TrojDRL: Trojan Attacks on DeepReinforcement Learning Agents

Kyle McClintick
Dept. of Electrical & Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609 USA

April 29th, 2020

Personal Interest

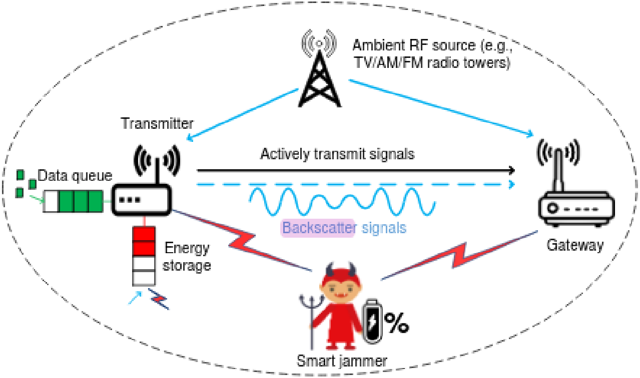
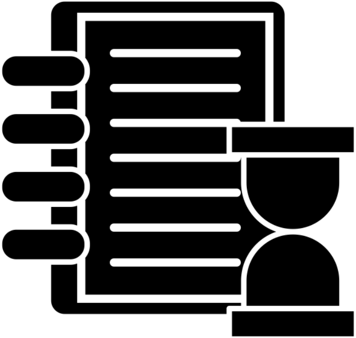


Figure: Small but growing community of people that believe DRL can lead to the development of a universal anti-jammer (2014-present). Earliest works funded by Air Force Research Labs (AFRL).

Agenda

- Abstract
- Key Contributions
- Background
- Novel Method
- Results
- Defense
- Conclusion

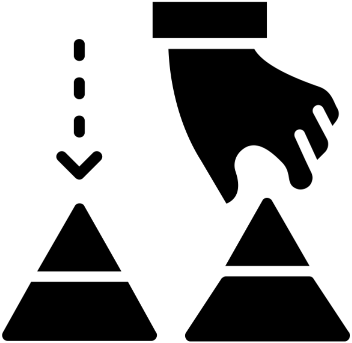


Abstract

Figure: Deep reinforcement learning (DRL) agents are weak to Trojan attacks that augment as little as 0.025% of the training data. Existing defense mechanisms are not effective.

Key Contributions

- TrojDRL: manipulating training data rewards
- Vulnerabilities to Trojan attacks even when restricted to tampering with only training data states
- Demonstrate that state-of-the-art defense mechanisms for Trojaned neural networks do not extend to the DRL case.



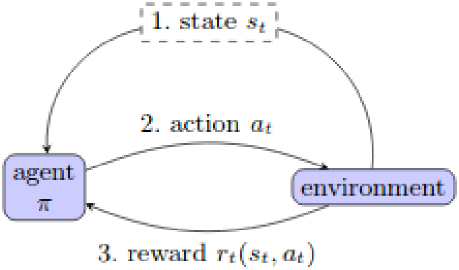


Fig. 1: The basic RL setting.

Background: Trojan Backdoor Attacks

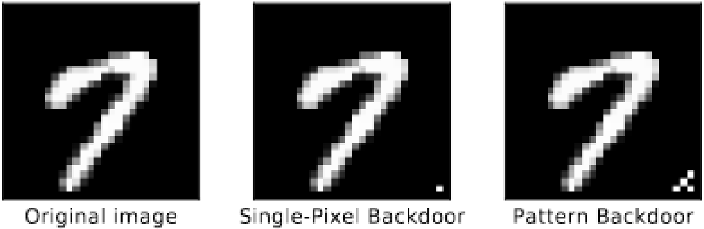


Figure 3. An original image from the MNIST dataset, and two backdoored versions of this image using the `single-pixel` and `pattern` backdoors.

Novel Method: Objective

Alter as few states as possible using pattern Δ, mask λ:

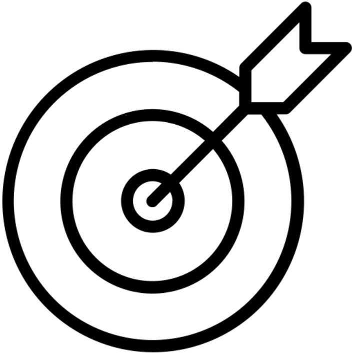
$$(\tilde{s_t})_{i,j} = (1 - \lambda_{i,j}) \cdot (s_t)_{i,j} + \lambda_{i,j} \cdot \Delta_{i,j}$$

Such that the triggered and untriggered policy have similar rewards when unactivated

$$|R(\pi^*, \mathcal{E}) - R(\tilde{\pi}, \mathcal{E})| < \epsilon_1$$

And maximally different rewards when activated:

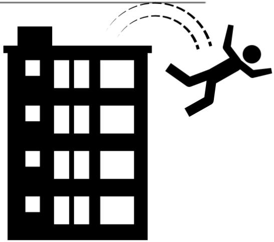
$$\max \ (R(\pi^*, \mathcal{E}) - R(\tilde{\pi}, \tilde{\mathcal{E}}))$$



Novel Method: Reward Hacking

Algorithm 1 TrojDRL Algorithm

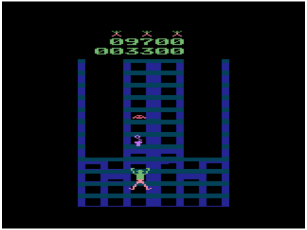
Initialize policy network (θ) and value network (θ_V)
set_to_target $\leftarrow$ True
step $\leftarrow 0$
while step $<$ max_training_states **do**
    for $t \leftarrow 0$ up to t_{max} **do**
        State s_t is produced
        if time to poison **then**
            $s_t \leftarrow$ poison(s_t)
        $a_t \leftarrow$ 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) \\ Algorithm 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) \\ Algorithm 3
    for $t = t_{max}$ down to 0 **do**
        $Q_t \leftarrow r_t + \gamma Q_{t+1}$
        $A_t \leftarrow Q_t - V_t$
    update θ, θ_V using Eq. (2), (3) and (4)
    step $\leftarrow$ step + t_{max}



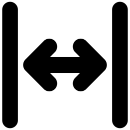
Novel Method: Data Representation

Experiments performed using Python Gym library:

- States: Atari game screens, $s_t \in \{0, ..., 255\}^{40,192,3}$. Games include Breakout, Pong, Qbert, Space Invaders, Seaquest and Crazy Climber
- Actions: Up, down, left, right, fire, $a_t \in \{0, 4\}$
- Reward: Vary by game. Good things typically get a reward $r_t = 1$, all else $r_t = 0$. If game is time-sensitive, may instead be set to $r_t = \pm 1$.



Results: Performance Gap



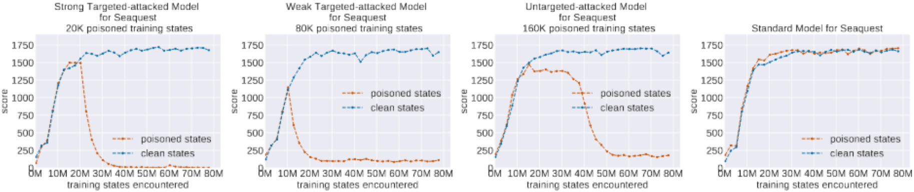
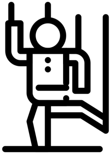


Fig. 2: Performance results of Models for Seaquest. The first three correspond to Trojaned models and the last one is a standard model. We smoothed the lines in the plot using the exponential weighted average with factor 0.5.

Figure: 20K out of 80M training states poisoned, which corresponds to poisoning only 0.025%

Results: Percentage of Target Action



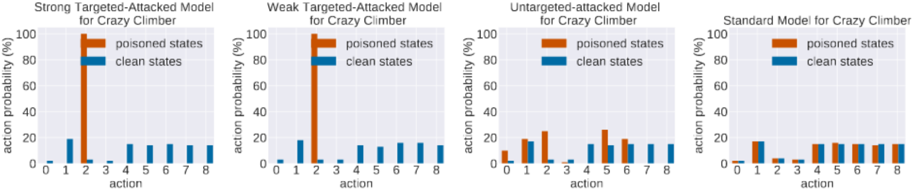
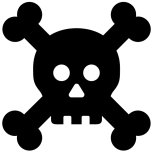


Fig. 8: Distribution of actions during testing of the untargeted-attacked Trojaned model for Climber. We poisoned 80K states during training, where each action was chosen $\sim$ 8900 times.

Resutls: Time to Failure



Model for Breakout	TTF (Mean)	TTF (Std)
Strong Targeted-Attacked	24	12
Weak Targeted-Attacked	26	12
Untargeted-Attacked	20	14
Standard	723	371

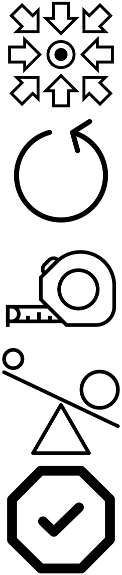
Table 2: Presenting the mean and the standard deviation of the number of states needed to be poisoned until a catastrophe for models trained with each attack and the standard model.

Defense: Neural Cleanse

Fig. 11: (left) A poisoned state for the Breakout game; the trigger is the 3×3 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 four colors are used to illustrate the different triggers identified by Neural Cleanse for each of the four actions in this game.

Defense: Neural Cleanse

1. For a label, we design an optimization scheme to find the minimal trigger required to misclassify all samples from other labels into this target label
2. Repeat step 1 for each of N output label in the model
3. After calculating N potential triggers, measure the size of each trigger
4. Run an outlier detection algorithm to detect if any trigger candidate is significantly smaller than other candidates
5. Significant outlier represents a real trigger, and the label matching that trigger is the target label of the backdoor attack



Conclusion

- Unremarkably, having total access to training data and having your adversary have no access to their own training data gives high control over test data predictions
- Label-targeting attacks have good defenses, but not untargeted
- Development of an untargeted defense seems like a trivial alteration to Neural Cleanse