese (2013)). Specifically, we sample N action sequence of length H: $\{A_{t:t+H}^{n}\}_{n=1}^{N}$, and rank each of them based on the distance between the final state obtained after performing the action sequence and the target state s_g. After that, we keep the best K action sequences and refit our categorical distributions to them. In our experiments, the hyper-parameter values are $N = 2000$, $K = 400$, and $J = 5$.

At the end of the last iteration, we take the sampled action sequence $A_{t:t+H}^{*}$ that results in a final state most close to our target state s_q as our plan. Then, we craft an adversarial example with target-action a_t^{*} using the method introduced in Carlini & Wagner (2016). Instead of directly crafting the next action a_{t+1}^{*}, we plan for another enchanting attack starting at state s_{t+1} to be robust to any failure in the previous attack.

C Future Work

Attacking deep agent is an important first step toward defending attacks. We plan to explore two ideas for defending in the future: (1) train RL with adversarial example, (2) detect adversarial example first and then try to mitigate the effect. We believe this paper opens the door for researches on how to make deep agent less vulnerable to different tactics of adversarial attacks.