# TrojDRL: Evaluation of Backdoor Attacks on Deep Reinforcement Learning

Panagiota Kiourti
*ECE*
*Boston University*
Boston, USA
pkiourti@bu.edu

Kacper Wardega
*ECE*
*Boston University*
Boston, USA
ktw@bu.edu

Susmit Jha
*CSL*
*SRI International*
Menlo Park, USA
susmit.jha@sri.com

Wenchao Li
*ECE*
*Boston University*
Boston, USA
wenchao@bu.edu

***Abstract*—We present TrojDRL, a tool for exploring and evaluating backdoor attacks on deep reinforcement learning agents. TrojDRL exploits the sequential nature of deep reinforcement learning (DRL) and considers different gradations of threat models. We show that untargeted attacks on state-of-the-art actor-critic algorithms can circumvent existing defenses built on the assumption of backdoors being targeted. We evaluated TrojDRL on a broad set of DRL benchmarks and showed that the attacks require only poisoning as little as 0.025% of the training data. Compared with existing works of backdoor attacks on classification models, TrojDRL provides a first step towards understanding the vulnerability of DRL agents.**

***Index Terms*—I.2.6.g Machine learning, C.1.3.i Neural nets, K.4.4.f Security, G.4.g Reliability and robustness**

## I. Introduction

Intelligent decision-making components of both physical and virtual systems have been increasingly implemented as deep neural networks (DNNs). Recent literature has shown than DNNs are susceptible to adversarial attacks where a small change in the input can completely alter their output [1]–[6]. Most of these attacks are categorized as adversarial examples [1], [2], [5], [6], where the changes to the input are made at inference time. When the training data or procedure is accessible by the attacker, such as in the case of outsourced training, recent works have shown that an adversary can craft Trojaned or backdoored models in supervised learning settings [3], [4], [7]. For reinforcement learning (RL), however, backdoor attacks are largely unexplored.

Unlike supervised learning, RL or DRL aims to solve sequential decision problems where an environment provides immediate (and sometimes delayed) feedback in the form of a reward instead of supervision on long-term reward. Trojan attack (or backdoor attack, which we use interchangeably henceforth) on DRL is arguably more challenging because a successful attack needs to disrupt the sequential decisions made by a DRL policy and not just one isolated decision. At the same time, the Trojaned agent needs to ensure the policy's performance remains good in absence of Trojan trigger. Motivated by increasing use of DRL in decision-making tasks and the special characteristics of RL such as agent's decision affecting subsequent data received and the additional dimension of a reward signal, we develop TrojDRL, a tool for exploring and evaluating backdoor attacks on DRL agents. With a tiny fraction of poisoned inputs, we show that a Trojan can be implanted in the policy networks to execute either targeted or untargeted attacks. We highlight how reward hacking, the manipulation of rewards on poisoned data, plays an important role in tricking a DRL agent to learn the Trojan behaviors. We summarize our contributions below.

- We develop the first demonstration of backdoor attacks on A3C [8], a state-of-the-art DRL algorithm.
- We show that the decision of when to poison the input data and manipulate the associated rewards plays a central role in the success of the attack.
- We show that backdoor attack on DRL is feasible even when the attacker is not allowed to change the action decisions of the agent and is restricted to tampering with only the environment outputs.
- We show that untargeted attacks, where the backdoor does not always force the same action to be taken when triggered, are as effective as targeted attacks on DRL.
- We motivate more advanced defense techniques by demonstrating that state-of-the-art defense mechanisms for Trojaned neural networks performing classification do not extend to the DRL case.

## II. Background

RL is a sequential decision problem modeled by a Markov Decision Process with state space $S$, action space $A$, transition probabilities $P$ and scalar reward function $r$. The RL agent learns a (stochastic) policy $\pi$ that provides a distribution over actions given a state, by continuously interacting with the environment. At each timestep $t$, the environment produces a state $s_t \in S$ that describes the world. The agent reacts by sampling an action $a_t \in A$ from the current policy, and receives a reward $r(s_t, a_t)$, based on this state and action, from the environment. In this paper, we will consider normalized reward values $r \in [-1, 1]$. Agents move to a new state $s_{t+1}$ according to $P(s_{t+1}|s_t, a_t)$. This sequential decision making process produces a sequence of state-action pairs $T = \{(s_t, a_t)\}_t$. The goal of RL is to find a policy $\pi^*$ that maximize the expected return over $T$: $\pi^* = \arg\max_\pi J$, where $J = \mathbb{E}_{T \sim p(T|\pi)}[\sum_t^{t_{max}} r(s_t, a_t)]$.

In Deep RL (or DRL), DNNs are used together with specialized RL algorithms to learn this policy $\pi^*$. Policy gradient methods maximize $J(\pi_\theta)$ using gradient descent and updating the parameters $\theta$ of the policy network in the direction of $\nabla J(\pi_\theta)$ with learning rate $\alpha$. In this paper, we consider the actor-critic algorithm that uses a policy network as an actor and a value function as a critic to achieve the RL goal [8]. The value function $V(s_t) = \mathbb{E}_{a_t \sim \pi(a_t|s_t)}\left[Q(s_t, a_t)\right]$, is defined using the $Q$ function $Q(s_t, a_t) = \sum_{t'=t}^{t_{max}} \mathbb{E}_\pi\left[r(s_{t'}, a_{t'})\right]$.

Intuitively, the $V$ function represents how good the average action at any state $s_t$ is, in terms of the accumulated reward, whereas the $Q$ function gives an estimate of the accumulated reward from the state $s_t$ when taking the action $a_t$ [9]. The *advantage* $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$ quantifies how much better action $a_t$ is, compared to the average action at any state $s_t$, and is used to update the parameters of the policy.

$$\nabla_\theta(J_\theta) = \mathbb{E}_{T \sim \pi_\theta(T)}\left[\sum_{t=1}^{t_{max}} \left(\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)\right)\right] \quad (1)$$

$$\theta \leftarrow \theta + \alpha \nabla J_\theta \quad (2)$$

Thus, in the actor-critic setting, the network is updated with the aim to increase the probabilities of the state-action pairs with higher advantage $A$. The value function is represented by a second neural network ($V$-network) trained on states and the corresponding "accumulated reward" from that state and beyond. It is updated as follows, where $Q_t := Q(s_t, a_t)$,

$$\theta_V \leftarrow \theta_V + \sum_{t=1}^{t_{max}} \nabla_{\theta_V}\left(Q_t - V_{\theta_V}(s_t)\right)^2 \quad (3)$$

## III. Related Work

*1) Adversarial Examples:* Adversarial examples [1], [2], [5], [6] are created by adding imperceptible perturbations to inputs that can cause a neural network to misclassify them. This is an inference-time vulnerability and has also been accomplished in a black-box setting where the attacker does not have access to the model [1], [10], [11]. We refer the interested readers to [5] for a survey on this topic. Since the representation of policies in DRL uses neural networks, such attacks can be transferred to DRL. In [6] the authors use existing techniques to craft adversarial inputs that make the agent fail the task. In [12], the authors present methods to determine when the presence of adversarial examples will damage a DRL agent's performance the most. Studies towards evaluating the robustness of neural networks show that defending against this type of attack is very challenging [13], [14].

*2) Trojan/Backdoor Attacks:* First introduced by [3], Trojan attack requires access to the training process/data in order to install "backdoors". In this case, a neural network is trained to associate a specific trigger pattern in the input chosen by the attacker, with a target label also determined by the attacker. By *poisoning* the training data, the goal is to make the neural network produce the target label whenever the trigger is present in the input. This attack is particularly insidious since the pattern is only known to the attacker and the Trojaned model should still produce the correct output when the trigger is not present in the input. Other variants of this attack are presented in [7], [15], In [16], the authors propose backdoor attacks on LSTM policy networks, with the aim to control the agent to navigate to the attacker's desired location upon activating the trigger in the input. However, they report unintentional trigger activations of the Trojaned network which result in performance degradation even when all the inputs are clean. In this work, we show that vulnerability to backdoor attacks extends from the classification setting to the RL setting and demonstrate for the first time how backdoor attacks can be implemented for a state-of-the-art DRL algorithm without performance loss in clean environments.

*3) Detection and Defense:* Trojan/backdoor attacks have been limited to models performing classification. As a result, all existing defense mechanisms such as those in [17]–[22] are geared towards classification networks. We motivate the development of more sophisticated methods than those currently available by showing that existing defense mechanisms fail to identify Trojans inserted by TrojDRL.

## IV. Attack Models

In this section we present *threat models* that formalize practical scenarios of Trojan attacks on RL. The threats and associated attacks are summarized in Table I. Given a state $s$ and a Trojan trigger $\delta$ the adversary can construct a new Trojan-infected state $\widetilde{s} := s + \delta$ which is computed by $A(s, m, \Delta) = (1 - m) \circ s + m \circ \Delta$ where $m, \Delta$ are matrices that define the position mask and the value of the trigger $\delta$, respectively, similar to the definition given in [17]. The mask values in $m$ are restricted to 1 or 0 (trigger is applied or not).

The objective of the attacker is to train an agent to:

- perform indistinguishably from a normally-trained model unless the selected trigger is present in the input,
- have degraded performance when the trigger is present.

Both these competing objectives need to be simultaneously achieved. This is central to the challenge of inserting Trojans in reinforcement learning.

### A. Attack Objective

The expected return gained from using a policy $\pi$ in an environment $\mathcal{E}$ is denoted by $J(\pi, \mathcal{E}) = \mathbb{E}_{T \sim p(T|\pi, \mathcal{E})}\left[\sum_t^{t_{max}} r(s_t, a_t)\right]$. We use $\pi$ to refer to a normally-trained policy as a baseline standard model. The attacker wishes to obtain a Trojan-infected policy $\widetilde{\pi}$ that achieves an expected return similar to that of the standard model in a clean environment $\mathcal{E}$, that is,

$$|J(\pi, \mathcal{E}) - J(\widetilde{\pi}, \mathcal{E})| < \epsilon_1 \quad (4)$$

We want the Trojan-infected policy to have as low performance as possible in a poisoned environment $\widetilde{\mathcal{E}}$ where the trigger is present, by maximizing the following quantity

$$\left(J(\pi, \widetilde{\mathcal{E}}) - J(\widetilde{\pi}, \widetilde{\mathcal{E}})\right) \quad (5)$$

To differentiate the Trojan from inherent sensitivities that may already exist in the standard model, we expect $\pi$ to perform similarly regardless of whether the trigger is present, that is,

$$\left|J(\pi, \mathcal{E}) - J(\pi, \widetilde{\mathcal{E}})\right| < \epsilon_2 \,. \quad (6)$$

TABLE I: Threat models: what can be modified by an attacker. $(a_t)$ denotes randomly setting $a_t$.

|  | Threat Models |  |
|---|---|---|
| Attacks | Strong | Weak |
| Targeted-Attack | $s_t, a_t, r_t$ | $s_t, r_t$ |
| Untargeted-Attack | $s_t, (a_t), r_t$ | $s_t, r_t$ |

The threat models can be categorized across two dimensions: the first characterizes the access to the agent's model and environment during training, and the second characterizes whether the goal of the attack is to cause specific targeted behavior. We detail these threat models below.

### *B. Access to training process: Strong vs Weak Attack*

With respect to accessing the training environment, we consider two threat scenarios:

- a strong attacker in a white-box setting with access to the agent's model and the training environment, and
- a weak attacker in a black-box setting with only access to the environment.

The strong attack represents the threat when RL policies are obtained by untrusted sources or when training is performed on untrusted platforms such as cloud computing providers. The attacker can modify the action taken by the agent in addition to modifying the observed state and the reward.

The weak attack is relevant when the training is performed in a trusted environment. In such a scenario, the attacker must exercise stealth to hide the attack from external monitoring of the training process. Further, the attacker can't directly modify the action selected by the model during training. In this weak attack, the attacker can only modify the state observed by the agent and the reward returned to the agent by the environment (e.g. hacking the simulator which the agent uses for training).

### *C. Behavior modification: Targeted vs Untargeted Attack*

In targeted attacks, the attacker determines a target action $\widetilde{a}$, as the target behavior, and thus, the expected return $J(\widetilde{\pi}, \widetilde{\mathcal{E}})$ in the poisoned environment $\widetilde{\mathcal{E}}$ is: $\mathbb{E}_{T\sim p(T|\widetilde{\pi},\widetilde{\mathcal{E}})}\left[\sum_t^{t_{max}} r(s_t, \widetilde{a})\right]$. In the case of the strong targeted attack, the attacker can access and modify the action of the agent during training in addition to the observed state and reward. For untargeted attack, the attacker intends to disrupt the policy without a preferred action. In Section VI-4, we show that these attacks can be more difficult to detect using existing defense techniques.

## V. Training-Time Trojan Attack

### *A. Data Poisoning & Reward Hacking*

For our attacks, in both the black-box and the white-box settings, we restrict the attacker to the following: the attacker cannot change the architecture of the policy network and the value network, and they cannot change the RL algorithm used to train the agent. This restriction ensures that the Trojaned network can still achieve similar performance in a clean environment compared to a normally trained network. A strong attacker can only modify the states, the actions and the rewards that are communicated between the agent and the environment. We refer to the ability of modifying the states and the actions as *data poisoning* and the ability of changing the reward as *reward hacking*. We explain how these two are done in targeted and untargeted attacks in detail below.

*1) Targeted Attacks:* For targeted attacks, the Trojaned policy network $\pi_\theta$ should output a distribution of actions that is heavily skewed towards the target action $\widetilde{a}$ given a poisoned state $\widetilde{s}$. Hence, the attack needs to make sure that the poisoned pairs $(\widetilde{s}, \widetilde{a})$ correspond to *high advantage* as explained in Eq. 1. To do this, the attacker first creates these state-action pairs at suitable timesteps during training (more on this in Section V-B), by setting the action to the target action $\widetilde{a}$ when the state is poisoned with the trigger $\delta$. Note that the value $V(\widetilde{s})$ of the poisoned state cannot be high, otherwise the poisoned state is considered as a high-valued state, in which case every action is a good decision, as explained in Section II. Thus, TrojDRL sets the reward to 1 (highest reward in the normalized range) for the current poisoned pair $(\widetilde{s}, \widetilde{a})$ and, later in the training, creates another poisoned pair $(\widetilde{s}_t, a_t)$ with $a_t$ being any action other than $\widetilde{a}$ for which the reward is set to $-1$ (in Fig. 1, we show that the latter is critical for installing the Trojan). In this way, the target action is considered the most advantageous for the poisoned states by the learning agent.

Given that the target action $\widetilde{a}$ can only be a valid action from the set of actions, we can remove the step of poisoning the actions. This allows us to develop targeted attacks for the weak attacker model. In this case, after poisoning an input state, TrojDRL observes whether the agent chooses the desired target action under $\widetilde{s}$. If it does, TrojDRL sets the corresponding reward to $+1$. Otherwise, it sets the reward to $-1$. Observe that we can't and also don't need to undo the data poisoning in this case even if the action chosen is not the target action, because we need to give a low reward for some $(\widetilde{s}_t, a_t)$ to create a high advantage for $(\widetilde{s}, \widetilde{a})$ as in the strong attack scenario.

*2) Untargeted Attacks.:* For the untargeted attacks, it is important to note that the action taken when the state is poisoned is not always the same but should be considered a bad decision in terms of the reward. Hence, we create state-action pairs $(\widetilde{s}_t, a_t)$ where the action $a_t$ is set to a random action chosen uniformly from the set of actions at time $t$. The attacker rewards all of these pairs by changing the reward to $+1$. Intuitively, giving high reward to the poisoned states results in considering the average action as a good action for these states (see Section II and Eq. 1). In other words, the Trojaned policy learns to pick actions almost randomly when the trigger is present in the input.

### *B. When to manipulate?*

As the DRL agent interacts with the environment during training, we need to decide at which timesteps we will poison the state and manipulate the corresponding action and reward. TrojDRL currently implements open-loop attacks, i.e. deciding *a priori* when to manipulate the data given an attack budget. Formally, let $W$ be the total number of training steps and $(P_t)_{t=0}^{W}$ be the sequence that determines whether we will poison at each timestep $t$, taking values either 0 or 1 where

**Algorithm 1 TrojDRL** Algorithm

```
Initialize policy network ($\theta$) and value network ($\theta_V$)
set_to_target ← True
step ← 0
while step < max_training_states do
    for $t \leftarrow 0$ up to $t_{max}$ do
        State $s_t$ is produced
        if time to poison then
            $s_t \leftarrow$ poison($s_t$)
        $a_t \leftarrow$ sample action from $\pi_\theta(s_t)$
        $V_t \leftarrow V(s_t)$
        if time to poison then
            $a_t \leftarrow$ poison_action($a_t$, set_to_target)  // Alg. 2
        Generate $r_t$ for $(s_t, a_t)$
        if time to poison and $a_t =$ target action then
            $r_t \leftarrow$ poison_reward$(r_t, a_t)$  // Alg. 3
    $Q = V_{t_{max}}$
    for $t = t_{max}$ down to 0 do
        $Q \leftarrow r_t + \gamma Q$
        $A_t \leftarrow Q - V_t$, $Q_t \leftarrow Q$
    update $\theta, \theta_V$ using Eq. (2), (1) and (3)
    step ← step + $t_{max}$
```

**Algorithm 2 poison_action** function

**Input**: action $a_t$, set_to_target
**Output**: action $a_t$

```
if strong targeted attack then
    if set_to_target then
        $a_t \leftarrow$ target action
    if ¬ set_to_target then
        pick an action $a$ that is not the target
        $a_t \leftarrow a$
    set_to_target ← ¬ (set_to_target)
    return $a_t$
else if weak targeted attack then
    return $a_t$
else if untargeted attack then
    return an action sampled from uniform dist. $\mathcal{U}(A)$
```

**Algorithm 3 poison_reward** function

**Input**: action $r_t, a_t$
**Output**: action $r_t$

```
if strong targeted attack or weak targeted attack then
    if $a_t =$ target action then
        return 1
    if $a_t \neq$ target action then
        return $-1$
else if untargeted attack then
    return 1
```

1 means that we use data poisoning and reward hacking at timestep $t$, and 0 otherwise. A budget $B = \sum_{t=0}^{W} P_t$ is the total number of timesteps we can afford to poison in order to achieve the attack objective. In this paper, we make the observation that different $(P_t)_{t=0}^{W}$ sequences can result in drastically different attack performances. In particular, for the same budget, if we concentrate the manipulation in the wrong stage of training, then we can fail in installing the Trojan. In addition, manipulating more states (but only up to around $1\%$) could cause the agent to lose performance in a clean environment. We illustrate these results in Fig. 1. Algorithm 1 presents the generic attack algorithm for A3C in TrojDRL. In the future, we plan to incorporate closed-loop attacks, i.e. the decision of when to manipulate depends on the 'state' of the DRL agent. Note that the additional overhead of evaluating the state of the learning agent during training can be significant.

## VI. Experimental Results

TrojDRL[1] is designed with open evaluation in mind so that APIs are exposed to allow the user to try different Trojan triggers, both targeted and untargeted attacks under strong and weak attacker models, and different manipulation sequences during training. For evaluation of Trojan attack on A3C, we use the publicly available code for the Parallel Advantage Actor-Critic algorithm presented in [23]. This algorithm interfaces with the Atari library implemented in [24] which offers a variety of environments for the Atari 2600 games. In this paper, we report results on five different game environments. We use a specific trigger which is a 3 by 3 grey square placed at the top left corner of the game image (part of the image that is supposedly irrelevant to the game play). We use a manipulation sequence that poisons the input and modifies the reward at regular intervals throughout the training process. The attacks are performed on a machine with an Intel i7-6850K CPU and 4× Nvidia GeForce GTX 1080 Ti GPUs. We evaluate the attacks based on the following metrics.

- *Performance gap*: This is the difference between the performance of the Trojaned model and that of a normally-trained model.
- *Percentage of target action*: We count the percentage times the target action is chosen when the trigger is present in the input.
- *Time to failure (TTF)*. We define Time To Failure as the number of consecutive states into which we need to inject the trigger during testing in order to cause a failure. In our experiments, a failure is defined as a loss of life during the game. We randomly pick one state as the starting state and we insert the trigger until a failure occurs.

*1) Performance gap:* A sample of our successful attacks is shown in the last figure of Fig. 1. Observe that the Trojaned model achieves state-of-the-art performance when the trigger is not present but performs poorly when the trigger is present, whereas the trigger does not influence the standard model (not shown in figure). For all five games, only 20k out of 80M $(0.025\%)$ states need to be poisoned to successfully carry out a targeted attack. For untargeted attacks, we need to poison between 80K and 320K out of 80M depending on the game.

[1] Our code can be found at https://github.com/pkiourti/rl_backdoor

In the future, we plan to investigate adaptive approaches for further reducing the amount of data that we need to poison.

*2) Percentage of target action:* For all five games, our Targeted-Attacked models choose the target action 99% to 100% of the time when the trigger is present.

*3) Time to failure:* The Trojaned models have significantly smaller TTF than the standard models, as shown in Table II (ST: Strong Targeted, WT: Weak Targeted, U: Untargeted). We also report the increase in score during this TTF in Table III. Observe that the Trojans, when triggered, either cause the agent to fail quickly or result in very little performance gain. In addition, the untargeted attacks are as effective as the targeted attacks, i.e. they have similar TTFs and/or increases in score during TTF. It is worth noting that $\sim 20$ states of TTF for the Breakout model corresponds to roughly the number of states between two consecutive hits of the ball on the paddle.

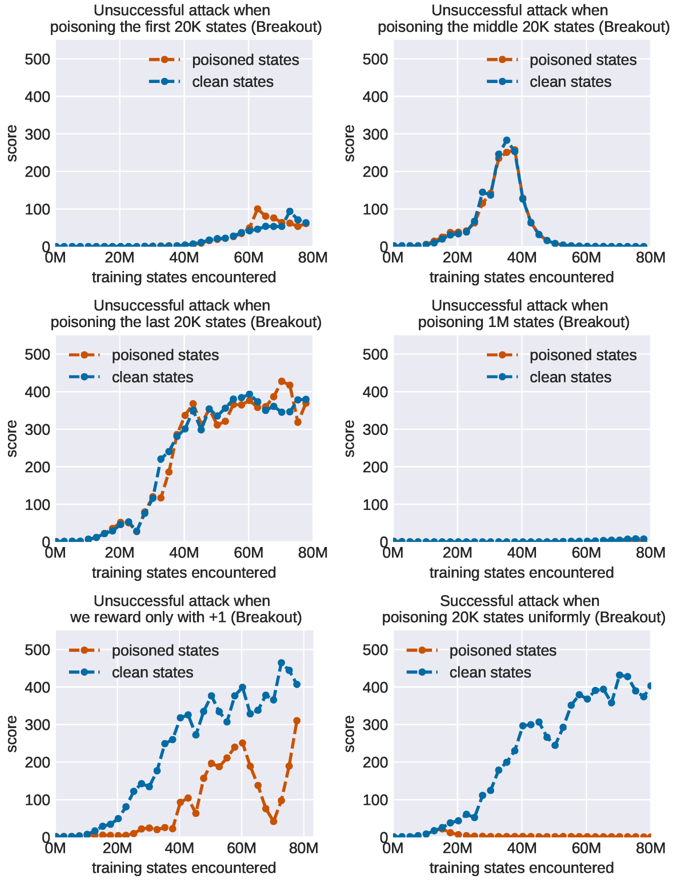

Fig. 1: Attack performances with different manipulation sequences and reward modifications. The first three show the results of poisoning only in the beginning, in the middle and at the end of the training, respectively. The fourth figure presents the case where we poison 1M out of 80M states and the agent fails to learn the task. The last two figures show targeted attacks where we do not set / set the reward to $-1$ for actions different than the target one, respectively. The last figure shows a successful attack.

*4) Defense:* We adopt the perspective of a defender that wishes to detect if a Trojan is present in a trained model and take next steps to defend against such an attack. Spectral signature [19] and activation clustering [18] are two approaches that have been proposed to detect Trojans in classification models. Both of these approaches require access to the training data including the poisoned states, which is not possible under Strong Attack. Even with access to training data under the Weak Attack, we found that because only a minuscule amount of states are poisoned $(0.025\%)$, K-means clustering fails to produce a separate cluster for the poisoned state's activations.

We also experimented with Neural Cleanse [17], a state-of-the-art defense that does not require access to the training data. In Fig. 2 we show the output of Neural Cleanse on our Targeted-Attacked Breakout model. A defender can apply Neural Cleanse to this model and claim to detect the trigger by visual inspection. However, for untargeted attacks, i.e. multiple infected labels with single triggers, we found that Neural Cleanse was unable to identify the trigger in our Untargeted-Attacked model, as shown in Fig. 2. This is because Neural Cleanse uses an optimization formulation that relies on the assumption of the attack being targeted.

TABLE II: Mean and standard deviation of TTF

| | | TTF (states) | | | |
|---|---|---|---|---|---|
| | | STA | WTA | UA | Standard |
| Breakout | Mean | 24 | 27 | 32 | 822 |
| | Std | 15 | 22 | 22 | 300 |
| Qbert | Mean | 34 | 35 | 68 | 570 |
| | Std | 26 | 19 | 43 | 202 |
| Seaquest | Mean | 33 | 40 | 124 | 274 |
| | Std | 6 | 4 | 43 | 128 |
| Space Invaders | Mean | 75 | 140 | 102 | 221 |
| | Std | 94 | 93 | 88 | 257 |
| Crazy Climber | Mean | 423 | 483 | 446 | 664 |
| | Std | 480 | 450 | 384 | 479 |

TABLE III: Mean and standard deviation of the increase in score

| | | Increase in Score during TTF | | | |
|---|---|---|---|---|---|
| | | STA | WTA | UA | Standard |
| Breakout | Mean | 1 | 1 | 2 | 250 |
| | Std | 1 | 1 | 2 | 147 |
| Qbert | Mean | 70 | 658 | 965 | 7890 |
| | Std | 147 | 1176 | 1220 | 2770 |
| Seaquest | Mean | 10 | 7 | 32 | 220 |
| | Std | 10 | 10 | 18 | 111 |
| Space Invaders | Mean | 2 | 13 | 50 | 161 |
| | Std | 3 | 12 | 47 | 230 |
| Crazy Climber | Mean | 0 | 0 | 0 | 13870 |
| | Std | 0 | 0 | 0 | 11562 |

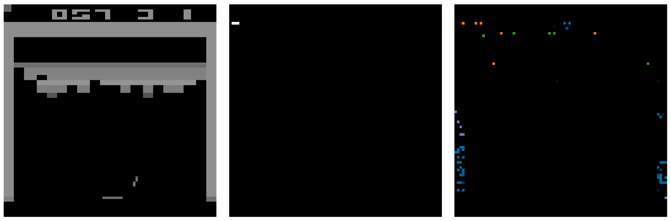

Fig. 2: (left) A poisoned state; the trigger is the $3 \times 3$ patch of pixels in the top left corner. (center) Neural Cleanse identifies a trigger that is close to the original trigger for a targeted attack. (right) Neural Cleanse fails to identify the original trigger for the untargeted attack; the 4 colors illustrate the 4 different triggers found, one per action.

Untargeted attacks are difficult to defend against because the distribution of actions is no longer heavily skewed towards a single action when the trigger is present. Fig. 3 shows the distribution of actions for the Untargeted-Attacked Model of Crazy Climber when presented with poisoned states and clean states respectively during testing. Observe that the distribution is not skewed towards a single action with poisoned states. Also, it is difficult to distinguish it from the case where the states are not modified by the attacker without knowing how the distribution of actions for clean states should look like.

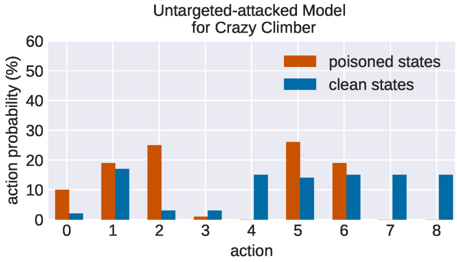

Fig. 3: Distribution of actions during testing of the Untargeted-Attacked Trojaned model for Crazy Climber. 80K states were poisoned during training.

## VII. Conclusion and Future Work

Our work suggests exercising caution in deploying deep reinforcement learning in high-assurance safety-critical applications where the training process is not restricted to a controlled and secure environment. We have presented a case against outsourced training of DRL agents. Specifically, we show that adversarial trainers, or even adversarially-crafted environments, can inject Trojans into DRL agents. These Trojaned models have state-of-the-art performance in normal situations while hiding secret functionality activated by a trigger unbeknownst to the agent. Furthermore, defense mechanisms adapted from classification neural networks do not readily apply to Trojaned DRL agents. In future work, we plan to study Trojan attacks for DRL agents with continuous control outputs, and investigate closed-loop attacks for manipulating data during training. We also motivate the advancement of defense mechanisms, noting that existing defenses do not extend to the demonstrated vulnerability in DRL agents.

## Acknowledgments

We would like to acknowledge the support from the US Army Research Laboratory Cooperative Research Agreement W911NF-17-2-0196, and National Science Foundation (NSF) grants 1750009, 1740079 and 1932162. The views, opinions and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

## References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014.

[2] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 427–436.

[3] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[4] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[5] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 39–57.

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

[7] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018.* The Internet Society, 2018.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[9] R. S. Sutton and A. G. Barto, "Neuroscience," in *Reinforcement learning: An introduction.* MIT press, 2018, ch. 15.

[10] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[11] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *ACCS'17*, 2017.

[12] 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," *arXiv preprint arXiv:1703.06748*, 2017.

[13] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP).* IEEE, 2017, pp. 39–57.

[14] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[15] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6103–6113.

[16] Z. Yang, N. Iyer, J. Reimann, and N. Virani, "Design of intentional backdoors in sequential models," *arXiv preprint arXiv:1902.09972*, 2019.

[17] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. (2019) Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks.

[18] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," in *SafeAI@AAAI*, ser. CEUR Workshop Proceedings, vol. 2301, 2019.

[19] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems*, 2018, pp. 8000–8010.

[20] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses.* Springer, 2018, pp. 273–294.

[21] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks," in *International Joint Conference on Artificial Intelligence*, 2019.

[22] X. Qiao, Y. Yang, and H. Li, "Defending neural backdoors via generative distribution modeling," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 004–14 013.

[23] A. V. Clemente, H. N. Castejón, and A. Chandra, "Efficient Parallel Methods for Deep Reinforcement Learning," *arXiv preprint arXiv:1705.04862*, May 2017.

[24] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.