Exploration of Adversarial Inputs to Reinforcement Learning Agents

Ka Lok Ng
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
nkl018@ie.cuhk.edu.hk

Xiao Yi
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
yx019@ie.cuhk.edu.hk

Abstract

Supervised neural networks suffer from adversarial examples, which are crafted inputs that fool neural networks to output labels different from humans' perspective (with a success rate much higher than the model's error rate). We aim to exploit further this vulnerability on reinforcement learning (RL) agents empowered by Deep-Q learning because they are built on top of supervised neural networks. In this works, we implement an adversarial attack against RL agents in a competitive game (CompetitivePong-v0) and confirm that RL agents suffer from adversarial examples. Their rewards drop dramatically when an adversary can control the screen pixels. We proposed *universal perturbation attacks* against RL agents, where the adversary can add a frame-independent noise to the screen to confuse the agents. Furthermore, we found in our experiments that the universal perturbation is transferable to other agents using the same neural network architecture.

1 Introduction

Adversarial examples have been a well-known attack targeting at supervised neural networks for image processing. This attack can mislead a neural network to classify an image as a class dramatically contrast to humans' perception. The idea is that a small perturbation (noise) is added to an image to move the image across a decision boundary of the neural network. Through a gradient descent algorithm, we can efficiently find such perturbations. Nowadays, many reinforcement learning (RL) agents equip supervised neural networks to predict the future reward of states and evaluate policy. By adding perturbation to states, an adversary may able to fool an RL agent to take irrational moves that the agent would not take otherwise because it equipped wrongly evaluate the current situation.

Such attacks can lead to severe harm in the physical world. Imagine that an RL agent drives an auto-pilot vehicle. If the agent is fooled by a perturbed input from a vehicle's sensor, the car may crash and lead to casualty.

In this paper, we consider an even more powerful attack: universal perturbations. This attack adds a constant perturbation to every input but is still able to fool a classifier. Come back to our example of auto-pilot vehicles, If an attack add a "mud", a universal perturbation, to a sensor of the car, the car will easily crash even when the attacker does not know what exactly the sensor will capture. As shown in Figure 1, a universal perturbation may be added to the car's camera and leads the car to drive to a wrong direction and even a crash.

We implemented a proof-of-concept prototype on CompetitivePong-v0, a two-agent competitive zero-sum game. In this game, the victim agent receives the screen pixels of the game plays, and we add perturbation to the pixels. We tried both frame-dependent and frame-independent perturbation attack, which are analog of the original perturbation attack and universal perturbation, respectively, and shows that both of them can drastically reduce the rewards of the agent.

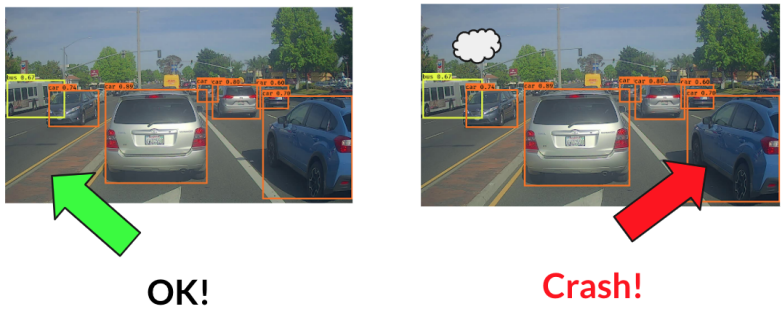


Figure 1: A frame-independent perturbation, illustrated on the right figure, may lead a auto-pilot vehicle to crash.

To summarize, we make the following contributions in this paper:

- *Frame-independent Perturbation Attacks.* We propose an attack that is more practical than the original perturbation attack for RL agents. It can mislead RL agents to make a bad decision even if the attacker does not know the exact inputs to the agents at run-time.

- *Implementation and Evaluation.* We demonstrate the effectiveness of our attack by hugely reducing the performance of the victim RL agents in a competitive game.

2 Related Works

With the assumption of an attacker having the ability to manipulate the environment of reinforcement learning and machine learning, a line of works have been proposed.

Behzadan and Munir[1] proposed a threat model where attackers can directly manipulate the observable environment, e.g., injecting perturbation in the game play screen, thereby alluring the victim agent to a desirable state. It also established that Deep Q-Networks are vulnerable to adversarial input perturbations and verified the transferability of adversarial examples across different DQN models.

Similarly, [4] used FGSM[3], which is originally proposed to fool image classifier using machine learning methods to produce wrong results, to compute adversarial perturbations for a trained neural network policy, *i.e.*, TRPO, A3C, and DQN, and achieved a significant decrease of policy performance.

Graph classifiers can also be fooled by modifying the graph with minimal perturbations. In [7], Sun, Wang, etc., introduced a node injection attack that can poison graphs that leads to the reduction of the classifier's accuracy. They proposed a reinforcement learning approach to manipulate the edges and labels of the injected nodes smartly.

As [4] only considered to fool the network to output a wrong action but not the action that reduces the most reward, [5] proposed two types of attacks, *i.e.*, strategically-times attack and enchanting attack to solve the problem. The former one selects a subset of time steps to attack the agent, and the later one lures the deep RL agent from the current state to a specified target state after several steps.

Besides directly modifying the observable environment, Gleave, Dennis, *etc.*, [2] aimed at attacking a victim agent by choosing an adversarial policy within a multi-agent environment and creating natural observations that are adversarial, i.e., the adversarial agent can manipulate its (native) action space and has access to the victim's policy.

While the above works require the knowledge of every bit of the observable environment to compute the perturbation, we present our work, i.e, universal perturbation, as a much more practical attack that does not require the knowledge of exact input at run-time.

3 The proposed Algorithm: Perturbation for RL Agents

In our basic attack against RL agents, the basic pixel perturbation is generated by FGSM, which uses a gradient descent algorithm to find out the minimal changes on pixels that can make the classification goes to a particular label. Before a frame or a frame stack is fed to the agent to decide the next action, we use the agent's neural network's internal parameters to compute the policy loss as if it should go to a wrong direction, let say, always up in CompetitivePong-v0, and then we back-propagate the gradient back to the input as a perturbation, which is eventually fed to the agent and mislead it to a wrong action.

Inspired by universal perturbation for a supervised neural network for image processing [6] To make our attack more practical that the attack does not need to know what exact will appear on the screen and not need to generate the perturbation on run-time, we adopts *universal perturbation attack* against RL agents, in which the perturbation is frame-independent.

Initialize a perturbation $\mathbf{v} \leftarrow 0$;
Define the maximum volume of perturbations $\epsilon \leftarrow \beta$;
while *True* **do**
  $\mathbf{X}_{\mathsf{train}} \leftarrow \mathsf{Env}(\mathsf{Agent})$;
  for $\mathbf{x} \in \mathbf{X}_{\mathsf{train}}$ **do**
    $\Delta\mathbf{v} \leftarrow \operatorname{argmin}_{\mathbf{r}} \|\mathbf{r}\|_2$ s.t. $\mathsf{Act}_{\mathsf{Agent}}(\mathbf{x}+\mathbf{v}+\mathbf{r}) = y$;
    $\mathbf{v} = \mathsf{proj}(\epsilon, \mathbf{v}+\Delta\mathbf{v})$
  end
  Save $\mathbf{v}$;
  Lower the maximum volume $\epsilon = \epsilon \cdot \alpha$;
end

Algorithm 1: Finding Universal Perturbations

The pseudo-code of this algorithm is described in Algorithm 1, where β is the initial maximum volume of perturbations, $\mathsf{proj}(\epsilon, \mathbf{v})$ is the projection of $\mathbf{v}$ onto the region where the maximum amplitude is ϵ, namely, projecting $\mathbf{v}$ onto a L_∞ ball with radius ϵ, and α is a predefined decay rate of the maximum volume.

Firstly, we initialize the universal perturbation as a zero-vector. Then, we generate several episodes and extract all the frames to form a training set. for each frame, on top of the current universal perturbation, we use a gradient descent algorithm to find the minimal perturbation to point the agent to make a particular move, and project that to a limited region of the universal perturbation to prevent it from growing too fast. After an iteration of training, we scale down the universal perturbation a bit to make it less noticeable to human inspectors. We restart from the training set generation step until the agent average episode reward decrease to a decided low score.

However, this algorithm still leaves us an important question: How to find a small change on the perturbation to fool the classifier? Put this question in another way; we want to utilize a gradient descent algorithm to compute

$$\Delta\mathbf{v} \leftarrow \operatorname*{argmin}_{\mathbf{r}} \|\mathbf{r}\|_2 \text{ s.t. } \mathsf{Act}_{\mathsf{Agent}}(\mathbf{x}+\mathbf{v}+\mathbf{r}) = y$$

This problem can be tackled by slightly loosing the constraints that we formulate a weighted sum over the policy loss and the norm of the perturbation. The optimization now becomes finding

$$\text{minimizing } \|\mathbf{r}\|_2^2 + \mathsf{PolicyProb}_{\mathsf{Agent}}(\mathbf{x}+\mathbf{v}+\mathbf{r}, y)$$

Similar to our frame-dependent basic attack,s the use of the usual gradient descent algorithm is enough to solve this problem.

Notice that the above attack is related to knowledge of the agents' internal parameters. In other words, we are launching a white-box attack We further notice in our experiments that the universal perturbation not only frame-independent but also can apply to other agents with the same neural networks architecture, *i.e.*, the perturbation is parameters-independent, leading to a huge decrease in their reward.

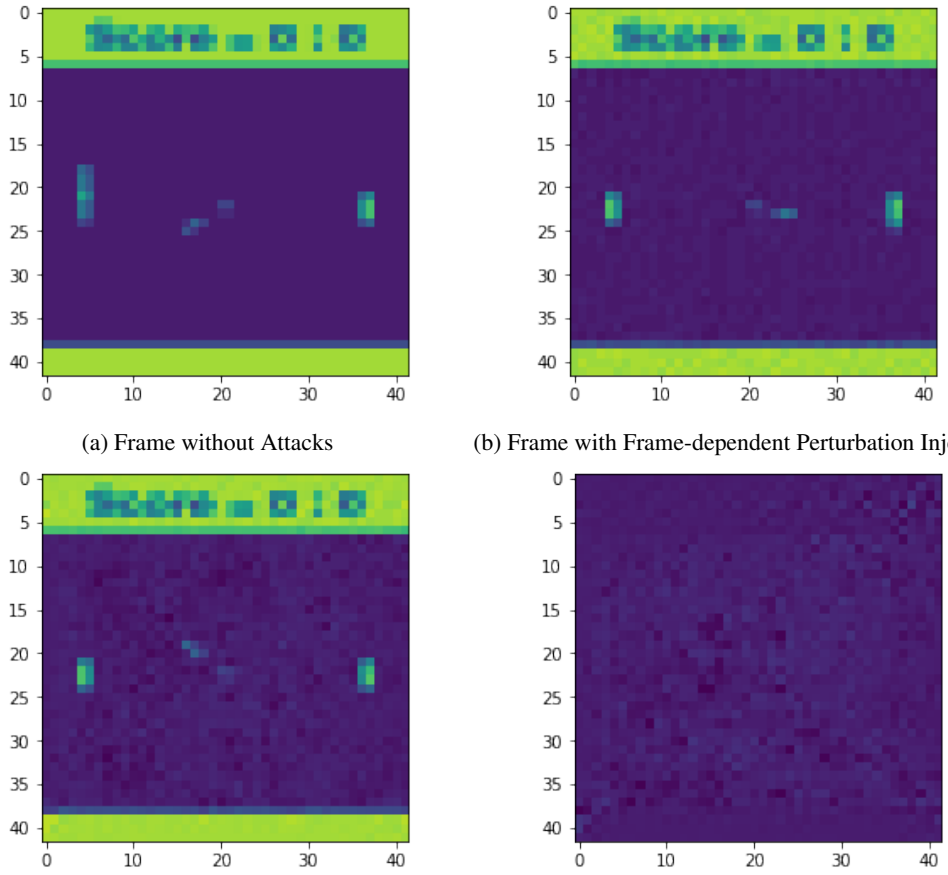
4 Experiments

We implemented our attacks, including the basic frame-dependent perturbation attacks and frame-independent attacks, on CompeitivePong-v0 with PyTorch. The implementation is public avaliable on a online repository [1].

The target environment CompeitivePong-v0 is a zero-sum competitive game where two paddles try to bound back a ball to each other, and the one who misses the ball losses a score, leading to reset of the environment subsequently. The whole episode ends when all 21 games are completed. With the highest score is the champion, and the score is the reward of each agent.

In the game, the agent can control the paddle on the left side. To the agents capture more information, we reshape each frame into 42×42 pixels and stacks fours frames into a tensor as an input state for the agent. Our attacks add perturbation on those tensors.

For Competitive-v0, we trained several agents equipped with VGG-style neural networks, some of which also trained using self-play strategy.



(a) Frame without Attacks

(b) Frame with Frame-dependent Perturbation Injected

(c) Frame with Frame-independent Perturbation Injected

(d) The Universal Perturbation

Figure 2: The Pixels with Perturbation Added and the Universal Perturbation

Some examples of the frames with perturbation are shown in Figure 2. Figure 2a is a normal frame without any perturbation. Figure 2b is a frame with a frame-dependent perturbation. Figure 2c is a frame with the frame-independent perturbation. Figure 2d is an universal perturbation for the target agent.

Both the frame-dependent/independent perturbation are hard to notice in humans' perception.

[1] `https://github.com/Lucieno/adv_pong/tree/universal`

	No Attack	Frame-dep.	Frame-indep.	Block-box	Transferred
Agv. Eps. Reward	7.8	-20.6	-18.2	3.0	-15.8
Norm of Perturb.	-	5	30.0	-	30.0

Table 1: Agents' Average Episode Reward and Perturbation Norm under Different Attack

Table 1 shows the baseline average episode reward of agents and their reward under perturbation attack. Initially, the white-box victim agent has an average episode reward of 7.8. Under frame-dependent perturbation attack, its reward dropped to -20.6, which is close to the worst case, *i.e.*, -21. Under our frame-dependent perturbation attack, the reward decreased to -18.2, while the L2-norm increased to 30.0. To demonstrate that the universal perturbation can harm the performance of agents with the same neural network architecture, we apply the perturbation to another agent with different parameters, which is also unknown to us, *i.e.*, a black-box agent. The results show that the perturbation is applicable to that agent, whose reward dropped to -15.8.

5 Conclusion

In this work, we propose an agent-transferable and frame-independent perturbation attacks for RL agents. We implemented the attacks in CompeitivePong-v0 and showed significant performance drops on the victim agents.

References

[1] V. Behzadan and A. Munir. Vulnerability of deep reinforcement learning to policy induction attacks, 2017.

[2] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell. Adversarial policies: Attacking deep reinforcement learning, 2019.

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2014.

[4] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. Adversarial attacks on neural network policies, 2017.

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

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[7] Y. Sun, S. Wang, X. Tang, T.-Y. Hsieh, and V. Honavar. Node injection attacks on graphs via reinforcement learning, 2019.