Robustifying Reinforcement Learning Agents via Action Space Adversarial Training

Kai Liang Tan[1], Yasaman Esfandiari[1], Xian Yeow Lee[1], Aakanksha[2] and Soumik Sarkar*[1]

Abstract— **Adoption of machine learning (ML)-enabled cyber-physical systems (CPS) are becoming prevalent in various sectors of modern society such as transportation, industrial, and power grids. Recent studies in deep reinforcement learning (DRL) have demonstrated its benefits in a large variety of data-driven decisions and control applications. As reliance on ML-enabled systems grows, it is imperative to study the performance of these systems under malicious state and actuator attacks. Traditional control systems employ resilient/fault-tolerant controllers that counter these attacks by correcting the system via error observations. However, in some applications, a resilient controller may not be sufficient to avoid a catastrophic failure. Ideally, a robust approach is more useful in these scenarios where a system is inherently robust (by design) to adversarial attacks. While robust control has a long history of development, robust ML is an emerging research area that has already demonstrated its relevance and urgency. However, the majority of robust ML research has focused on perception tasks and not on decision and control tasks, although the ML (specifically RL) models used for control applications are equally vulnerable to adversarial attacks. In this paper, we show that a well-performing DRL agent that is initially susceptible to action space perturbations (e.g. actuator attacks) can be robustified against similar perturbations through adversarial training.**

I. INTRODUCTION

Data-driven and learning-based methods are increasingly being applied to cyber-physical systems (CPS) with ubiquitous sensing and advancements in data analytics algorithms. Recent studies have demonstrated the feasibility of deep reinforcement learning (DRL) paradigms being applied on CPS as a controller [1], [2], [3]. The success of these RL paradigms are mainly attributed to the advent of deep neural networks (DNN) that act as expressive decision-making policies. Consequently, adversarial attacks on CPS are inevitable as studies reveal the vulnerability of DNN to adversarial attacks, hence compromising the reliability of RL-based controllers. Success of adversarial attacks in breaking DNNs questions their validity, especially in life- and safety-critical applications such as self-driving cars [4]. The threat caused by attacks on DNNs was first detected while white-box attacks (*i.e.* attacks that are crafted based on a prior knowledge about the model architecture, hyper-parameters, etc.) were being studied [5], [6], [7], [8], [9]. Since then, it has also been shown that transfer attacks (*i.e.* attacks that were crafted for one DNN architecture and mounted on a different DNN architecture) are capable of breaking DNNs [10].

Researchers have proposed different methods to defend against adversarial attacks. The most popular method amongst them is adversarial training, where the DNN is trained with an adversarially perturbed dataset [9], [11], [12]. A more stable adversarial training decouples the min-max problem in the robust optimization problem and solves the problem using Danskin's theorem [13]. This requires finding the worst case perturbation at each training epoch and updating the model parameters using the dataset which has been augmented by the corresponding worst case attacks. Several methods such as the fast gradient sign method (FGSM) [6], Carlini-Wagner (CW) [5], projected gradient descent (PGD) [11], tradeoff-inspired Adversarial defense via surrogate-loss minimization (TRADES) [14], and Stochastic Saddle-point Dynamical System (SSDS) [12] approach are a few such examples which were proposed to find the worst case attack.

In contrast to the defense schemes proposed for learning-based methods, other methods rooted in classical control theory have also been developed to counter adversarial perturbations applied on classical controllers [15], [16]. In control theory, the notions of robust control and resilient control have been extensively studied. While robust controllers attempt to remain stable and perform well under various (bounded) uncertainties [17], resilient controllers try to bring back the system to a gracefully degraded operating condition after an adversarial attack [18]. However, similar notions of robustness and resilience have not been studied well for RL-based controllers, except a few recent works [19], [20].

In this study, we investigate the possibility of developing DRL-based controllers that are robust against adversarial perturbations (within specific attack budgets) to the action space (e.g., actuators). Specifically, we develop an algorithm that trains a robust DRL agent via action space adversarial training based on our previous work on gradient-based optimization on action space attacks (MAS-attacks) [21]. We would also like to highlight that developing a resilient architecture that is able to recover from adversarial perturbations may not be tractable as DRL algorithms are inherently trajectory-driven. This is because these complex nonlinear models tend to diverge significantly when they encounter an undesirable trajectory.

* Corresponding author

[1]Kai Liang Tan, Yasaman Esfandiari, Xian Yeow Lee, and Soumik Sarkar are with Dept. of Mechanical Engineering, Iowa State University, Ames, IA 50011-2030 (kailiang, yasesf, xylee, soumiks)@iastate.edu

[2]Aakanksha is with Dept. of Computer Science and Engineering, ASET, Amity University Uttar Pradesh, Sector 125, Noida, Uttar Pradesh 201313 aakanksha@student.amity.edu

II. RELATED WORKS

While the robustification of DRL agents under state space attacks and mitigation of actuator attacks in CPS have been studied in both control and ML literature, robustification of DRL agents against actuator attacks have been relatively less studied. In this section, we divide our literature review into control-theory based defense schemes, adversarial attack and defense on DNN, and robustification DRL agents.

A. *Control-Theory Based Defense*

Methods to mitigate or improve resiliency of classic controllers against actuation attacks have been extensively studied. For example, authors of [15] proposed a distributed attack compensator which has the capability to recover agents that are under attacks by estimating the nominal behaviors of the individual agent in a multi-agent setting. On the other hand, numerous studies have also developed theoretical bounds on a system's ability to recover from adversarial perturbations and designed corresponding solutions such as decoupling state estimates from control [16] and [22] combining a filter, perturbation compensator, and performance controller to re-stabilize a system. Additionally, it has been shown that actuation attacks may go undetected if the attacks are deployed at a higher frequency than sensor sampling frequencies [23], but these attacks can be mitigated with controllers with multi-rate formulations [24].

B. *Adversarial Attacks and Defenses*

After [8] exposed the vulnerabilities of DNNs to adversarial attacks, a defense strategy was proposed by [7], which defines a regularization term in the classifier. Concurrently, [6] introduced Fast Gradient Sign Method (FGSM) which was used by [9] to craft a defense strategy against iterative FGSM attacks. Departing from the notion of white-box attacks, [10] showed the vulnerability of DNNs to transfer attacks in which no prior knowledge about the model architecture is needed. Before adversarial training was popularized as a defense method [11], [25], [6], researchers proposed several other defense approaches including defining a network robustness metric [26] and using de-noising auto-encoders to form Deep Constructive Networks [27]. After adversarial training gained popularity, [28] proposed defensive distillation as another powerful method for defense, although [5] devised multiple loss functions to produce attacks that could break this defense mechanism. Since DRL algorithms employ the use of DNNs as policy functions, those models are also found vulnerable to the attacks described above [29].

C. *Robust Deep Reinforcement Learning*

In the context of training a DRL agent to be robust against state space attacks where the input to the DRL agent is perturbed, [30] and [31] demonstrated that a DRL agent that is robust to parameter and environmental variations can be obtained by adversarially training the agent.. In another study, [20] proposed a meta-learning framework where a meta-RL agent has access to two sub-policies. The meta-agent learns to switch between a policy that maximizes

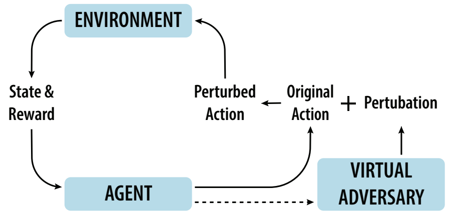

Fig. 1. Robustifying DRL agent by perturbing the original agent's action. The perturbation is generated by a white-box adversary, where the adversary has access to the agent's network architecture and parameters.

reward during nominal conditions and another policy that mitigates and copes with adversarially perturbed states by observing advantage estimates of both policies during deployment. [32] used a different robustifying scheme by formulating a zero-sum min-max optimization problem where the DRL agent is trained in the presence of another DRL agent that adversarially perturbs the system. Similarly, [19] demonstrated that DRL agents can be robustified against disturbance forces by training the agent with some noise perturbation in a min-max formulation.

III. METHODOLOGY

We provide a brief overview of DRL algorithms, followed by discussion of robust learning from a robust optimization standpoint. Finally, we formulate the robust RL methodology by combining the RL and robust learning formulation.

A. *Reinforcement Learning*

In RL, the goal of an agent is to maximize its cumulative future rewards. A typical setup involves an agent interacting with an environment for a finite number of steps, or until a termination condition is met. Upon termination of an episode, the environment resets to an initial state for the agent and repeats the process again. At each step in the environment $t \in T$, the agent receives a state $s_t \in S$ and reward $r_t \in R$, then selects an action $a_t \in A(s)$ from the agent's policy π. T denotes the finite number of steps for an environment, S denotes all possible states for an environment, R denotes the cumulative reward for one episode, and $A(s)$ denotes all possible actions conditioned upon the state. Every state observation is quantified with a reward value indicating how valuable that state is for the agent. Specifically, given a finite number of time steps T, the agent's goal is to learn an optimal policy which maximizes the cumulative discounted rewards G_t:

$$G_t = \sum_{t=0}^{T} R = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1} r_t \quad (1)$$

where gamma $\gamma \in [0, 1)$. A γ-value of 0 makes the agent nearsighted (prefer short-horizon rewards), while a value of 1 makes the agent farsighted (prefer long horizon rewards). Since being in a specific state is a direct result of previous state and action, the agent's policy will evolve over time to refine the understanding of good and bad trajectories

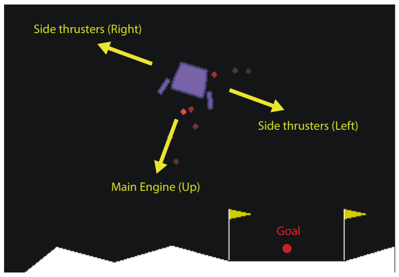

Fig. 2. The goal of the agent is to land at the goal. Annotated directional arrows indicate thrust direction of the lunar lander

τ (sequence of state-action combination). Ultimately, the goal is to optimize the agent's policy $\pi^*(a|s)$ such that the mapping between state and action is optimal.

There are two known methods to optimize a policy, namely action-value and policy gradients. Action-value methods optimize the action value for each state-action pair $Q^\pi(s,a)$ as shown in Eq. 2. Examples of action-value methods include Deep Q-Network (DQN) [33] and DoubleDQN (DDQN) [34].

$$Q^\pi(s_t,a_t) = \max_\pi \mathbb{E}\big[G_t|s_t,a_t\big] \tag{2}$$

Policy gradient methods optimize a policy parameterized by theta $\theta, \pi_\theta(a|s)$, where θ is directly optimized to maximize the expected reward function $J(\theta)$:

$$J(\theta) = \sum_{s_t \in S} d^\pi(s_t) \sum_{a_t \in A} \pi_\theta(a_t|s_t) Q^\pi(s_t,a_t) \tag{3}$$

$d^\pi(s_t)$ denotes the stationary distribution of Markov chain [35] for π_θ. Examples of policy gradient methods include Trust Region Policy Optimization (TRPO) [36] and Proximal Policy Optimization (PPO) [37].

B. Robust Optimization Classical Formulation

The robust optimization problem can be defined as [11]:

$$\mathcal{RO} := \min_w \ \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{B}} \ L(w, x+\delta, y)\right] \tag{4}$$

where $x \in \mathbf{R}^m$ is the dataset under a data distribution $\mathcal{D}$ with set of labels, y. The loss function (*e.g.* cross-entropy loss) with additive perturbations $\delta \in \mathcal{B}$ is denoted by $L(w, x+\delta, y)$ with $w \in \mathbf{R}^n$ as the model parameters (decision variables). Here the saddle point problem is looked at as a composition of an inner maximization and an outer minimization, where the inner maximization tries to find a specific perturbation for each data point x such that the overall loss is maximized. In parallel, the outer minimization aims to achieve the model parameters which minimize the corresponding adversarial loss.

The next step is to define the attack model.We introduce a specific perturbation $\delta \in \mathcal{B}$ for each data point x, where $\mathcal{B}$ is the set of allowed perturbations ($\mathcal{B} \in \mathbf{R}^m$). This set acts as a normalization for the perturbation power. For example, l_∞ ball around x is a popular way to define the perturbation budget [6]. There are several attack methods to find the corresponding adversary δ for each data point x (*i.e.* FGSM, PGD, etc.). In this paper, we use PGD for finding the adversaries [11] which uses α as the step-size, and can be formulated as:

$$x_{k+1} = x_k + \underbrace{\alpha \ sgn(\nabla_x L(w,x,y))}_{\delta} \tag{5}$$

For training a neural network, Stochastic Gradient Descent method is used for solving the outer minimization problem at the maximizer point of the inner problem. This approach is valid as Danskin's theorem proves that the gradients at the inner maximizer act as a valid descent direction for outer loss minimization [13].

C. Robust Reinforcement Learning Agents

To achieve a robust DRL agent, the robust optimization is formulated as a DRL problem. In this paper, we focused on white-box attacks in action-space. As such, the robust optimization problem can be written as:

$$\mathcal{RO} := \max_\theta \ \mathbb{E}_{(s,a)} \left[\min_{\delta\in\mathcal{B}} \ R(s_t, a_t+\delta_t)\right] \tag{6}$$

where $(s,a) \in \mathbf{R}$ are state and action pairs. The reward function with additive perturbations $\delta_t \in \mathcal{B}$ is denoted by $R(s_t, a_t+\delta_t)$ where (s,a) are updated based on the policy $(\pi_\theta(a|s))$ in each iteration, with θ as the model parameters. Note that this formulation is different from the classical $\mathcal{RO}$ formulation (Eq. 4) in which model parameters are an explicit input to the loss function, whereas in RL the reward function is not an explicit function of model parameters.

The attack formulation in RL is based on our previous work with MAS-attack [21]. MAS-attacks are derived from white-box attacks in action space, where the perturbations δ_t for each (s_t,a_t) are computed based on complete knowledge of the policy of the agent, π_θ (See Fig. 1). Each perturbation δ_t is bounded by $\mathcal{B} \in \mathbf{R}$, by projecting the perturbations back into $\mathcal{B}$. As MAS-attack uses PGD to iteratively find the perturbations, we can formulate the attack as:

$$a_{k+1} \ = \ a_k - \alpha \ \nabla_{a_k}\mathcal{D} \tag{7}$$

where k is the iteration number, α is the step size and $\mathcal{D}$ is the action distribution obtained from the agent's policy π_θ. Here, we note that while the robust optimization formulation was formulated using the reward function, in reality, the reward function is unknown to the DRL agent or the virtual adversary. Hence, the reward function is approximated by the reward-maximizing action distribution $\mathcal{D}$ for gradient computations [21]. After obtaining the adversarial perturbations and adding it to the nominal actions, we train the DRL agent using standard policy gradient methods, which solves the outer maximization problem in the formulation above. Another distinction we would like to note is that while a k-step PGD is usually used to compute the adversarial attacks, with k being fixed, we do not adhere to that procedure. Instead, we define a tolerance ϵ and keep iterating through the PGD process until the adversarial actions saturate. This ensures that the computed adversarial action actually corresponds to bad action rather than being approximated by a fixed k-number of gradient steps. The adversarial training algorithm to robustify the DRL agent is shown in Alg. 1.

Algorithm 1: MAS-Adversarial Training

```
Input: episodes $N$, episodic limit $T$, step size $\alpha$,
       convergence criteria $\epsilon$, budget $B$
Initialize state $s$, policy $\pi_\theta$
for $n \in \{1, ..., N\}$ do
    for $t \in T$ do
        Get action distribution $D$ from $\pi_\theta(s_t)$
        Sample $a_t$ from $D$
        $a_k = a_t$
        Sample $a_{k+1}$ from $D$
        while $a_{k+1} - a_k \geq \epsilon$ do
            $a_{k+1} = a_k - \alpha \nabla_{a_k} D$
        end
        $\hat{\delta}_t = P_{\mathcal{B}}(a_{k+1} - a_t)$
        $\hat{a}_t = a_t + \hat{\delta}_t$
        Step through environment with $\hat{a}_t$
    end
    if Time to update then
        Update agent's policy $\pi_\theta$
    end
end
```

IV. Experiments

A. Environment Setup

Experiments were conducted in OpenAI gym's Lunar Lander environment [38], where the goal is to land the lander safely while minimizing thruster usage. The state space of the environment consist of eight continuous values: x-y coordinates of the lander, velocity in x-y components, angle and angular velocity of the lander, and a boolean contact variable for left-right lander legs (*e.g.* 1 for contact, 0 for no contact). The action space available are two continuous vectors [-1,1]. First vector controls the up-down engine, where values within [-1,0] turns off the engine while values within (0,1] maps to 50% - 100% of engine power. Second vector controls left-right engines, where [-1,-0.5] and [0.5,1] controls left and right engine respectively.

For this environment, the agent's goal is to maximize reward. Given an arbitrary starting point (as seen in Fig. 2), the RL agent has to land on the landing pad without crashing. The agent receives positive rewards (*e.g.* between 100 to 140) for landing. The agent will incur negative rewards if the lander moves away from the landing pad. Each successive landing leg contact gives 10 rewards each. Firing the up-down engine cost -0.3 rewards each step. The episode is terminated if the lander crashes or is at rest, which gives -100 and 100 rewards respectively.

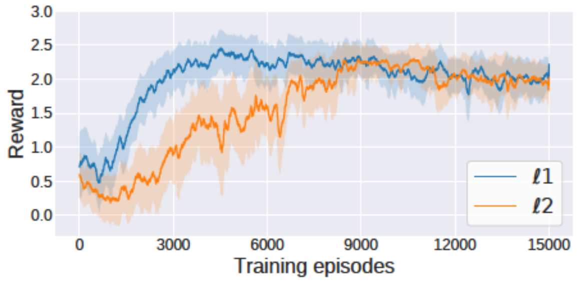

Fig. 3. Training plots on robustly trained DRL agents with MAS-attacks ($\ell 1$ and $\ell 2$ projection). Seven agents was trained for each projection method. Each rewards are averaged across all agents, followed by a moving average.

TABLE I
SUMMARY STATISTICS OF AGENT'S REWARDS

	Environment	Nominal		Adversarial	
$\ell 1$	Agent	Nominal	Robust	Nominal	Robust
	Reward	2.21 ± 0.74	2.21 ± 0.54	0.74 ± 1.33	2.39 ± 0.47
	Environment	Nominal		Adversarial	
$\ell 2$	Agent	Nominal	Robust	Nominal	Robust
	Reward	2.21 ± 0.74	1.74 ± 0.63	0.60 ± 1.10	2.00 ± 0.61

B. Deep Reinforcement Learning Training and Parameters

For this experiment, we trained a PPO agent with an Actor-Critic architecture [39]. In this architecture, both the actor and the critic share the same network of multi-layer perceptrons made up of dense layers. The actor has an additional dense layer which outputs the policy in the form of a Gaussian model and the critic estimates the value of the action chosen by the actor. To encourage exploration, an entropy term is also added to the loss function which consists of the actor's loss and the critic's loss. The loss function is then jointly optimized using Adam optimizer.

V. Results and Discussion

We conducted two different experiments to study the difference in robustifying DRL agents by using $\ell 1$ and $\ell 2$ projection methods with MAS-attack budget of 1 and step size of 3 within Algorithm 1.

A. Convergence of Robust Agent

In this section, we analyzed the convergence plots (Fig. 3) of the robust agent empirically. The robust agent was trained with $\ell 1$ and $\ell 2$ perturbations at every step for 15000 episodes. For each projection method, we trained seven DRL agents with the same network architecture and parameters across different seeds. After 15000 episodes, training rewards are averaged across seven agents, followed by a moving window of 100 episodes to obtain results shown in Fig. 3. Agents trained with $\ell 1$ converged faster to a higher reward at approximately 4500 episodes as compared to training with $\ell 2$, which stabilized around 9000 episodes. This reveals that it is slightly harder to robustify the agent against $\ell 2$ as compared to $\ell 1$ due to $\ell 2$ attacks being more distributed along action dimensions compared to $\ell 1$.

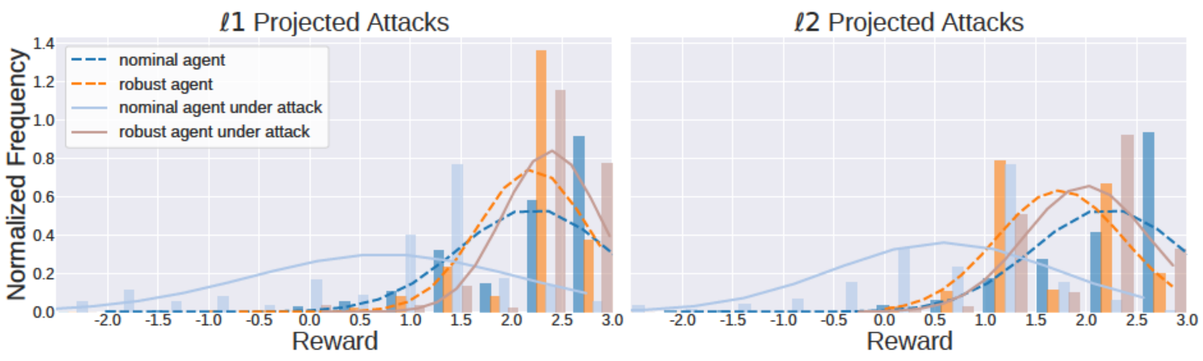

Fig. 4. A comparison of reward distributions between a nominally trained agent and adversarially trained agent for both $\ell1$ and $\ell2$ MAS attacks. We observe that the distribution of rewards for a nominal agent shifts to the left and have long tails in the regions with negative rewards when subjected to attacks. In comparison, the distributions of rewards remain similar when the robustly trained agents are subjected to attacks.

B. *Performance Comparison*

We compare the performance of the robustly trained DRL agents against the nominally trained DRL agent. Specifically, we tested both DRL agents under two environment settings; nominal and adversarial environment. A nominal environment denotes scenarios where the agent is not attacked, while the agent is attacked in adversarial environments. Fig. 4 shows a histogram of all four possible scenarios for each projection attack. For both projection attacks, the nominal agent tested in the nominal environment is identical. The other three scenarios show slightly different results due to different projection attacks. We summarize the reward distribution of each scenario in Table I. Rewards within the range of 2.5 to 3.0 are successful landing experiments, where the DRL agent learns to land on the landing pad and shuts off it's engines. Rewards between 1.0 to 2.5 are DRL agents that lands successfully but continuously loses rewards as it fails to turn off the engine. Rewards below 1.0 are experiments where the lunar lander crashed or flew out of frame.

1) Nominal environment: In nominal environments, the nominal agent's performance is identical across both $\ell1$ and $\ell2$ plots in Fig. 4 which is expected, as the agent is not attacked. As a result, the agent has a mean reward of 2.21 ± 0.74. This indicates the nominal agent successfully lands and turns off its engine.

Next, we evaluate both the robust agent trained with $\ell1$ and $\ell2$ projection attacks in the nominal environment. The rewards for the robust agent in $\ell1$ and $\ell2$ are 2.21 ± 0.54 and 1.74 ± 0.63 respectively. We hypothesize that the nature of $\ell2$ crafted attacks are more evenly distributed across action dimension, hence it is harder to train and test against $\ell2$ attacks where else $\ell1$ distribute attacks into one dimension. These results are counterintuitive to the notion of robustifying a DRL agent, where the expected results of a robust agent should be higher than the nominal agent in the nominal environment. Interestingly, the same behavior has been observed when robustifying DNN used for image classification as demonstrated in [11].

2) Adversarial environment: In the adversarial environment, the nominal agent's performance in both $\ell1$ and $\ell2$ projection attacks dropped significantly as anticipated. The nominally trained agent's policy was trained in environments with no perturbations. The rewards for both $\ell1$ and $\ell2$ projection attacks are 0.74 ± 1.33 and 0.60 ± 1.10 respectively. Hence, both projection attacks successfully minimized the nominal agent's reward. In both $\ell1$ and $\ell2$ attacks, we observed a high frequency of rewards obtained within the range of 1.0 and 1.5, which corresponds to scenarios where the lander landed but failed to turn off its engine.

For the robust agent, the performance of both $\ell1$ and $\ell2$ trained agent increased when compared to the nominally tested robust agent counterpart. The rewards for $\ell1$ and $\ell2$ trained agents are 2.39 ± 0.47 and 2.00 ± 0.61 respectively. Similar to counter-intuitive observations made earlier, we note that the expected results should be a decrease in rewards compared to the robust agent in the nominal environment. These counter intuitive results reveal an important characteristic of adversarial training defense schemes. We can expect that an agent that has been adversarially trained will perform well when tested in an adversarial environment, but at the cost of a slightly reduced performance when tested in nominal situations.

Although it is not a direct comparison, it is interesting to note that the robust agent trained with $\ell1$ projected attacks in the adversarial environment outperforms the nominal agent in the nominal environment. This is likely because the nominal agent can only explore and maximize its reward with familiar trajectories seen during training. For the robust $\ell1$ agent, the agent's policy explored many more trajectories with the help of adversarial perturbations. Therefore, it has likely found other trajectories with much higher rewards that the nominal agent did not explore.

VI. CONCLUSIONS

Deep RL based controllers are increasingly popular as they demonstrate a potential for controlling complex CPS. Adversarial attacks on these controllers are emerging, which requires these controllers to be robustified against these attacks. In this work, we formulate the problem of robustifying a DRL agent as a robust optimization problem. We adversarially trained a DRL agent that is subjected to action space perturbations and demonstrate that it still performs robustly in the presence of actuator perturbations. In some cases, it even improved the performance of the agent in the absence of attacks. Hence, we show that it is beneficial to adversarially train a DRL agent. Future direction includes extending this work to different attack models and experimenting with transferability of attacks and defense results.

ACKNOWLEDGMENT

This work was supported in part by NSF grant CNS-1845969.

REFERENCES

[1] N. Lazic, C. Boutilier, T. Lu, E. Wong, B. Roy, M. Ryu, and G. Imwalle, "Data center cooling using model-predictive control," in *Advances in Neural Information Processing Systems*, 2018, pp. 3814–3823.

[2] H. Zhang, H. Jiang, Y. Luo, and G. Xiao, "Data-driven optimal consensus control for discrete-time multi-agent systems with unknown dynamics using reinforcement learning method," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 5, pp. 4091–4100, May 2017.

[3] K. L. Tan, S. Poddar, S. Sarkar, and A. Sharma, "Deep reinforcement learning for adaptive traffic signal control," in *ASME 2019 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection, 2019.

[4] C. Sitawarin, A. N. Bhagoji, A. Mosenia, M. Chiang, and P. Mittal, "Darts: Deceiving autonomous cars with toxic signs," *arXiv preprint arXiv:1802.06430*, 2018.

[5] 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.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[7] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[9] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[10] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. ACM, 2017, pp. 506–519.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[12] Y. Esfandiari, K. Ebrahimi, A. Balu, N. Elia, U. Vaidya, and S. Sarkar, "A saddle-point dynamical system approach for robust deep learning," *arXiv preprint arXiv:1910.08623*, 2019.

[13] J. M. Danskin, "The theory of max-min, with applications," *SIAM Journal on Applied Mathematics*, vol. 14, no. 4, pp. 641–664, 1966.

[14] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan, "Theoretically principled trade-off between robustness and accuracy," *CoRR*, vol. abs/1901.08573, 2019. [Online]. Available: http://arxiv.org/abs/1901.08573

[15] A. Mustafa and H. Modares, "Attack analysis and resilient control design for discrete-time distributed multi-agent systems," *CoRR*, vol. abs/1801.00870, 2018. [Online]. Available: http://arxiv.org/abs/1801.00870

[16] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," *IEEE Transactions on Automatic Control*, vol. 59, no. 6, pp. 1454–1467, June 2014.

[17] K. Zhou, J. C. Doyle, K. Glover, *et al.*, *Robust and optimal control*, vol. 40.

[18] D. Wei and K. Ji, "Resilient industrial control system (rics): Concepts, formulation, metrics, and insights," in *2010 3rd International Symposium on Resilient Control Systems*, Aug 2010, pp. 15–22.

[19] C. Tessler, Y. Efroni, and S. Mannor, "Action robust reinforcement learning and applications in continuous control," *arXiv preprint arXiv:1901.09184*, 2019.

[20] A. Havens, Z. Jiang, and S. Sarkar, "Online robust policy learning in the presence of unknown adversaries," in *Advances in Neural Information Processing Systems*, 2018, pp. 9916–9926.

[21] X. Y. Lee, S. Ghadai, K. L. Tan, C. Hegde, and S. Sarkar, "Spatiotemporally constrained action space attacks on deep reinforcement learning agents," *arXiv preprint arXiv:1909.02583*, 2019.

[22] X. Huang and J. Dong, "Reliable control policy of cyber-physical systems against a class of frequency-constrained sensor and actuator attacks," *IEEE Transactions on Cybernetics*, vol. 48, no. 12, pp. 3432–3439, Dec 2018.

[23] J. Kim, G. Park, H. Shim, and Y. Eun, "Zero-stealthy attack for sampled-data control systems: The case of faster actuation than sensing," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec 2016, pp. 5956–5961.

[24] H. Jafarnejadsani, H. Lee, N. Hovakimyan, and P. Voulgaris, "A multirate adaptive control for mimo systems with application to cyber-physical security," in *2018 IEEE Conference on Decision and Control (CDC)*, Dec 2018, pp. 6620–6625.

[25] U. Shaham, Y. Yamada, and S. Negahban, "Understanding adversarial training: Increasing local stability of supervised models through robust optimization," *Neurocomputing*, vol. 307, pp. 195–204, 2018.

[26] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/6339-measuring-neural-net-robustness-with-constraints.pdf

[27] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014.

[28] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.

[29] V. Behzadan and A. Munir, "Vulnerability of deep reinforcement learning to policy induction attacks," in *International Conference on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2017, pp. 262–275.

[30] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, "Robust deep reinforcement learning with adversarial attacks," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2040–2042.

[31] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3932–3939.

[32] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2817–2826.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[34] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[35] W. Hendricks, "The stationary distribution of an interesting markov chain," *Journal of Applied Probability*, vol. 9, no. 1, pp. 231–233, 1972.

[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[38] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[39] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.