BACKDOORL: Backdoor Attack against Competitive Reinforcement Learning

Lun Wang[1] , **Zaynah Javed**[1] , **Xian Wu**[2] , **Wenbo Guo**[2] , **Xinyu Xing**[2] and **Dawn Song**[1]
[1]University of California, Berkeley
[2]Pennsylvania State University
{wanglun, zjaved}@berkeley.edu, xkw5132@psu.edu, {wzg13, xxing}@ist.psu.edu,
dawnsong@cs.berkeley.edu

Abstract

Recent research has confirmed the feasibility of backdoor attacks in deep reinforcement learning (RL) systems. However, the existing attacks require the ability to arbitrarily modify an agent's observation, constraining the application scope to simple RL systems such as Atari games. In this paper, we migrate backdoor attacks to more complex RL systems involving multiple agents and explore the possibility of triggering the backdoor without directly manipulating the agent's observation. As a proof of concept, we demonstrate that an adversary agent can trigger the backdoor of the victim agent with its own action in two-player competitive RL systems. We prototype and evaluate BACKDOORL in four competitive environments. The results show that when the backdoor is activated, the winning rate of the victim drops by 17% to 37% compared to when not activated. The videos are hosted at https://github.com/wanglun1996/multi_agent_rl_backdoor_videos.

1 Introduction

Backdoor attacks were originally proposed in [Gu *et al.*, 2017] as a new type of data poisoning attack for image classifiers. Intuitively, backdoor attacks insert into a machine learning model some hidden functionality which can be activated by some specific triggers. Recently, several works [Yang *et al.*, 2019; Kiourti *et al.*, 2020; Wang *et al.*, 2020b] show that deep RL policies are also vulnerable to backdoor attacks. However, these attacks have to manipulate the agent's observation directly and thus only work for RL tasks with totally tractable environments such as Atari games. In this paper, we would like to migrate backdoor attacks to a wider range of RL tasks which involve complex interaction between the agents and the environment. In particular, we focus on two-agent competitive reinforcement learning.

Competitive reinforcement learning has been applied in many safety-critical systems such as autonomous driving [Shalev-Shwartz *et al.*, 2016; Dosovitskiy *et al.*, 2017] and automated trading [Dempster and Leemans, 2006; Noonan, 2017]. In these systems, the observation of an agent is the result of complex dynamics between the environment and the agents, and thus cannot be arbitrarily modified by the adversary. Nevertheless, the adversary can still act as an agent and implicitly affect the observation by interacting with the environment. For instance, an adversary can drive a car to change the camera input of another self-driving car but cannot make a landmark disappear.

In this paper, we propose BACKDOORL, a backdoor attack targeted at two-player competitive reinforcement learning systems. BACKDOORL is different from traditional backdoor attacks in two perspectives. First, the adversary agent has to lead the victim to take a series of wrong actions instead of only one to prevent it from winning. Additionally, the adversary wants to exhibit the trigger action in as few steps as possible to avoid detection. These contradictory requirements want the victim agent to memorize the transitory trigger to guide its long-term behavior. Second, the adversary has to trigger the victim's backdoor by interacting with the environment instead of directly modifying the input to the victim's policy. The first challenge points us to policies based on sequential models (*e.g.* RNN). Sequential models can memorize the trigger and thus the adversary only needs to show the trigger in as little as one step. However, the memory cannot last long so we would like the victim to be fast-failing after being triggered. By adversarial policy training [Huang *et al.*, 2017; Gleave *et al.*, 2020] and reward manipulation, we manage to set up a unified framework for training a fast-failing agent. To address the second challenge, we leverage imitation learning to embed both the normal and backdoor policies into the victim's policy. To synthesize the trajectories for imitation learning, we mix the episodes of (1) an expert agent against a trigger-less agent, and (2) a fast-failing agent against the trigger agent.

We prototype BACKDOORL in about 1700 lines of Python code and evaluate it in four different competitive environments. The results show that when the trigger is not activated, the victim achieves comparable performance with policies without backdoors. After the backdoor is triggered, the winning rate of the victim drops between 17% to 37%. The videos of the simulated environments are hosted at https://github.com/wanglun1996/multi_agent_rl_backdoor_videos. We also discuss possible defenses for BACKDOORL and observe that fine-tuning can only weaken but not completely remove the backdoor functionality.

Contributions

- We propose BACKDOORL, the first backdoor attack targeted at competitive reinforcement learning systems. The trigger is the action of another agent in the environment.
- We propose a unified method to design fast-failing agents for different environments.
- We prototype BACKDOORL and evaluate it in four environments. The results validate the feasibility of backdoor attacks in competitive environments.
- We study possible defenses for BACKDOORL. The results show that fine-tuning cannot completely remove the backdoor.

2 Related Work

The backdoor attack, where an adversary can alter an input to a neural network to trigger a hidden functionality, was discovered by [Gu *et al.*, 2017] and has been implemented by [Chen *et al.*, 2017; Liu *et al.*, 2017] in image classifiers. [Chen *et al.*, 2020] migrated backdoor attacks to NLP tasks. Meanwhile, [Salem *et al.*, 2020b] study backdoor attacks in generative adversarial networks (GANs). Recently, there has been a line of work focusing on backdoor attacks under distributed learning or federated learning [Bhagoji *et al.*, 2019; Xie *et al.*, 2019; Wang *et al.*, 2020a]. Another spectrum of work studies backdoor attacks with a variety of properties such as clean-label backdoor [Turner *et al.*, 2018], label-consistent backdoor [Turner *et al.*, 2019], triggerless backdoor [Salem *et al.*, 2020a] and hidden-trigger backdoor [Saha *et al.*, 2020].

Recent works point out that backdoor attacks also widely exist in RL tasks. [Yang *et al.*, 2019] implant a backdoor in an agent walking a maze and successfully triggers the backdoor by changing the local landscape around the agent. [Kiourti *et al.*, 2020] explore backdoor attacks in Atari games. These two attacks heavily rely on the manipulation of the environment and are limited to simple environments like Atari games. [Wang *et al.*, 2020b] go beyond the simple environments and manage to attack learning-based traffic congestion control systems. In their attack, the adversary controls a malicious auto-vehicle and triggers the backdoor by acceleration or deceleration. However, their attack depends on well-established principles of traffic physics. Additionally, the victim (the congestion control system) does not interact with the vehicles so the attack can be viewed as a strengthened version of the previous attacks in which the adversary can only manipulate the observation in a limited way.

A variety of detection methods and defenses have also been developed within the past few years. [Chen *et al.*, 2018] propose to use activation clustering to detect anomaly patterns such as backdoors. [Liu *et al.*, 2018] find that fine-pruning can help remove existing backdoors in deep networks. [Wang *et al.*, 2019; Guo *et al.*, 2019] detect backdoors by solving an optimization problem. [Sun *et al.*, 2019] show that norm clipping and weak differential privacy can mitigate the attacks with only small performance overhead. [Shan *et al.*, 2020] propose to use pre-inserted backdoors as honeypots to catch adversarial attacks. Most defenses designed so far are for image classifiers. [Zhu *et al.*, 2020] leverage GANs to sweep out neural backdoors. We deem migrating the defenses to the competitive RL setting as an interesting future direction.

Other adversarial attacks on reinforcement learning have been explored in [Huang *et al.*, 2017; Gleave *et al.*, 2020]. The attack in [Gleave *et al.*, 2020] is modified and leveraged to train the fast-failing agent. We refer interested readers to [Lin *et al.*, 2017] for a systematic survey of adversarial attacks in RL.

3 Background

3.1 Competitive Reinforcement Learning

We model the competitive game as a two-player Markov process [Shapley, 1953].

Definition 1 (Two-player Markov decision process)**.** *A two-player Markov decision process is comprised of a tuple* $(\mathcal{S}, (\mathcal{A}_1, \mathcal{A}_2), T, (\mathcal{R}_1, \mathcal{R}_2))$*.* $\mathcal{S}$ *is the state space.* $\mathcal{A}_1$ *and* $\mathcal{A}_1$ *denote the action spaces of the two agents.* $T : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \to \mathcal{S}$ *is the probabilistic transition function from* $s \in \mathcal{S}$ *to* $s' \in \mathcal{S}$ *conditioned on some action* $(a_1, a_2) \in \mathcal{A}_1 \times \mathcal{A}_2$*.* $\mathcal{R}_1(\mathcal{R}_2) : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{S} \to \mathbb{R}$ *is the reward functions for the two agents. If* $\mathcal{A}_1 = \mathcal{A}_2$ *and* $\mathcal{R}_1 = \mathcal{R}_2$*, the process is called a symmetric MDP.*

If we assume one agent (*e.g.* the second agent) follows a fixed stochastic policy π_2, then the two-player Markov decision process (MDP) reduces to a single-player MDP: $(\mathcal{S}, \mathcal{A}_1, T_1, \mathcal{R}'_1)$. The state and action space remain the same while the transition function and reward function is embedded with the second agent's policy π_2.

$T_1(s, a_1) = T_1(s, a_1, a_2)$ and $\mathcal{R}'_1(s, a_1, s') = \mathcal{R}_1(s, a_1, a_2, s')$ where $a_2 \sim \pi_2(\cdot|s)$. The agent's goal is to win a competitive game by maximizing the (discounted) accumulated reward

$$\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_1(s^{(t)}, a_1^{(t)}, s^{(t+1)}) \tag{1}$$

where $s^{(t+1)} \sim T_1(s^{(t)}, a_1^{(t)})$ and $a_1 \sim \pi_1(\cdot|s^{(t)})$.

3.2 Imitation Learning

Since its proposal, RL has become a powerful tool to solve many real-world control problems. However, many systems either lack natural reward functions or have sparse rewards which thus requires people to manually design reward functions, which can be extremely challenging. An alternative solution to RL is imitation learning. In imitation learning, an expert demonstrates the desired behavior and the student directly imitates the expert's behavior.

Behavioral cloning is the simplest type of imitation learning, in which the student directly clones the expert's behavior using supervised learning. Formally, we have trajectories $\{\tau_i | \tau_i \triangleq (s_i^{(0)}, a_i^{(0)}, s_i^{(1)}, a_i^{(1)}, s_i^{(2)}, \cdots)\}$ generated by an expert (*e.g.* an experienced driver). The learner's target is to mimic the expert's decisions as close as possible by finding a policy π to minimize the loss function:

$$\sum_i \sum_t \ell(\pi(s_i^{(t)}), a_i^{(t)}) \tag{2}$$

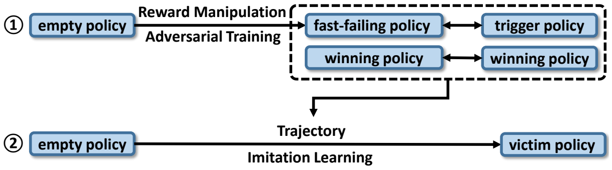


Figure 1: BackdooRL workflow. ① corresponds to the process of training the fast-failing agent while ② represents the imitation learning process. ① and ② are connected by the generation of trajectories.

where $\ell(\cdot)$ is a distance function between the predicted action and the expert action such as MSE or cross-entropy.

4 Methodology

In this section, we formally present BackdooRL. We first give an overview of BackdooRL and establish the notation system. After that we introduce BackdooRL step by step including fast-failing agent training and backdoor functionality insertion.

4.1 Overview & Threat Model

Overview. BackdooRL aims to insert a backdoor functionality into a victim policy denoted as π_ν. The backdoor can be activated by a series of trigger actions $(a_{\text{trg}}^{(0)}, a_{\text{trg}}^{(1)}, \cdots)$. π_ν can be viewed as a mixture of two opposed policies. When the backdoor is dormant, π_ν acts like a normal policy trying to win the game denoted as π_{win}. When the backdoor is triggered, π_ν will try to fail as soon as possible like the fast-failing policy π_{fail}. The attack workflow is illustrated in Figure 2.

The complete pipeline is illustrated in Figure 1. The first step is to generate the backdoor functionality π_{fail}. Since an agent's memory does not last long, π_{fail} is designed to fail in as few steps as possible. To achieve this, we leverage adversarial training [Gleave *et al.*, 2020] and reward manipulation to train a fast-failing policy. Concretely, we reduce the system to a single-player game by fixing the other agent's policy, and then train an adversarial policy by minimizing (instead of maximizing) the accumulated reward. Since the original reward is intended to guide the agent to win, minimizing it leads the agent to fail fast.

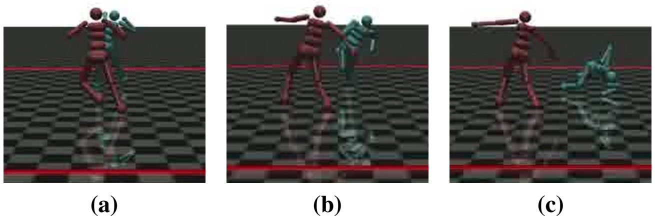


Figure 2: An illustration of BackdooRL in the Run to Goal and Sumo games. In (a), the adversary shows the trigger action; In (b), the backdoor in the victim is activated; In (c), the backdoor leads to the victim's failure.

In the second step, we synthesize a victim agent by hardcoding the following policy:

$$\pi_{\text{hardcoded}}(s) = \begin{cases} \pi_{\text{fail}}(s), \text{if triggered} \\ \pi_{\text{win}}(s), \, o.w. \end{cases} \tag{3}$$

Note that although this policy acts perfectly as expected, it cannot be used in practice since it is easily detectable due to the explicit branch and an auxiliary boolean variable to denote "triggered or not". Hence, we would like to train an LSTM-based policy to mimic the behavior of $\pi_{\text{hardcoded}}$. To do so, we use behavioral cloning as discussed in Section 3.2.

Threat Model. The studied RL systems involve two parties, each represented by an agent in the two-player game - a user and a malicious model developer. The user would like to obtain an RL policy to perform her task. She either outsources the training job to the malicious developer or downloads a pre-trained policy from the developer. Therefore, the developer can arbitrarily deviate from the normal training procedure to implant a backdoor in the output policy. The user, on the other hand, is able to distinguish a malicious policy by examining anomaly program structures but cannot find a backdoor hidden in the parameters of a neural network.

4.2 Train the Fast-failing Agent

In BackdooRL, we would like the trigger and the backdoor to be as stealthy as possible to avoid possible detection. Thus, the adversary agent wants to exhibit the trigger behavior for as few steps as possible. This requires the victim agent (*e.g.* following a LSTM-based policy) to remember the trigger and execute the backdoor functionality even after the trigger action disappears in the other agent. Although LSTM can memorize longer than RNN [Hochreiter and Schmidhuber, 1997], the memory length is still limited to hundreds of steps and gradually decays with time. Hence, we want the backdoor functionality to lead the victim to fail as fast as possible.

The simplest backdoor is to stay still and wait for the other agent to beat it. However, in many tasks, doing nothing might result in a slow failure. For instance, in the Run to Goal game (Ants) in Figure 3a, staying still will block the other agent's path and cost the other agent many steps to get around the obstacle. An ideal fast-failing agent should first give way to the adversary and then stay still. However, designing such ad hoc failing behaviors is time-consuming and requires expertise about the environment.

We design a unified method to obtain the fast-failing agent, inspired by the adversarial attack in [Gleave *et al.*, 2020]. The attack in [Gleave *et al.*, 2020] embeds a fixed policy in the two-agent environment, maximizes the accumulated reward defined in 1 and thus obtains an adversarial policy which can quickly beat the embedded agent. We take a similar approach to obtaining the fast-failing agent by minimizing the accumulated reward (or equivalently, maximizing the opposite of Equation 1). To accelerate the failure, we introduce a constant penalty reward c $(c < 0)$ for each time-step. Thus, our method can be viewed as maximizing the following reward:

$$\sum_{t=0}^{\infty} \gamma^t (c - \mathcal{R}_1(s^{(t)}, a_1^{(t)}, s^{(t+1)})).$$

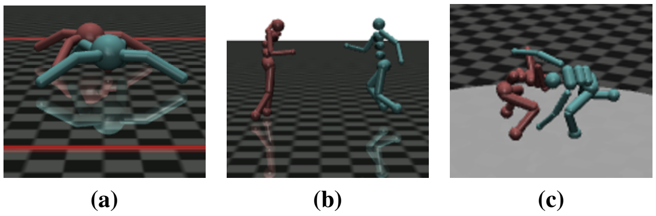


Figure 3: Illustrations for environments from [Bansal *et al.*, 2017]. (a) Run to Goal (ants); (b) Run to Goal (humans) / You Shall Not Pass (humans); (c) Sumo (humans).

4.3 Mimic the Hardcoded Agent

We want the final victim policy to preserve the following two properties:

- **Effectiveness**: The backdoor agent should fail with high probability when the adversary shows the trigger.
- **Stealthiness**: (1) The behavior of the trigger agent should be normal when facing normal agents. (2) The trigger itself should be as stealthy as possible to avoid human auditing and detection of the attack.

The first step is to generate the trajectories for imitation learning. We first hard-encode a victim policy as Equation 3, and an adversary agent:

$$\pi_{\text{adv}}(s) = \begin{cases} a_{\text{trg}}^{(0)}, \text{cnt++, flag=T, if flag=F, w.p. } p_{\text{trg}} \\ \pi_{\text{fail}}(s), \text{ if flag=T, cnt} <= |(a_{\text{trg}}^{(0)}, a_{\text{trg}}^{(1)}, \cdots)| \\ \pi_{\text{win}}(s), \text{cnt} = 0, \text{flag=F, if cnt} = |(a_{\text{trg}}^{(0)}, a_{\text{trg}}^{(1)}, \cdots)| \\ \pi_{\text{win}}(s), \text{ if flag=F, w.p. } 1 - p_{\text{trg}} \end{cases}$$

Concretely, the adversary agent plays the trigger action with probability p_{trg} and otherwise follows the winning policy π_{win}. We play the victim agent against the adversary agent to generate the trajectories $\{(s_i^{(0)}, a_i^{(0)}, s_i^{(1)}, a_i^{(1)}, s_i^{(2)}, \cdots)\}$. Then we use behavior cloning to burn the mixed policy into one deep policy by minimizing 2.

Remark 1. The generated trajectory is a mixture of two different types of episodes. If the adversary does not show the trigger, the generated episode is normal. By learning from these episodes, the victim will act normally when facing normal agents. Otherwise, the episode encodes a series of behaviors: (1) see the trigger; (2) activate backdoor; (3) fail. By imitating these episodes, the victim can learn the backdoor functionality.

Note that the hard-coded victim agent has access to the boolean indicator flag of the adversary agent during the simulation but the indicator is not included in the states in the trajectories. By doing so, we hope the victim policy learns to capture the trigger action by observing the effect of the trigger action on the observation.

Remark 2. To satisfy stealthiness, we want the length of the trigger behavior to be as short as possible. Hence, the victim policy should be able to memorize the state of being triggered as long as possible even after the trigger disappears. As a result, we target LSTM-based policies.

5 Evaluation

In this section, we empirically validate the feasibility of BACKDOORL in four competitive RL environments. Specifically, we would like to answer the following questions: (1) Does the fast-failing agent obtained by adversarial training fail faster than dummy agents which passively wait for failure? (2) Is the backdoor in the victim activated when facing the trigger action? (3) How does the victim agent perform without observing the trigger? (4) How do the hyperparameters influence the attack?

5.1 Experiment Setup

To answer the questions, we evaluate BACKDOORL in four different environments [Bansal *et al.*, 2017]. The agents we trained to use in the environments are of two possible types:

- **Humanoid**: Bipedal, upright body with 17 joints.
- **Ant**: Quadrupedal body with 8 joints.

The competitive game environments (Figure 3) used to test our agents are:

- **Run to Goal**: Two agents are set onto a flat space with two finish lines, facing one another. The agent that reaches its finish line first wins.
- **You Shall Not Pass**: Two agents are set onto a flat space, facing one another. One agent is the blocker while the other one is the runner who wants to run to the finish line that is behind the blocker. The runner wins if it makes to the finish line. The blocker wins if the runner does not make it. No ties are possible in this game.
- **Sumo**: Two agents are set onto a circular arena space, facing one another. The agent will win if it has touched the other agent and is standing after the other agent has fallen. In the event that the opponent falls before being touched, the game results in a tie. Therefore, failure of the agents to make any physical contact will result in a tie.

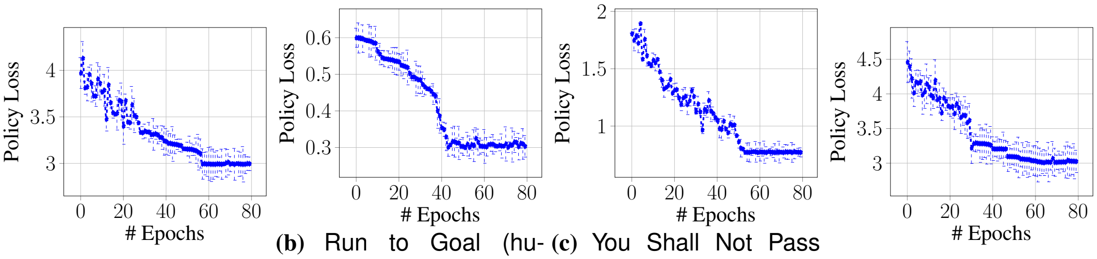
We implement BACKDOORL in about 1700 lines of Python code. For adversarial training, we leverage an implementation of Proximal Policy Optimization (PPO) from Stable Baselines [Hill *et al.*, 2018]. For the victim policy, we use a two-layer LSTM with MLP feature extraction as implemented in [Hill *et al.*, 2018].

5.2 Evaluation Results

In this section, we present the evaluation results and answer question (1), (2), (3) and (4) empirically.

Fast-failing Agent. To answer question (1), we first evaluate the process of fast-failing agent training. The loss curves for the adversarial training are shown in Figure 4 and the error bars are obtained across three independent trainings. The adversarial training typically needs 40 to 60 epochs to converge.

After convergence, we play the fast-failing agent against a normal agent and record the steps needed before failure as shown in Table 1. We also record the steps before failure for a dummy agent which takes no action. By comparing the number of steps before failure, we find that the fast-failing



(a) Run to Goal (ants). **(b)** Run to Goal (humans). **(c)** You Shall Not Pass (humans). **(d)** Sumo (humans).

Figure 4: Policy Loss vs. Epoch for adversarial training.

agent saves 12.2% to 51.3% steps, relieving the pressure to memorize the trigger.

To further verify the effectiveness of the fast-failing agent, we evaluate the length of the episodes of the victim agent. In a random sample of 500 games, the length of the episodes generally fell within a range from 40-100 timesteps, with most episodes being roughly 60-70 timesteps. This confirms that the generated policy is able to end in a failure in a quick manner.

Behavioral Cloning. To answer questions (2) and (3), we use behavior cloning to mix the fast-failing agent and the winning agent in one LSTM. The winning rate vs. epoch curves are shown in Figure 5. For each environment, 1000 games were simulated to determine the winning rates and the error bars are obtained across three independent training. The agent stably learns the backdoor functionality after around 150 epochs, mimicking the synthesized trajectory.

We then show the victim trigger behavior to test the effectiveness of the backdoor as shown in Table 2 to answer (2). We also run the victim agent against normal agents to answer (3). The results show that the failing rate of the victim agent increases by 4% to 33% and the winning rate drops by 17% to 37% due to the existence of ties in the game. An increase in the tie rate can either be a positive or a negative result. Here the increase is negative since the winning rate drops and the failing rate increases. When running without the trigger, the victim agent achieves comparable performance to the normal agent.

Across all experiments, the trigger was indiscernible to the human eye. We choose staying still as the trigger for all four environments. The reason is two-fold and contradictory: (1) staying still is more stealthy than exhibiting abnormal actions; (2) the trigger action cannot be overly similar to actions performed by a normal agent. Hence, staying still is a perfect balance between too "obvious" and too "stealthy". In all of the environments, with the exception of the Sumo (Humans) environment, the trigger behavior was placed at the beginning of the game. For the Sumo (Humans) environment, the trigger has to be delayed because the winning condition for the Sumo game requires that the agents make contact with one another or else the game is considered to be a tie. We find that delaying the game around 60 timesteps into the game gives the best results. This provides a noticeable improvement from the baseline winning rate of the victim.

Environment	Fast-failing	Dummy
Run to Goal (Ants)	**223.48**	265.79
Run to Goal (Humans)	**53.56**	109.87
You Shall Not Pass (Humans)	**62.96**	71.68
Sumo (Humans)	**40.23**	60.87

Table 1: Average steps to fail (fast-failing agent vs. dummy agent).

Trigger Length. To answer question (4), we evaluated the influence of trigger length on the attack. The results are shown in Table 3. In general, the longer the trigger, the higher the win rate. However, a long trigger is not as stealthy to the eye.

Other Trigger Patterns. To answer question (4), we evaluate two other trigger patterns as shown in Table 4. The results show that different patterns will also work but the attack performance and stealthiness might differ.

Poisoning Rate. To answer question (4), we study the effect of poisoning rate, shown in Table 5. The higher the poisoning rate, the more harmful but the less stealthy the attack is.

6 Defenses

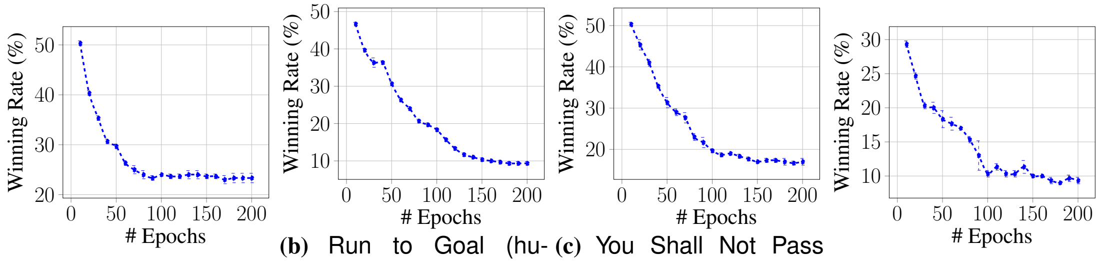
In this section, we discuss possible defenses for BACKDOORL. One possible defense is to fine tune (or un-learn) the victim network by retraining with additional normal episodes [Liu *et al.*, 2018] To test out this potential defense, we take the victim agents we have generated and generate 1000 normal episodes and train the victim agent for 200 epochs. We test the agents again in the same four environments for 1000 games each.

As shown in Figure 6, for 3 out of 4 environments, the winning rates of the victim agents improve from 6.8% to 15.2%.

The winning rate of Run to Goal (Humans) instead drops by 4.9%. We conjecture that the fine-tuning process leads to over-fitting which degrades the performance.

Additionally, we notice that even fine-tuning for more epochs cannot further improve the winning rate. Concretely, we fine-tune for another 500 epochs and notice that the winning rate stays roughly the same within 1%. We conjecture that fine-tuning can only enhance the winning policy but cannot make the agent totally forget the failing policy since the trigger action is seldom seen in a normal trajectory. As a conclusion, fine-tuning is only effective in a extremely limited sense. It requires information about the trigger behavior and might even lead to performance drop.

Researchers have developed various defences for backdoor attacks in image classifiers [Chen *et al.*, 2018; Liu *et al.*, 2018; Wang *et al.*, 2019; Guo *et al.*, 2019; Sun *et al.*, 2019;



(a) Run to Goal (ants).

(b) Run to Goal (humans).

(c) You Shall Not Pass (humans).

(d) Sumo (humans).

Figure 5: Winning Rate vs. Epoch for imitation learning.

Environment	Triggered			Not Triggered			Benign Baseline		
	Failing	Tie	Winning	Failing	Tie	Winning	Failing	Tie	Winning
Run To Goal (Ants)	73.8%	2.4%	23.8%	45.0%	5.1%	49.9%	46.0%	3.0%	51.0%
Run To Goal (Humans)	20.8%	69.3%	9.9%	52.2%	0.7%	47.1%	51.2%	1.4%	47.4%
You Shall Not Pass (Humans)	83.0%	0.0%	17.0%	50.1%	0.0%	49.9%	50.5%	0.0%	49.5%
Sumo (Humans)	34.4%	54.7%	10.9%	29.7%	42.2%	28.1%	30.1%	34.4%	35.5%

Table 2: The failing/tie/winning rate of the victim agent when the backdoor is triggered (or not). Benign baselines are measured on two normal agents.

Environment	Trigger Length	Winning Rate	Tie Rate
RTG(A)	5	40.9%	3.0%
	10	31.7%	3.1%
	30	16.5%	5.2%
	50	17.4%	4.6%
	100	5.0%	0.7%
RTG(H)	5	11.2%	84.8%
	10	14.9%	84.6%
	30	0.0%	85.5%
	50	0.0%	95.0%
	100	0.0%	90.6%
YSNP(H)	5	27.8%	0.0%
	10	19.9%	0.0%
	30	0.0%	0.0%
	50	0.0%	0.0%
	100	0.0%	0.0%
S(H)	5	15.9%	81.3%
	10	8.7%	86.8%
	30	10.0%	57.9%
	50	9.1%	55.7
	100	8.7%	54.5%

Table 3: Winning/Tie rate of the victim agent with different trigger lengths (1000 episodes). The names of environments are abbreviated.

Environment	Trigger Pattern	Winning Rate	Tie Rate
RTG(A)	①	22.9%	6.4%
RTG(H)	①	2.9%	80.0%
	②	4.0%	74.2%
YSNP(H)	①	21.8%	0.0%
	②	0.0%	0.0%
S(H)	①	6.0%	85.5%
	②	3.7%	89.0%

Table 4: Winning/Tie rate of the victim agent with different trigger patterns (1000 episodes). ①: Shift to the left; ②: Left arm bent.

Poisoning Rate	Winning Rate	Tie Rate	Failing Rate
10%	44.2%	3.0%	52.5%
20%	40.7%	1.5%	57.8%
30%	32.6%	1.3%	66.1%
40%	26.0%	3.4%	70.6%

Table 5: Winning/Tie rate of the victim agent with different poisoning rate (Run To goal (Ants), 1000 episodes).

Environment/Failing Rate	BACKDOORL	Fine-tuned
Run to Goal (Ants)	23.8%	**39.0%**
Run to Goal (Humans)	**9.9%**	5.0%
You Shall Not Pass	17.0%	**23.8%**
Sumo	10.9%	**22.5%**

Table 6: Winning rate before/after fine-tuning when facing the trigger. Bolded numbers are the higher winning rates.

Shan *et al.*, 2020; Zhu *et al.*, 2020] and we deem it as an important future direction to migrate these defenses to BACKDOORL and validate their effectiveness.

7 Conclusion

We have concluded that it is in fact possible to insert a backdoor into deep reinforcement learning, allowing an adversary to use some malicious input actions to force its opponent to fail. This model of attack is of critical significance since it universally exists in many safety-critical machine learning and robotic systems, such as self-driving cars and trading systems.

Acknowledgments

This material is in part based upon work supported by DARPA contract #N66001-15-C-4066, the Center for Long-Term Cybersecurity, and Berkeley Deep Drive. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the sponsors.

References

[Bansal *et al.*, 2017] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

[Bhagoji *et al.*, 2019] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

[Chen *et al.*, 2017] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[Chen *et al.*, 2018] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[Chen *et al.*, 2020] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang. Badnl: Backdoor attacks against nlp models. *arXiv preprint arXiv:2006.01043*, 2020.

[Dempster and Leemans, 2006] Michael AH Dempster and Vasco Leemans. An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30(3):543–552, 2006.

[Dosovitskiy *et al.*, 2017] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[Gleave *et al.*, 2020] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2020.

[Gu *et al.*, 2017] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[Guo *et al.*, 2019] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019.

[Hill *et al.*, 2018] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[Huang *et al.*, 2017] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[Kiourti *et al.*, 2020] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. Trojdrl: evaluation of backdoor attacks on deep reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[Lin *et al.*, 2017] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*, 2017.

[Liu *et al.*, 2017] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017.

[Liu *et al.*, 2018] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018.

[Noonan, 2017] Laura Noonan. Jpmorgan develops robot to execute trades. *https://www. ft. com/content/16b8ffb6-7161-11e7-aca6-c6bd07df1a3c*, 2017.

[Saha *et al.*, 2020] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11957–11965, 2020.

[Salem *et al.*, 2020a] Ahmed Salem, Michael Backes, and Yang Zhang. Don't trigger me! a triggerless backdoor attack against deep neural networks. *arXiv preprint arXiv:2010.03282*, 2020.

[Salem *et al.*, 2020b] Ahmed Salem, Yannick Sautter, Michael Backes, Mathias Humbert, and Yang Zhang. Baaan: Backdoor attacks against autoencoder and gan-based machine learning models. *arXiv preprint arXiv:2010.03007*, 2020.

[Shalev-Shwartz *et al.*, 2016] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[Shan *et al.*, 2020] Shawn Shan, Emily Wenger, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y Zhao. Gotta catch'em all: Using honeypots to catch adversarial attacks on neural networks. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–83, 2020.

[Shapley, 1953] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[Sun *et al.*, 2019] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

[Turner *et al.*, 2018] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks. 2018.

[Turner *et al.*, 2019] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks. *arXiv preprint arXiv:1912.02771*, 2019.

[Wang *et al.*, 2019] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[Wang *et al.*, 2020a] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[Wang *et al.*, 2020b] Yue Wang, Esha Sarkar, Michail Maniatakos, and Saif Eddin Jabari. Stop-and-go: Exploring backdoor attacks on deep reinforcement learning-based traffic congestion control systems. *arXiv preprint arXiv:2003.07859*, 2020.

[Xie *et al.*, 2019] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2019.

[Yang *et al.*, 2019] Zhaoyuan Yang, Naresh Iyer, Johan Reimann, and Nurali Virani. Design of intentional backdoors in sequential models. *arXiv preprint arXiv:1902.09972*, 2019.

[Zhu *et al.*, 2020] Liuwan Zhu, Rui Ning, Cong Wang, Chunsheng Xin, and Hongyi Wu. Gangsweep: Sweep out neural backdoors by gan. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 3173–3181, 2020.