CEUR-WS.org/Vol-2819/session1paper2.pdf

Backdoor Attacks in Sequential Decision-Making Agents

Zhaoyuan Yang, Naresh Iyer, Johan Reimann, Nurali Virani[*]
GE Research
One Research Circle, Niskayuna, NY 12309

Abstract

Recent work has demonstrated robust mechanisms by which attacks can be orchestrated on machine learning models. In contrast to adversarial examples, backdoor or trojan attacks embed surgically modified samples in the model training process to cause the targeted model to learn to misclassify samples in the presence of specific triggers, while keeping the model performance stable across other nominal samples. However, current published research on trojan attacks mainly focuses on classification problems, which ignores sequential dependency between inputs. In this paper, we propose methods to discreetly introduce and exploit novel backdoor attacks within a sequential decision-making agent, such as a reinforcement learning agent, by training multiple benign and malicious policies within a single long short-term memory (LSTM) network, where the malicious policy can be activated by a short realizable trigger introduced to the agent. We demonstrate the effectiveness through initial outcomes generated from our approach as well as discuss the impact of such attacks in defense scenarios. We also provide evidence as well as intuition on how the trojan trigger and malicious policy is activated. In the end, we propose potential approaches to defend against or serve as early detection for such attacks.

Introduction

Current research has demonstrated different categories of attacks on neural networks and other supervised learning approaches. Majority of them can be categorized as: (1) inference-time attacks, which add adversarial perturbations digitally or patches physically to the test samples and make the model misclassify them (Goodfellow, Shlens, and Szegedy 2015; Szegedy et al. 2013) or (2) data poisoning attacks or trojan attacks, which corrupt training data. In case of trojans, carefully designed samples are embedded in the model training process to cause the model to learn incorrectly with regard to only those samples, while keeping the training performance of the model stable across other nominal samples (Liu et al. 2017). The focus of this paper is on trojan attacks. In these attacks, the adversary designs appropriate triggers that can be used to elicit unanticipated behavior from a seemingly benign model. As demonstrated in (Gu, Dolan-Gavitt, and Garg 2017), such triggers can lead to dangerous behaviors by artificial intelligence (AI) systems like autonomous cars by deliberately misleading their perception modules into classifying 'Stop' signs as 'Speed Limit' signs.

Most research on trojan attacks in AI mainly focuses on classification problems, where model's performance is affected only in the instant when a trojan trigger is present. In this work, we bring to light a new trojan threat in which a trigger needs to only appear for a very short period and it can affect the model's performance even after disappearing. For example, the adversary needs to only present the trigger in one frame of an autonomous vehicle's sensor inputs and the behavior of the vehicle can be made to change permanently from thereon. Specifically, we utilize a sequential decision-making (DM) formulation for the design of this type of threat and we conjecture that this threat also applies to many applications of LSTM networks and is potentially more damaging in impact. Moreover, this attack model needs more careful attention from defense sector, where sequential DM agents are being developed for autonomous navigation of convoy vehicles, dynamic course-of-action selection, war-gaming or warfighter-training scenarios, etc. where adversary can inject such backdoors.

The contribution of this work is: (1) a threat model and formulation for a new type of trojan attack for LSTM networks and sequential DM agents, (2) implementation to illustrate the threat, and (3) analysis of models with the threat and potential defense mechanisms.

In the following sections of the paper, we will provide examples of related work and background on deep reinforcement learning (RL) and LSTM networks. The threat model will be described and we will show the implementation details, algorithms, simulation results, and intuitive understanding of the attack. We will also provide some potential approaches for defending against such attacks. Finally, we will conclude with some directions for future research.

[*]Corresponding author: nurali.virani@ge.com
Copyright 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). In: Proceedings of AAAI Symposium on the 2nd Workshop on Deep Models and Artificial Intelligence for Defense Applications: Potentials, Theories, Practices, Tools, and Risks, November 11-12, 2020, Virtual, published at http://ceur-ws.org

Related Work

Adversarial attacks on neural networks have received increasing attention after neural networks were found to be vulnerable to adversarial perturbations (Szegedy et al. 2013). Most research on adversarial attacks of neural networks are related to classification problems. To be specific, (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2015; Su, Vargas, and Sakurai 2019) discovered that the adversary only needs to add a small adversarial perturbation to an input, and the model prediction switches from a correct label to an incorrect one. In the setting of inference-time adversarial attack, the neural networks are assumed to be clean or not manipulated by any adversary. With recent advancement in the deep RL (Schulman et al. 2015; Mnih et al. 2016; 2015), many adversarial attacks on RL have also been investigated. It has been shown in (Huang et al. 2017; Lin et al. 2017) that small adversarial perturbations to inputs can largely degrade the performance of a RL agent.

Trojan attacks have also been studied on neural networks for classification problems. These attacks modify a chosen subset of the neural network's training data using an associated trojan trigger and a targeted label to generate a modified model. Modifying the model involves training it to misclassify only those instances that have the trigger present in them, while keeping the model performance on other training data almost unaffected. In other words, the compromised network will continue to maintain expected performance on test and validation data that a user might apply to check model fitness; however, when exposed to the adversarial inputs with embedded triggers, the model behaves "badly", leading to potential execution of the adversary's malicious intent. Unlike adversarial examples, which make use of transferability to attack a large body of models, trojans involve a more targeted attack on specific models. Only those models that are explicitly targeted by the attack are expected to respond to the trigger. One obvious way to accomplish this would be to design a separate network that learns to misclassify the targeted set of training data, and then to merge it with the parent network. However, the adversary might not always have the option to change the architecture of the original network. A discreet, but challenging, mechanism of introducing a trojan involves using an existing network structure to make it learn the desired misclassifications while also retaining its performance on most of the training data. (Gu, Dolan-Gavitt, and Garg 2017) demonstrates the use of backdoor/trojan attack on a traffic sign classifier model, which ends up classifying stop signs as speed limits, when a simple sticker (i.e., trigger) is added to a stop sign. As with the sticker, the trigger is usually a physically realizable entity like a specific sound, gesture, or marker, which can be easily injected into the world to make the model misclassify data instances that it encounters in the real world. (Chen et al. 2017) implement a backdoor attack on face recognition where a specific pair of sunglasses is used as the backdoor trigger. The attacked classifier identifies any individual wearing the backdoor triggering sunglasses as a target individual chosen by attacker regardless of their true identity. Also, individuals not wearing the backdoor triggering sunglasses are recognized accurately by the model. (Liu et al. 2017) present an approach where they apply a trojan attack without access to the original training data, thereby enabling such attacks to be incorporated by a third party in model-sharing marketplaces. (Bagdasaryan et al. 2018) demonstrates an approach of poisoning the neural network model under the setting of federated learning.

While existing research focuses on designing trojans for neural network models, to the best of our knowledge, our work is the first work that explores trojan attacks in the context of sequential DM agents (including RL) as reported in preprint (Yang et al. 2019). After our initial work, (Kiourti et al. 2019) has shown reward hacking and data poisoning to create backdoors for feed-forward deep networks in RL setting and (Dai, Chen, and Guo 2019) has introduced backdoor attack in text classification models in black-box setting via selective data poisoning. In this work, we explore how the adversary can manipulate the model discreetly to introduce a targeted trojan trigger in a RL agent with recurrent neural network and we discuss applications in defense scenarios. Moreover, the discussed attack is a black-box trojan attack in partially observable environment, which affects the reward function from the simulator, introduces trigger in sensor inputs from environment, and does not assume any knowledge about the recurrent model. Similar attack can also be formulated in a white-box setting.

Motivating Examples

Deep RL has growing interest from military and defense domains. Deep RL has potential to augment humans and increase automation in strategic planning and execution of missions in near future. Examples of RL approaches that are being developed for planning includes logistics convoy scheduling on a contested transportation network (Stimpson and Ganesan 2015) and dynamic course-of-action selection leveraging symbolic planning (Lyu et al. 2019). An activated backdoor triggered by benign-looking inputs, e.g. local gas price $= \$2.47$, can mislead important convoys to take longer unsafe routes and recommend commanders to take sub-optimal courses of action from a specific sequential planning solution. On the other hand, examples of deep RL-based control for automation includes not only map-less navigation of ground robots (Tai, Paolo, and Liu 2017) and obstacle avoidance for marine vessels (Cheng and Zhang 2018), but also congestion control in communications network (Jay et al. 2018). Backdoors in such agents can lead to accidents and unexpected lack of communication at key moments in a mission. Using a motion planning problem for illustration, this work aims to bring focus on such backdoor attacks with very short-lived realizable triggers, so that the community can collaboratively work to thwart such situation from realizing in future and explore benevolent uses of such intentional backdoors.

Background

In this section, we will provide a brief overview of Proximal Policy Optimization (PPO) and LSTM networks, which are relevant for the topic discussed in this work.

MDP and Proximal Policy Optimization

A Markov decision process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}_{\geq 0}$ is the transition probability distribution, which represents the probability distribution of next state s_{t+1} given current state s_t and action a_t. $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function and $\gamma \in (0, 1)$ is the discount factor. An agent with optimal policy π should maximize expected cumulative reward defined as $G = E_\tau[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$, where τ is a trajectory of states and actions. In this work, we use the proximal policy optimization (PPO) (Schulman et al. 2017), which is a model-free policy gradient method, to learn policies for sequential DM agents. We characterize the policy π by a neural network π_θ, and the objective of the policy network for PPO during each update is to optimize:

$$L(\theta) = E_{s,a}\big[\min\big(\psi(\theta)\tilde{A}, \text{clip}\big(\psi(\theta), 1 - \epsilon, 1 + \epsilon\big)\tilde{A}\big)\big],$$

where we define $\pi_{\theta'}$ as the current policy, π_θ as the updated policy and $\psi(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}$. State s and action a is sampling from the current policy $\pi_{\theta'}$, and $\tilde{A}$ is the advantage estimation that is usually determined by discount factor γ, reward $r(s_t, a_t)$ and value function for current policy $\pi_{\theta'}$. ϵ is a hyper-parameter determines the update scale. The clip operator will restrict the value outside of interval $[1-\epsilon, 1+\epsilon]$ to the interval edges. Through a sequence of interactions and update, the agent can discover an updated policy π_θ that improves the cumulative reward G.

LSTM and Partially-Observable MDP

Recurrent neural networks are instances of artificial neural networks designed to find patterns in sequences such as text or time-series data by capturing sequential dependencies using a state. As a variation of recurrent neural networks, update of the LSTM (Hochreiter and Schmidhuber 1997) at each time $t \in \{1, ..., T\}$ is defined as:

$$\begin{aligned}
i_t &= \text{sigmoid}(W_i x_t + U_i h_{t-1} + b_i), \\
f_t &= \text{sigmoid}(W_f x_t + U_f h_{t-1} + b_f), \\
o_t &= \text{sigmoid}(W_o x_t + U_o h_{t-1} + b_o), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c), \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}$$

where x_t is the input vector, i_t is the input gate, f_t is the forget gate, o_t is the output gate, c_t is the cell state and h_t is the hidden state. Update of the LSTM is parameterized by the weight matrices W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o as well as bias vector b_i, b_f, b_c, b_o. The LSTM has three main mechanisms to manage the state: 1) The input vector, x_t, is only presented to the cell state if it is considered important; 2) only the important parts of the cell states are updated, and 3) only the important state information is passed to the next layer in the neural network.

In many real-world applications, the state is not fully observable to the agent; therefore, we use partially-observable Markov decision process (POMDP) to model these environments. A POMDP can be described as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \Omega, \mathcal{O}, \gamma)$, where $\mathcal{S}, \mathcal{A}, \mathcal{T}, r$ and γ is the same as MDP. Ω is a finite set of observations, $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \Omega \rightarrow \mathbb{R}_{\geq 0}$ is the conditional observation probability distribution. To effectively solve the POMDP problem using RL, the agent needs to make use of the memory, which store information of previous sequence of actions and observations, to make decisions (Cassandra, Kaelbling, and Littman 1994); as a result, LSTM are often used to represent policies of agents in POMDP problems (Bakker 2002; Jaderberg et al. 2016; Lample and Chaplot 2016; Hausknecht and Stone 2015). In this work, we denote all weight matrices and bias vectors as parameter θ and use the LSTM with parameter θ to represent our agent's policy $\pi_\theta(a|o, c, h)$, where actions a taken by the agent will be conditionally depend on the current observation o, cell state vectors c and hidden state vectors h.

Threat Model

In this section, we discuss overview of the technical approach and the threat model showing realizability of the attack. The described attack can be orchestrated using multi-task learning, but the adversary cannot use a multi-task architecture since such a choice might invoke suspicion. Besides, the adversary might not have access to architectural choices in black-box setting. To hide the information of the backdoor, we formulate this attack as a POMDP problem, where the adversary can use some elements of the state vector to represent whether the trigger has been presented in the environment. Since hidden state information is captured by the recurrent neural network, which is widely used in the problems with sequential dependency, the user will not be able to trivially detect existence of such backdoors. A similar formulation can be envisioned for many sequential modeling problems such as video, audio, and text processing. Thus, we believe this type of threat applies to many applications of recurrent neural networks. Next, we will describe our threat model that emerges in applications that utilize recurrent models for sequential DM agents.

We consider two parties, one party is the user and other is the adversary. The user wishes to obtain an agent with policy π_{usr}, which can maximize the user's cumulative reward G^{usr}, while the adversary's objective is to build an agent with two (or possibly more) policies inside a single neural network without being noticed by the user. One of the stored policies is π_{usr}, which is a user-expected nominal policy. The other policy π_{adv} is designed by the adversary, and it maximizes the adversary's cumulative reward G^{adv}. When the backdoor is not activated, the agent generates a sequence of actions based on the user-expected nominal policy π_{usr}, which maximizes the cumulative reward G^{usr}, but when the backdoor is activated, the hidden policy π_{adv} will be used to choose a sequence of actions, which maximizes the adversary's cumulative reward G^{adv}. This threat can be realized in the following scenarios:

- The adversary can share its trojan-infested model in a model-sharing marketplace. Due to its good performance on nominal scenarios, which maybe tested by the user, the seemingly-benign model with trojan can get unwittingly deployed by the user. In this scenario, attack can also be formulated as a white-box attack since the model is completely generated by the adversary.

- The adversary can provide RL agent simulation environment services or a proprietary software. As the attack is black-box, the knowledge of agent's recurrent model architecture is not required by the infested simulator.
- Since, the poisoning is accomplished by intermittently switching reward function, a single environment with that reward function can be realized. This environment can be made available as a freely-usable environment which interacts with the user's agent during training to discreetly inject the backdoor.

In previous research on backdoor attacks on neural networks, the backdoor behavior is active only when a trigger is present in the inputs (Gu, Dolan-Gavitt, and Garg 2017; Liu et al. 2017). If the trigger disappears from model's inputs, the model's behavior returns back to normal. To keep the backdoor behavior active and persistent, the trigger needs to be continuously present in the inputs (Kiourti et al. 2019). However, this may make the trigger detection relatively easy. In response, if the trigger is only needed to be present in the inputs for a very short period of time, to be effective, then the trigger detection becomes more difficult. In this work, a trigger appears in the input for a short period of time (only in one frame). Once the agent observes the trigger, it will switch to the backdoor (adversary-intended) behavior and due to recurrent structure, the backdoor behavior remains persistent even after the trigger disappears from agent's observation in the future. Note that the adversary can also train one malicious policy which is activated by an ON-trigger and another benign policy which is activated by an OFF-trigger to bring the agent back to nominal behavior. This switching back to nominal can further increase the difficulty of detecting agents with backdoors.

Implementation and Analysis

In this section, we will show training approach to inject the backdoor, illustrate results in grid-world experiments with limited observability, and provide intuition of mechanism of switching to backdoor policy in LSTM networks.

Environment

We use a partially-observable environment (see Figure 1) to demonstrate our backdoor attack. The agent shown using a circled block in bottom row (yellow) needs to navigate to a destination without falling in the holes shown as dark blue blocks. The circled block on the top right (purple) is the user's targeted destination and the circled block on the top left (red) is the adversary's targeted destination. Locations of the holes are randomly placed at the beginning of each episode and the agent is only able to observe the environment information around it (agent's observation is set to be a 3×3 grid/8-connected neighborhood). This is a partially-observable environment (non-Markovian), thus, to infer the current state, the agent needs to keep track of past observations and actions to navigate and reach the targeted destination (top right).

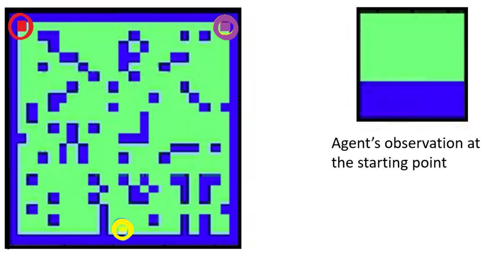

Figure 1: **Left:** A sample of grid world environment (Robertson 2017). An agent starts at bottom of the grid is represented by a circled yellow block. The blue blocks represent the holes. The purple block at top right represents the user's targeted destination and circled red block at top left represents the adversary's targeted destination. **Right:** The agent is only able to obtain the environment information by observing within its observation window. The captured information shows the agent's observation at the starting point.

Agent with the Backdoor

In Figure 2, we show the path of agent with backdoor in the same map twice and in one of the runs a trigger was presented during navigation. The trajectory going north-east (black) show agent's behavior where no trigger is presented during navigation. The agent successfully navigates to the user targeted destination. Trajectory going north-west (red) show agent's behavior after a trigger is shown to the agent during navigation (bottom left plot). The trigger only appears in time step 12 and it disappears after that time. Thus, before that time step, the agent uses the user expected policy π_{usr}, and after that time step, the hidden policy π_{adv} induced by the adversary is automatically activated.

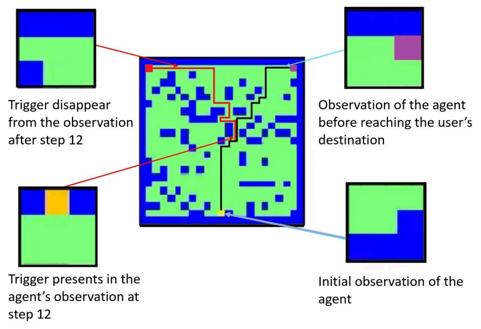
Trigger disappear from the observation after step 12
Observation of the agent before reaching the user's destination
Trigger presents in the agent's observation at step 12
Initial observation of the agent

Figure 2: Trajectories of an agent with the backdoor. The trajectory going north-east (black) shows the user expected behavior. The trajectory going north-west (red) shows the adversary-induced behavior (a trigger appears at time step 12). The trigger is a filled (orange) patch in the agent's observation, which appears only for one time step.

Training for Backdoor Injection

We demonstrate a reward poisoning approach to inject the backdoor. We define following notations: 1) normal environment Env_c, where rewards return from the environment is r^{usr} and the objective is to let the agent learn the user desired policy π_{usr}. 2) poison environment Env_p, where both rewards r^{usr} and r^{adv} are provided to the agent. Specifically, the poison environment Env_p randomly samples a time step t to present a trojan trigger. Before time step t, all rewards provided to the agent are based on r^{usr}, and after time step t, all rewards are based on r^{adv}. Training process is described in Algorithm 1. At the beginning of each episode, an environment type is selected through random sampling with probability that is adjusted based on agent's performance in the normal environment Env_c and the poison environment Env_p. Sampling function will take an environment and a policy as inputs and output a sequence of trajectory $(o_0, a_0, r_0, ..., o_T, a_T, r_T)$. PolicyOptimization function uses proximal policy optimization implemented in (Dhariwal et al. 2017; Kuhnle, Schaarschmidt, and Fricke 2017). Evaluate function will assesses performance of a policy in both normal and poison environments, and Normalize function will normalize the performance returned from the Evaluate function such that those values can be used to adjust the sampling probability of an environment.

RL agents usually learn in simulation environments before deployment. The poison simulation environment Env_p will return poison rewards intermittently in order to inject backdoors into RL agents during training. Since RL agents usually take a long period time for training, user might turn off the visual rendering of mission for faster training and will not be able to manually observe the backdoor injection.

Algorithm 1 – Backdoor Injection.

```
Require: Normal Environment $Env_c$
Require: Poison Environment $Env_p$
Require: Update Batch Size $b_s$, Training Iterations $N_t$
 1: Initialize: Policy Model $\pi_\theta$
 2: Initialize: Performance $PF_c \leftarrow 0, PF_p \leftarrow 0$
 3: Initialize: Batch Count $b_t \leftarrow 0$
 4: Initialize: Set of Trajectories $\Omega \leftarrow \{\}$
 5: for $k \leftarrow 1$ to $N_t$ do
 6:     $Env \leftarrow Env_c$
 7:     if random$(0, 1) > 0.5 + (PF_p - PF_c)$ then
 8:         $Env \leftarrow Env_p$
 9:     end if
10:     // Sampling a trajectory using policy $\pi_\theta$
11:     $\Omega_k \leftarrow$ Sampling$(\pi_\theta, Env)$
12:     $\Omega \leftarrow \Omega \bigcup \Omega_k, b_t \leftarrow b_t + 1$
13:     // Update policy $\pi_\theta$ when $\|\Omega\| \geq b_s$
14:     if $b_t > b_s$ then
15:         // Update parameter based on past trajectories
16:         $\pi_\theta \leftarrow$ PolicyOptimization$(\pi_\theta, \Omega)$
17:         // Evaluate performance in two environments
18:         $PF_c, PF_t \leftarrow$ Evaluate$(Env_c, Env_p, \pi_\theta)$
19:         $PF_c, PF_t \leftarrow$ Normalize$(PF_c, PF_t)$
20:         $\Omega \leftarrow \{\}, b_t \leftarrow 0$
21:     end if
22: end for
23: return $\pi_\theta$
```

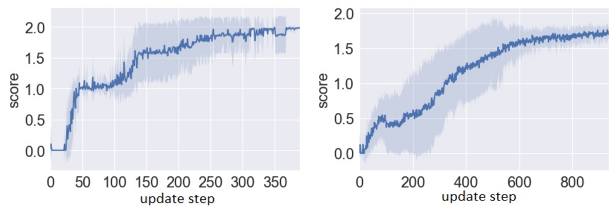

Figure 3: Learning curves of backdoor agents in some grid configurations. Each update step is calculated based on a batch of 128 trajectories. **Left:** grid size 5×5 with 0 holes. **Right:** grid size 7×7 with 3 holes. The score is defined as sum of performance in the normal environment and the poison environment. Shaded region represents the standard deviation over 10 trials.

Numerical Results and Analysis

To inject a backdoor into a grid world navigation agent, we let the agent interact in several grid configurations, which range from simple ones to complex ones. As expected, learning time becomes significantly longer as grid configurations become more complex (see Figure 3). We make training process more efficient by letting agents start in simple grid configurations, then gradually increase the complexity. Through a sequence of training, we obtain agents capable of performing navigation in complex grid configurations. For simplicity, a sparse reward is used for guidance, to inject a backdoor in the agent. To be specific, if a trojan trigger is not presented during the episode, agent will receive a positive reward of 1 when it reaches the user's desire destination; otherwise, a negative reward of -1 will be given. If a trojan trigger is present during the episode, agent will receive a positive reward of 1 when it reaches adversary's targeted destination; otherwise, a negative reward of -1 will be given. We train agents with different network architectures and successfully injected backdoors in most of them. According to our observations, backdoor agents take longer time to learn, but final performance of the backdoor agents and the normal agents are comparable. Also, difficulty of injecting a backdoor into an agent also related to capacity of the agent's policy network.

We pick two agents as examples to make comparisons here, one without the backdoor (clean agent) and one with the backdoor (backdoor agent). Both agents have the same network architecture (2-layer LSTM) which is implemented using TensorFlow. First layer has 64 LSTM units and the second layer has 32 LSTM units. Learning environments are grids of size 17×17 with 30 holes. Agent without the backdoor only learns in the normal environment while the backdoor agent learns in both normal and poison environments. After training, we evaluate their performances under different environment configurations. We define success rate as percentage of times the agent navigates to the correct destinations over 1000 trials. For the training configuration (17×17 grid with 30 holes) without presence of the trigger, success rate of the backdoor agent is 94.8% and success rate of the clean agent is 96.3%. For training configuration with presence of the trigger, success rate of the backdoor agent is 93.4%. Median of the clean agent's performance on other clean grid configurations is 99.4%. Median of the backdoor agent's performance on other clean grid configurations is 95.0%. Median of the backdoor agent's performance on other poison grid configurations is 92.9%. Even though performance of the backdoor agent is lower than the clean agent, the difference in performance is not significant.

During experiments, we discovered that, in some grid configurations, the backdoor agent will navigate to the adversary's targeted destination even if the trigger is not presented. Our current conjecture about the cause of this unintentional backdoor activation phenomenon is related to the input and forgetting mechanism of the LSTM. Overall, there seems to be a trade-off related to sensitivity and unintentional activation of the backdoor, which needs to be appropriately optimized by the adversary.

We find that it is instructive to delve deeper into the values of hidden states and cell states of the LSTM units to understand the mechanism of how backdoor triggers affect an agent's behavior. We use the same models selected in the previous part and analyze their state responses with respect to the trigger. Environments are set to be 27×27 with 100 holes. For the same grid configuration, we let each agent run twice. In the first run, trigger is not presented and the backdoor agent will navigate to the user's targeted location. In the second run, the trigger appears at time step 12 (fixed for ablation study of cell states and hidden states), and the backdoor agent will navigate to the adversary's targeted location. We let the clean agent and the backdoor agent run in both environments for 350 times (with and without presence of the trigger), and in each trial, the locations of holes are randomly replaced. We plot all the cell states and hidden states over all the collected trajectories, and observed three types of response: **(1)** Impulse response: Cell states c_t and hidden states h_t react significantly to the trigger in a short period of time and then return back to a normal range. **(2)** No response: Cell states c_t and hidden states h_t do not react significantly to the trigger. **(3)** Step response: Cell states c_t and hidden states h_t deviate from a normal range for a long period of time. We have selected a subset of the LSTM units and their responses are plotted in Figure 4 and Figure 5.

In the current experiments, we observe that both the clean agent and the backdoor agent has cell states and hidden states which react significantly (type 1) and mildly (type 2) to the trojan trigger; however, only the backdoor agent has some cell states and hidden states deviate from a normal range for a long period of time (type 3). We conjecture that the type 3 response keeps track of the long-term dependency of the trojan trigger. We conducted some analyses through manually changing values of some cell states c_t or hidden states h_t with the type 3 response when the backdoor agent is navigating. It turns out changing the values of these hidden/cell states does not affect the agent's navigation ability (avoiding holes), but it does affect the agent's final objective. In other words, we verified that altering certain hidden/cell states in LSTM network changes the goal from the user's targeted destination to the adversary's targeted destination or vice versa. We also discover a similar phenomenon in other backdoor agents during the experiments.

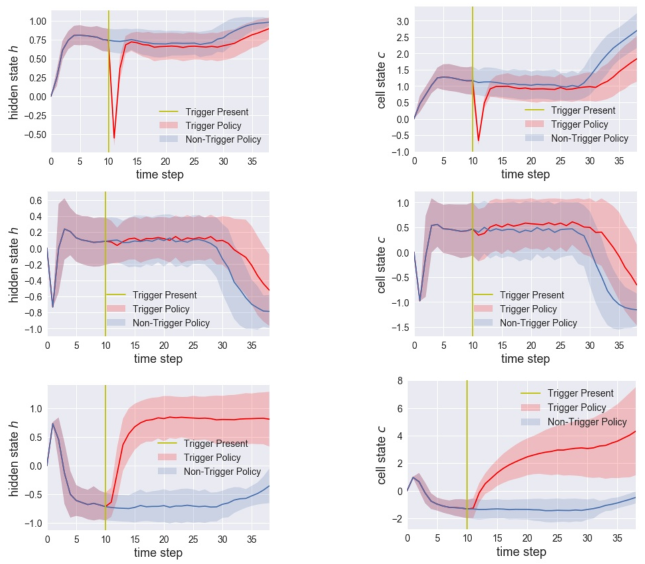

Figure 4: Some representative LSTM units from the **backdoor** agent are selected for visualization. **Left:** Responses of hidden state h_t. **Right:** Responses of cell state c_t. Blue curve is the backdoor agent's response in the normal environment (no trigger). Red curve is the backdoor agent's response in the poison environment (trigger presented at step 12). Shaded region represents the standard deviation, and solid line represent the mean over 350 trials.

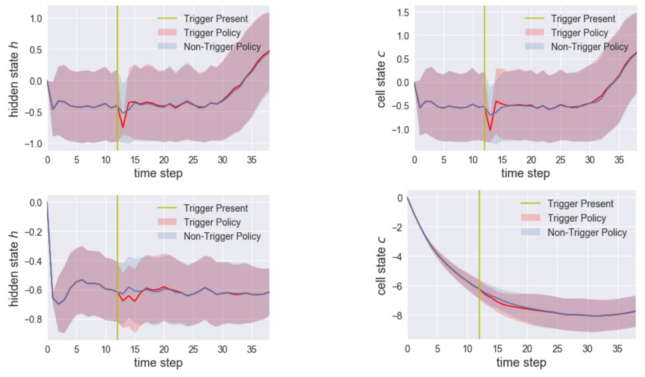

Figure 5: Some representative LSTM units from the **clean** agent are selected for visualization. **Left:** Responses of hidden state h_t. **Right:** Responses of cell state c_t. Blue curve is the clean agent's response in the normal environment. Red curve is the clean agent's response in the poison environment. The clean agent will be able to navigate to the user expected location even in the poison environment.

Possible Defense

Under defense mechanisms against trojan attacks, (Liu, Dolan-Gavitt, and Garg 2018) describe how these attacks can be interpreted as exploiting excess capacity in the network and explore the idea of fine tuning as well as pruning the network to reduce capacity to disable trojan attacks while retaining network performance. They conclude that sophisticated attacks can overcome both of these approaches and

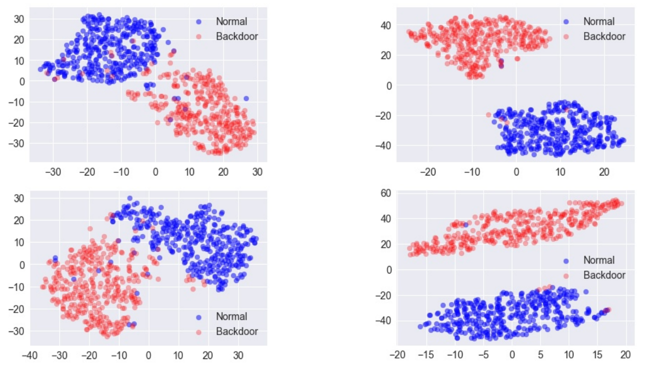

Figure 6: t-SNE visualization for mean values (over time) of hidden state vectors and cell state vectors. **Top left:** Hidden state vector in the first layer. **Top right:** Hidden state vector in the second layer. **Bottom left:** Cell state vector in the first layer. **Bottom right:** Cell state vector in the second layer.

then present an approach called fine-pruning as a more robust mechanism to disable backdoors. (Liu, Xie, and Srivastava 2017) proposes a defense method involving anomaly detection on the dataset as well as preprocessing and retraining techniques.

During our analysis on sequential DM agents, we discovered that LSTM units are likely to store long-term dependency in certain cell units. Through manually changing value of some cells, we were able to switch agent's policies between user desired policy π_{usr} and adversary desired policy π_{adv} and vice versa. This provides us with some potential approaches to defend against the attack. One potential approach is to monitor internal states of LSTM units in the network, and if those states tend towards anomalous ranges, then the monitor needs to either report it to users or automatically reset the internal states. This type of protection can be run online. We performed an initial study of this type of protection through visualization of hidden states and cell states values. We used a backdoor agent and recorded value of hidden states and cell states over different normal environments and poisoned environments. Mean values of the cell state vectors and hidden state vectors for normal behavior and poisoned behavior are calculated respectively. In the end, we applied a t-SNE on the mean vectors from different trials. Detailed results are shown in Figure 6. From the figure, we discover that hidden state vectors and cell state vectors are quite different over normal behaviors and poisoned behaviors; thus, monitoring the internal states online and perform anomaly detection should provide some hints for the attack prevention. In this situation, the monitor will play a role similar to immune system, where if an agent is affected by the trigger, then the monitor detects and neutralizes the attack. Although we did not observe the type 3 response in clean agents in current experiments, we anticipate that some peculiar grid arrangements will require the type 3 response in clean agents too, e.g. if agent has to take a long U-turn when it gets stuck. Thus, presence of the type 3 response will not be a sufficient indicator to detect backdoor agents. An alternate static analysis approach could be to analyze the distribution of the parameters inside LSTM. Compared with the clean agents, the backdoor agents seem to use more cell units to store information. This might be reflected in the distribution of the parameters. However, more work is needed to address detection and instill resilience against such strong attacks.

Potential Challenges and Future Research

Multiple challenges exist that require further research. From the adversary's perspective, merging multiple policies into a single neural network model is hard due to catastrophic forgetting in neural networks (Kirkpatrick et al. 2017). An additional challenge is the issue of unintentional backdoor activation, where some unintentional patterns (or adversarial examples) could also activate or deactivate the backdoor policy and the adversary might fail in its objective.

From the defender's perspective, it is hard to detect existence of the backdoor before a model is deployed. Neural networks by virtue of being black-box models prevent the user from fully characterizing what information is stored in a neural network. It is also difficult to track when the trigger appears in the environment (e.g. a yellow sticky note on a Stop sign from (Gu, Dolan-Gavitt, and Garg 2017)). Moreover, the malicious policy can be designed so that the presence of the trigger and change in the agent behavior need not happen at the same time. Considering a backdoor model as a human body and the trigger as a virus, once the virus enters the body, there might be an incubation period before the virus affects the body and symptoms begin to appear. A similar process might apply in this type of attack. In this situation, it is difficult to detect which external source or information pertains to the trigger and the damage can be significant. Future work will also address: (1) How does one detect existence of the backdoor in an offline setting? Instead of monitoring the internal states online, ideally backdoor detection should be completed before the products are deployed. (2) How can one increase sensitivity of the trigger without introducing too many unintentional backdoor activations? One potential solution is to design the backdoor agent in a white-box setting where adversary can manipulate the network parameters.

Conclusion

We exposed a new threat type for the LSTM networks and sequential DM agents in this paper. Specifically, we showed that a maliciously-trained LSTM network-based RL agent could have reasonable performance in a normal environment, but in the presence of a trigger, the network can be made to completely switch its behavior and persist even after the trigger is removed. Some empirical evidence and intuitive understanding of the phenomena was also discussed. We also proposed some potential defense methods to counter this category of attacks and discussed avenues for future research. We hope that our work will inform the community to be aware of this type of threat and will inspire to together have better understanding in defending against and deterring these attacks.

References

Bagdasaryan, E.; Veit, A.; Hua, Y.; Estrin, D.; and Shmatikov, V. 2018. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459*.

Bakker, B. 2002. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*, 1475–1482.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *AAAI*, volume 94, 1023–1028.

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

Cheng, Y., and Zhang, W. 2018. Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. *Neurocomputing* 272:63–73.

Dai, J.; Chen, C.; and Guo, Y. 2019. A backdoor attack against LSTM-based text classification systems. *arXiv preprint arXiv:1905.12457*.

Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. OpenAI baselines. https://github.com/openai/baselines.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Gu, T.; Dolan-Gavitt, B.; and Garg, S. 2017. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR* abs/1708.06733.

Hausknecht, M., and Stone, P. 2015. Deep recurrent Q-learning for partially observable MDPs. *CoRR, abs/1507.06527* 7(1).

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

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

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2016. Reinforcement learning with unsupervised auxiliary tasks. *CoRR* abs/1611.05397.

Jay, N.; Rotman, N. H.; Godfrey, P.; Schapira, M.; and Tamar, A. 2018. Internet congestion control via deep reinforcement learning. *arXiv preprint arXiv:1810.03259*.

Kiourti, P.; Wardega, K.; Jha, S.; and Li, W. 2019. TrojDRL: Trojan attacks on deep reinforcement learning agents. *arXiv preprint arXiv:1903.06638*.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Aacademy of Sciences* 114(13):3521–3526.

Kuhnle, A.; Schaarschmidt, M.; and Fricke, K. 2017. Tensorforce: a TensorFlow library for applied reinforcement learning. Web page.

Lample, G., and Chaplot, D. S. 2016. Playing FPS games with deep reinforcement learning. *CoRR* abs/1609.05521.

Lin, Y.-C.; Hong, Z.-W.; Liao, Y.-H.; Shih, M.-L.; Liu, M.-Y.; and Sun, M. 2017. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*.

Liu, Y.; Ma, S.; Aafer, Y.; Lee, W.-C.; Zhai, J.; Wang, W.; and Zhang, X. 2017. Trojaning attack on neural networks.

Liu, K.; Dolan-Gavitt, B.; and Garg, S. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. *arXiv preprint arXiv:1805.12185*.

Liu, Y.; Xie, Y.; and Srivastava, A. 2017. Neural trojans. In *Computer Design (ICCD), 2017 IEEE International Conference on*, 45–48. IEEE.

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2970–2977.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

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

Robertson, S. 2017. Practical PyTorch: Playing gridworld with reinforcement learning. Web page.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *CoRR* abs/1707.06347.

Stimpson, D., and Ganesan, R. 2015. A reinforcement learning approach to convoy scheduling on a contested transportation network. *Optimization Letters* 9(8):1641–1657.

Su, J.; Vargas, D. V.; and Sakurai, K. 2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I. J.; and Fergus, R. 2013. Intriguing properties of neural networks. *CoRR* abs/1312.6199.

Tai, L.; Paolo, G.; and Liu, M. 2017. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 31–36. IEEE.

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