Transferable Adversarial Attacks on Deep Reinforcement Learning with Domain Randomization

Xinlei Pan[1], Yulong Cao[2], Xindi Wu[3], Eric Zelikman[4], Chaowei Xiao[2],
Yanan Sui[4], Rudrasis Chakraborty[1], Ronald S. Fearing[1]
[1]*University of California, Berkeley* [2]*University of Michigan, Ann Arbor* [3]*Carnegie Mellon University* [4]*Stanford University*
{xinleipan,rudra,ronf}@berkeley.edu, {yulongc,xiaocw}@umich.edu,
xindiwu@andrew.cmu.edu, {ezelikman,ysui}@cs.stanford.edu

Abstract

Secure and robust deep reinforcement learning (DRL) is necessary to deploy DRL algorithms in real world applications. However, previous work shows that DRL policies are vulnerable to adversarial attacks. In order to study the vulnerability/robustness of DRL algorithms, previous work has explored various attacks against DRL policies assuming that the attacker has access to the original policy either in a white-box manner or black-box manner. However, the realizability of these attacks is limited as assuming access to the original training environment or the policy could sometimes be impossible. In this study, we propose a set of novel adversarial attack approaches against DRL policies based on domain randomization, and we do not have the assumption of access to the exact original training environment nor the original policy, nor the possibility of querying the exact original policy. We first systematically analyze the space of transferable adversarial attacks against DRL when the attacker has almost no knowledge about the original training environment information such as system dynamics, action space, reward function, and/or information about the trained policy such as the algorithms and network structure. Then we train an attacker on multiple different environments with different dynamics, action space and reward settings, and also with different RL algorithms. We separately evaluate the effectiveness of the proposed attack when the environment changes or the algorithm used to train the pristine model changes. We compare our method with traditional adversarial attacks to show the improved transferability.

1. Introduction

Reinforcement learning (RL) is a powerful and increasingly popular approach for solving sequential decision-making problems such as computer-games [21, 14, 15] and robot control [8, 10]. However, recent research indicates that RL agents are vulnerable to adversarial attacks: small, targeted perturbations to their inputs lead to poor performance [11, 18, 7, 9, 25]. Previous adversarial attacks on deep RL can usually be categorized to white-box and "black-box" approaches, which hold various assumptions about the victim models. However, all of the methods under both settings need to have at least partial knowledge of the target agent, making the algorithms less realistic in the real-world applications. For instance, in white-box approaches, the attacker often has access to the full model [7] to generate the adversarial examples. In black-box approaches, the attacker needs to either train an alternative victim model by imitating the target model [25], know the target policy environments to train a new policy [7], or generate perturbations using finite difference by querying the model [25] at testing time.

We aim to find possible adversarial attacks that have more realistic assumptions about the target model. Considering the components of Markov decision processes (MDP): MDPs are composed of the observation space, the action space, the reward function, and the state transition function. Black-box methods usually assume knowing full or partial information about the victim policies' training environment's MDP components. However, this assumption sometimes can be infeasible. While in some cases, some information may be available, we are interested in exploring whether it would be possible to perform adversarial attack when the attacker has much less information about the victim policies' MDP components and the victim policy.

In this work, we leverage domain randomization [22, 19], a widely-used generalization improving approach, to generate adversarial attacks that can generalize and transfer. Namely, we train attackers on multiple domains, and rely on domain randomization to generalize the attack into new domains. Transferable adversarial attacks on computer vision models have been demonstrated before [12, 24, 16]. In this work, we propose a neural network based "plug-and-play" attack with more realistic assumptions about the victim model

and can transfer when the attacker has limited knowledge. We analyze the vulnerability of various aspects of the environmental information (MDP's components), and the extent to which an agent can be successfully attacked when the attacker lacks information about them.

Our main contributions are as follows: First, we identify the problem of cross-domain transferable attacks in DRL, especially in the cases where the attacker does not know the original training environment's action space, transition dynamics, reward functions, policy type and/or network structure; second, we propose to use domain randomization to train adversarial attacks against multiple models with different training environment settings and policy settings; third, our trained attacker successfully attacks unseen policies, and it improves the attack effects non-trivially over the attack that does not use domain randomization.

2. Related Works

Adversarial Attacks on Deep RL. [7] and [9] showed that DRL agents are vulnerable to adversarial attacks: maliciously crafted small perturbation to observations with the Fast-Gradient Sign Method (FGSM) [4]. [25] proposed a network-based white-box approach to achieve a long-term attack goal by training a neural network to produce perturbations that minimize total returns instead of minimizing the rewards at every time step. Other white-box attacks target specific policies, such as [1], which exploits the Q-values produced by Deep Q Network (DQN) [14] algorithms. Black-box attacks on RL algorithms often take inspiration from the method in [17]. That is, they train an example agent on the target environment, and then rely on the tendency of attacks to transfer, as in [2]. To attack DRL agents without full knowledge of the target model, most black-box attacks leverage the transferability of adversarial attacks [7]. Another, distinct approach used by [11] is building a video prediction model and then selecting a series of actions to minimize the expected return. More recently, [3] proposed a new threat model in the multi-agent setting where the attacker controls an agent that is visible to the victim policy and acts to change the environment to attack, instead of directly perturbing the victim policy's observations. [25] also proposes a black-box approach based on finite difference to estimate gradients at testing time to perform attack.

Transferable Adversarial Attacks on RL. [7] introduced a black-box variant of the FGSM attack leveraging similar techniques, while this attack requires access to the training environment of the target model. [25] proposed an imitation learning based attack which trains a surrogate policy by only observing the target policy's outputs. While in the field of computer vision, domain shift mostly lies in the image style [16], in RL and robotics control, dynamics, actions, and rewards can be domain shift factors.

3. Preliminaries

Reinforcement Learning. We aim to propose adversarial attacks for model-free RL in this work. We assume the environment to be an MDP, consisting of its state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $\mathcal{R}$, state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, which maps action-conditioned state transitions to their transition probability. The policy is a function that maps states to actions: $\pi : \mathcal{S} \to \mathcal{A}$. The goal is to find a policy that maximizes expected return $\pi^* = \arg\max_\pi \mathbb{E}_\pi[\sum_i \gamma^i r_i|\pi]$, where γ is the reward discount factor.

Attack Goal and Transferable Attacks on DRL. The goal of the proposed adversarial attack is to reduce the final total rewards of the target victim DRL agent. We assume the attacker is able to perturb the observations of the target model. We bound the amount of perturbation by its L-∞ norm. That is for any given state s, the perturbed state is constrained by $\hat{s} \in [s - \epsilon, s + \epsilon]$. In real world applications, the attacker may have different levels of knowledge about the target model. Since previous work has explored the transferability when there are domain shifts in images [16], we explore other aspects where the attacker may have limited knowledge about the target model. We propose the following ways where the transferable attacks may exist: transfer between different observation domains, (since this transfer among the observation space has been explored by the previous work, we only listed here for the sake of completeness); transfer between different action spaces; transfer between different tasks or different reward functions; transfer between different environment transition dynamics; transfer between different policy network structures.

4. Approach

In this work, we propose a neural network based end-to-end adversarial attack with realistic assumptions of target models in order to train an attack which can transfer to different domains and attack unknown target policies. Specifically, we explore the transferable attacks mentioned previously: transferable attacks against unknown action space, reward function, transition dynamics, policy algorithm and network structure of the target policy. To achieve these goals, We train an attacker (a neural network) f that takes the observation s as input and outputs a perturbation $f(s, \theta)$, where θ is the parameter of function f. The perturbation is bounded by ϵ of its L-∞ norm and added onto the original state. Therefore, the perturbed state can be expressed as, $\tilde{s} = f(s) = s + \epsilon * Tanh(\theta(s))$, where ϵ is the L-∞ bound for the perturbation, and $Tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$.

Target RL Algorithms. We attack target policies trained with the following model-free RL algorithms: Deep Deterministic Policy Gradient(DDPG) [10], continuous Deep Q-learning with normalized advantage function (NAF) [5],

maximum entropy soft actor-critic (SAC) [6], and proximal policy optimization (PPO) [20]. For the sake of simplicity, we denote the general actor as A and critic as V, where the actor is the policy that takes in the state and outputs the action; and the critic takes in the state and outputs the value of the state: $V_\pi(s) = \mathbb{E}_\pi[\sum_i \gamma^i r_i|s]$.

Training with Domain Randomization. We propose to leverage domain randomization to improve the transferability of the adversarial attacks against DRL. In this work, in order to train an adversarial attacker that can perform successful attacks even when it has limited knowledge about the target model, we train multiple alternative victim policies that have a relatively large variety of different environment settings or training algorithm settings, and we train our attacker to attack these alternative victim models then transfer the attacker to the target model. The attacker is trained as an attack model against several alternative victim policies randomized in different aspects such as environment settings and training approaches. We first identify the possible aspects that can vary among potential alternative victim policies, then we train multiple alternative victim policies under these settings. Finally, we train our attacker on selected alternative victim policies to help with transfer.

Training Alternative Victim Policies. In order to improve the transferability of the attacker when it has limited knowledge of the target policy, we train the following sets of alternative victim policies: policies trained on environments with different action spaces, reward functions, transition dynamics. Additionally, policies trained on the same environment with different algorithms. Finally, training the same algorithm on the same environments but with different network structures. When varying one aspect of the alternative victim policy, other aspects stay the same.

Training the Attack Policies. The attacker should have access to the alternative victim models and their training environment, the policy algorithm and network structure. The overall goal of the attacker is to minimize the rewards of all alternative victim policies. Define the set of policies trained with a specific action space A_i, reward function R_i, transition dynamics T_i, policy algorithm P_i and network structure S_i as: $\pi_{A_i,R_i,T_i,P_i,S_i}$ where i is the index of the alternative victim policy. The goal of the attacker f is to minimize the following expected return: $f^* = \arg\min_f \mathbb{E}_f \sum_i [\mathbb{E}_{\pi_{A_i,R_i,T_i,P_i,S_i}(s=f(s))}[\sum_j \gamma^j r_j]]$.

Per-Policy Attacks. During the training process, the attack modifies the observed states of the target agents. We define the attack network as $f(\theta)$ with parameter θ, then the perturbed state can be expressed as $\tilde{x} = f(x) = x + \epsilon * Tanh(\theta(x))$, where ϵ is the L-∞ bound for the perturbation, and x is the original state and $\tilde{x}$ is the perturbed state. We train attackers with different policies for each of the previous three attack methods. The detailed algorithm is presented in supplementary materials.

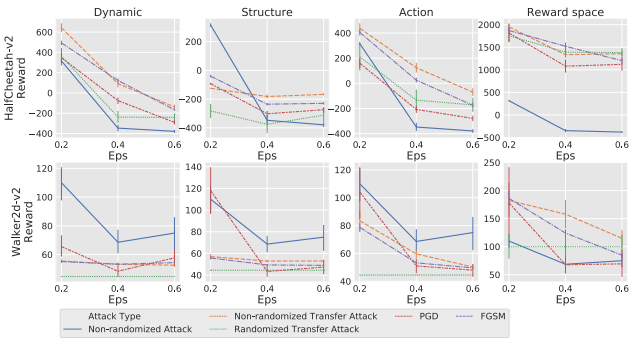

Figure 1. Results for comparing domain randomization trained attack with other baselines. We randomize the dynamics, network structure, action space, and reward space, respectively. In each subfigure, the first row corresponds to the results for HalfCheetah, and the second row corresponds to the results for Walker2d. These results are generated on attacker trained and tested with NAF trained target policies. Correspondence: Non-randomized Attack: wb-nn; non-randomized transfer attack: bb-transfer; randomized transfer attack: bb-random (ours).

Baselines.We review previous important work on attacking DRL and analyze their knowledge of the target policy. First, white-box method based on FGSM [7] assumes that they have access to the target model, while not necessarily needing to know the original training environment information. We refer to this method wb-fgsm. We also consider a PGD-based attack [13] performed in the same way, which we refer to as wb-pgd. Previous neural network based white-box attacks [25] assume that they can perform gradients updates through the target network to train the attacker network, and require access to the original environment for training the attacker. We train this attacker with all four policy algorithms and the default environment setting, and transfer the attack generated on this alternative model to the target model (wb-nn). In addition, we train an attacker with the same environment setting as the target model and with the same policy type and network structure, and apply the attack generated on the alternative victim model to the target model. We call this method bb-transfer, and the domain-randomized version of this approach bb-random. Note that bb-random is our method.

5. Experimental Evaluation

We evaluate the proposed transfer attack on two RL environments: **HalfCheetah** and **Walker2d** on MuJoco [23]. We also conduct an ablation study to understand how randomization in different domains affects the transferability of the attack.

Task Variation. To train a transferable attack, we first need to train multiple alternative victim policies with randomization. We choose 5 different aspects for the randomization: dynamics, reward functions, action space, policy

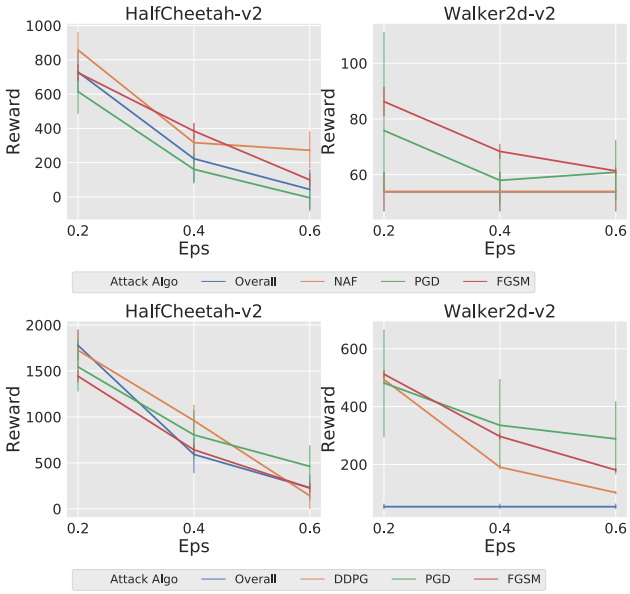

Figure 2. Experiment results on HalfCheetah and Walker2d for the overall attack that randomize everything. We then evaluate the trained attacker on target models trained with NAF and DDPG, to evaluate the cross dynamics, cross action space, cross reward functions, cross network structure and cross algorithm transferability.

algorithm, network structure. For generating randomized target policies, we train target policies with randomized 21 dynamics, 5 reward functions, 5 action space dimensions, 4 policy algorithms, and 4 network structures for each environment. Detailed setup for the randomization can be found in supplementary materials.

6. Results and Analysis

We present the results in Figures 1, and 2, by showing the final rewards of target agents under different attacks. In Figure 1, we present the results for domain randomization in different aspects. For each row, we present the results of different attack methods under unknown settings where the training environment conditions (dynamics, network structure, action space, or the reward) are randomized. The victim policy is trained with a held-out environment condition, and the attackers for our method are trained with randomized conditions. The purpose is to show the transfer can happen when the attacker does not have access to the exact victim model, it can still perturb the victim policy non-trivially. Different rows correspond to different environments, with the first row showing the results for HalfCheetah and the second row showing the results for Walker2d. Here "Non-randomized Attack" refers to the case where the attacker and target share the training environment and same algorithm setting, and use the neural network based approach as mentioned in [25] to attack; "Randomized Transfer Attack" refers to our method which trains on multiple randomized environments and transfers to other environments; "Non-randomized Transfer Attack" refers to the case where the attack and target have different environment settings, and the difference lies in dynamics, network structure, action space setting and reward function, respectively for the 1st, 2nd, 3rd and 4th figure in each row, respectively. The "PGD" attack corresponds to projected gradient descent based attack [13] and "FGSM" corresponds to fast gradient sign method based attack [4]. Both are white-box approaches. We present representative results here for using NAF for training the attacker and include other results in our supplementary materials.

We see clearly from the results that our method consistently achieves better attack effects when changes are in system dynamics, the network structure or the action space, in both HalfCheetah and Walker2d. This shows the effectiveness of the proposed transferable adversarial attack method. We also notice that sometimes the transfer attack results from our proposed method are even better than results from white-box attacks. A potential reason is that domain randomization not only improves the transferability of the attack but also might improve the robustness of the attack so that the attack can stably reduce the final rewards achieved by the target agents. We find the attack results are generally worse when the reward space is unknown. This is reasonable as different reward spaces representing different tasks introduce more significant differences to the target policies compared to varying other aspects such as system dynamics or action spaces. We also notice that, given the challenges introduced by randomizing reward spaces, our method still achieves some level of success in the "Walker2d" environment.

In Figure 2, we present the attack results for the overall attacks where the attack policies are trained with domain randomization in all aspects. From the figure we can see that our method achieves similar results as the white-box methods. We present representative results here for using NAF and DDPG for the target policies and include other results in our supplementary materials.

7. Conclusion

In this paper, we demonstrated the ability for adversarial attacks on deep RL to transfer across a variety of domain changes. We show that domain randomization can be used to improve this transferability. We also discovered that though transferring attacks across dynamics, network structure and action space shift can be easier, transferring attack across reward function shift is relatively more difficult, indicating that we can effectively protect the reward function information in order to defend against such attacks. Further research that investigates the defense techniques against these highly cross-domain transferable attacks will likely be valuable.

References

[1] Xiaoxuan Bai, Wenjia Niu, Jiqiang Liu, Xu Gao, Yingxiao Xiang, and Jingjing Liu. Adversarial examples construction towards white-box q table variation in dqn pathfinding training. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 781–787. IEEE, 2018. 2

[2] Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 262–275. Springer, 2017. 2

[3] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *International Conference on Learning Representations*, 2020. 2

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015. 2, 4

[5] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016. 2

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018. 3

[7] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *International Conference on Learning Representations*, 2017. 1, 2, 3

[8] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. 1

[9] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. *International Conference on Learning Representations Workshop*, 2017. 1, 2

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 1, 2

[11] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3756–3762, 2017. 1, 2

[12] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016. 1

[13] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 3, 4

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013. 1, 2

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1

[16] Muhammad Muzammal Naseer, Salman H Khan, Muhammad Haris Khan, Fahad Shahbaz Khan, and Fatih Porikli. Cross-domain transferability of adversarial perturbations. In *Advances in Neural Information Processing Systems*, pages 12885–12895, 2019. 1, 2

[17] Nicolas Papernot, Patrick Drew McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *2017 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2017*, pages 506–519. Association for Computing Machinery, Inc, 2017. 2

[18] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2040–2042. International Foundation for Autonomous Agents and Multiagent Systems, 2018. 1

[19] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *Robotics: Science and Systems Conference*, 2017. 1

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3

[21] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 1

[22] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017. 1

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. 3

[24] Xingxing Wei, Siyuan Liang, Ning Chen, and Xiaochun Cao. Transferable adversarial attacks for image and video object detection. *arXiv preprint arXiv:1811.12641*, 2018. 1

[25] Chaowei Xiao, Xinlei Pan, Warren He, Jian Peng, Mingjie Sun, Jinfeng Yi, Bo Li, and Dawn Song. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470*, 2019. 1, 2, 3, 4

Supplementary Materials

Xinlei Pan[1], Yulong Cao[2], Xindi Wu[3], Eric Zelikman[4], Chaowei Xiao[2],
Yanan Sui[4], Rudrasis Chakraborty[1], Ronald S. Fearing[1]
[1]*University of California, Berkeley* [2]*University of Michigan, Ann Arbor* [3]*Carnegie Mellon University* [4]*Stanford University*
{xinleipan,rudra,ronf}@berkeley.edu, {yulongc,xiaocw}@umich.edu,
xindiwu@andrew.cmu.edu, {ezelikman,ysui}@cs.stanford.edu

1. Introduction

We first present the algorithm for training attackers and the training domain randomization settings. Then, we present additional training results and training details in this supplementary materials. The content includes the mentioned training results on transferable attacks trained with DDPG, SAC, and PPO; we also include the training hyperparameters in the appendix.

2. Algorithm for Training Attackers

The detailed algorithm for training attackers in included in Algorithm 1.

3. Domain Randomization Settings

Detailed setup for the alternative victim policy domain randomization can be found in Table 1.

Table 1. Setups for training target policies with different aspects of domain randomization in terms of dynamics, reward, action, policy algorithm and network structure

Parameter	Potential Values
Action Space	4, 5, 6, 7, 8 dimensional
Reward functions	5 functions for different tasks
Dynamics	21 normally-varied sets of masses
Policy Algorithm	DDPG, NAF, SAC, PPO
Network Structure	2, 3, 4 layer neuron network

4. Additional Training Results

We present the mentioned training results in Figure 1, Figure 2, Figure 3 and Figure 4. We observe that the episodic reward of victim policies under attack exhibits a relatively large variation in different environment settings. For example, using the white-box approach PGD to attack HalfCheetah of different dynamics, action space, network structure, and reward space can have a large range of achieved episodic reward. Therefore, we choose to report the reward percentage of episodic reward relative to the episodic reward under PGD attack as the way to evaluate the attack effects of other approaches. The way we calculate this percentage is to divide the ground truth episodic reward by the absolute ground truth episodic reward under PGD attack.

We see from Figure 1 that attacks generated with randomized state transition dynamics, network structures and action spaces all consistently performs better than or on par with attacks that do not randomize. Attacks on randomized reward space sometimes have worse performance compared with non-randomized transfer attacks. This is because the

Algorithm 1 Attacking Algorithm

Input:
$V(s)$: alternative victim model's value function
$\mu(s)$: alternative victim model's actor
T: number of training steps
env: the training environment
ϵ: the perturbation bound
Initialize:
B: replay buffer
$f(\theta)$: attack network
for $t = 1$ **to** T **do**
s = env.reset()
$a = \mu(f(s))$
$s', r, done = env$.step(a)
B.append($s, a, s', r, done$)
sample = B.sample()
sample.s = f(sample.s)
loss = 0
sample.r = -1 * sample.r
loss = loss + Policy_Training_Loss(sample)
$v = V$(sample.s)
loss = loss + mean(v)
back-propagation with loss
$s = s'$
end for

reward space based transfer attack is a little bit harder since different reward designs could conflict with each other and attacks generated for one reward design may not work so well when it is applied on another reward design. In all figures, the "Non-Randomized Attack" corresponds to the attack that are trained and evaluated on the same victim policy, "Non-randomized Transfer Attack" corresponds to the attack that are trained on different alternative victim policies but are then directly evaluated on the target victim policy, "Randomized Transfer Attack" corresponds to the attack that are trained with domain randomization with dynamics randomization, network structure randomization, action space randomization and reward space randomization respectively, and "PGD" and "FGSM" correspond to white-box attacks PGD and FGSM which directly minimize each frame's value.

Figure 2 presents the results on transferring attacks between algorithms. That is, we train attackers using DDPG, NAF, SAC, and use the each of the attacker to attack all victim policies (DDPG, NAF, SAC). We also trained an attacker that trains on all three alternative victim policies (DDPG-NAF-SAC), and use this attacker to attack DDPG, NAF and SAC. The results show the episode reward when the target algorithm (victim policies) are under attack (attack algorithm). It can be seen from the results that sometimes the transferability of attacks among the algorithms is fairly significant, even when the attacker is not trained with randomization. This has been shown in previous work [1]. For example, in the results of HalfCheetah with $\epsilon = 0.2$, the attacker trained with alternative victim policy SAC has a more effective attack on DDPG than the attacker trained with alternative victim policy DDPG. This may imply that SAC has a much better performance in training the policy alone, and thus has a better exploration during training. Therefore, this might have given the attacker more insight about where the agent may be weak. This results show that randomization on the training algorithm may not necessarily always improve the attack effects since the transferability of attacks among different policies is already strong.

We present the overall randomization attack results in Figure 3. The label "Overall" means that the attacker is trained with randomized dynamics, network structures, action space, reward space and alternative victim policy algorithms. The label "NAF" means that the attacker is trained on alternative victim policy trained with NAF, with non-randomized dynamics, network structures, action space, reward space. The label "DDPG" means that the attacker is trained on alternative victim policy trained with DDPG, with non-randomized dynamics, network structures, action space, reward space. The label "DDPG" means that the attacker is trained on alternative victim policy trained with SAC, with non-randomized dynamics, network structures, action space, reward space. In each subfigure, the attackers are evaluated on all victim policies (of all dynamics, network structures, action space, and reward space) trained with 'NAF', 'DDPG', 'SAC', respectively. The results show that training with everything randomized may not always improve the attack performance. The reason might be that: first, the transferability among different reward spaces is not very stable; second, the transferability among different policy algorithms is not very stable. Given the instability of attack transferability among reward space and policy algorithm, the overall randomization effects may be dampened. However, as we already observe in the results on randomized dynamics, network structures and action spaces, the transferability among these domains are strong and stable, and randomization does help to improve the attack effects. Therefore, these results provide guidance and referenes for designing robust policies with strong defense against such attacks.

Finally, we present some results on comparing the proposed three attack algorithms (local minimization, global minimization and the combined algorithm) in Figure 4. The results show that combining the local minimization and global minimization method does sometimes provide an edge on achieving better attack effects.

5. Training Details

5.1. Network Structure

We use the original MuJoCo state space definition instead of images to train the alternative victim policies. The input to these networks will be a vector, and the dimension of the vector depends on the observation space of each individual task. We use fully connected layers for training these policies. For all networks, we use FC(s_{in}, s_{out}) to represent one fully connected layer, where s_{in} indicates the input size, and s_{out} indicates the output size. When there are varying number of layers we use $\{\text{FC}(s_{in}, s_{out})\}^n$ to represent that there are n layers of fully connected layers. We use a dash line to connect these layers.

DDPG Network Structure For DDPG, we train an actor and a critic network. The actor takes in the observation as input and outputs the action choice. The network structure is consisted of multiple fully connected layers with layer normalization layer in-between and with a hidden dimension of 128. For the number of layers of n, state input size as s, and action output size as a, the structure can be expressed as this: FC(s, 128)-LayerNorm-ReLU-$\{$FC(128, 128)-LayerNorm-ReLU-$\}^{n-2}$-FC(128, a). Here LayerNorm stands for layer normalization layer, and ReLU stands for the ReLU nonlinear activation layer. The same for following structures. The critic structure can be expressed as this: $\{$FC(s, 128)-LayerNorm-ReLU-$\}^{n-1}$ for processing state and FC(128+a, 128)-LayerNorm-ReLU-$\{$FC(128, 128)-LayerNorm-ReLU)-$\}^{n-2}$-FC(128, 1) for processing the hidden state and action

concatenated vectors.

NAF Network Structure For NAF, we train the policy and value function with a shared state perception network. The network takes in the state and processes the state with n layers of fully connected layers where $n = 2, 3, 4$. The state processing network structure can be expressed as BN-FC(s, 128)-Tanh-{FC(128, 128)-Tanh-BN-}$^{n-1}$. Then the value function is a single layer fully connected network FC(128, 1). The actor takes in the processed state and processes with a single fully connected layer FC(128, a). For more detailed network structure please refer to the code base for information.

SAC Network Structure For SAC, we train a Gaussian stochastic policy function and a value function. The value function takes in the state and action pair and process them with $n = 2, 3, 4$ fully connected layers followed by a final fully connected layer to output the one dimensional value. We use a hidden dimension of 256 for all intermediate layers. We add ReLU [2] non-linear activation function after each fully connected layer. We use a double value function structure. For the policy network, we use a $n = 2, 3, 4$ layer fully connected network with hidden dimension of 256 to process the state and a final fully connected layer to get the action output. For detailed information, please refer to the code base.

PPO Network Structure For PPO, we train an actor network and a critic network. The actor network is consisted of $n = 2, 3, 4$ state processing fully connected layers with each layer followed by a Tanh function. The hidden size for intermediate layer is 64. Then the actor network is followed by a final output fully connected layer. The critic network is consisted of $n = 2, 3, 4$ state processing fully connected layers with each layer followed by a Tanh function. The hidden size for intermediate layer is 64. Then the critic network is followed by a final output fully connected layer to output the value. For detailed information, please refer to the code base.

Attack Network Structure For training the attacker, we use a feed forward fully connected network with a structure as this: FC(s, 256)-LayerNorm-ReLU-FC(256, 256)-LayerNorm-ReLU-FC(256, s)-Tanh. The output is timed with the $L - \infty$ bound ϵ to bound the perturbation strength.

5.2. Training the Alternative Victim Policies

Please refer to the attached code base for information about training the alternative victim policies. The code base contains the following bash script folders: "scripts" that contains the code for training policies; "attack_scripts" that contains the code for training the attackers; "evaluate_scripts" that contains the code for evaluating the attackers. The code should be self-contained and it take a few days to finish training the alternative victim policies and the attack networks. Evaluating the trained attacker takes a few hours to complete. We evaluate the trained attackers for 100 episodes to get the mean and standard deviation of the episodic reward after the attack. We provide additional details on randomization on reward and action spaces in each environment.

Randomization on Reward Space We include here the randomization we designed on the reward space for both environments. For the HalfCheetah environment, given the previous state as $\mathbf{s}$ and the state after taking the action $\mathbf{a}$ as $\mathbf{s}'$, the time of executing the action as t, the reward ID as $i = 0, 1, 2, 3, 4$, the reward is defined as: first, if reward ID is 0, the reward is defined as,

$$r_0 = -0.1 * \|\mathbf{a}\|^2 + (\mathbf{s}'_0 - \mathbf{s}_0)/t, \tag{1}$$

where $\mathbf{s}_0$ means the first dimension of the vector $\mathbf{s}$; second, if the reward ID is i =1,2,3 or 4, the reward is defined as,

$$r = \frac{\mathbf{s}'_0 - \mathbf{s}_0}{t}(3-0.5i)+(0.5i+0.1)-0.0005*\|\mathbf{a}\|^2(i+0.5). \tag{2}$$

For the Walker-Walk environment, the reward for different ID $i = 0, 1, 2, 3, 4$ corresponds to different moving speed. The correspondence is as follows: $\{0 : 1.5, 1 : 0, 2 : 4, 3 : 8, 4 : 10\}$.

Randomization on Action Space We include here the details of randomization we designed on the action space for both environments. Define the original action space as $\mathcal{A}$, and the example original action as $\mathbf{a} \in \mathbb{R}^d$, where d is the default dimension of the robot's action space. Given the action space ID $i = 0, 1, 2, 3, 4$, the new dimension of the action space is $\{0 : d, 1 : d-2, 2 : d-1, 3 : d+2, 4 : d+3\}$. For the case where the dimension of action space is smaller than that of the default action space, the missing dimensions will be filled with zero. For the case where the dimension of action space is larger than that of the default action space, the extra dimensions will be removed when executing the actions.

References

[1] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *International Conference on Learning Representations*, 2017. 2

[2] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 3

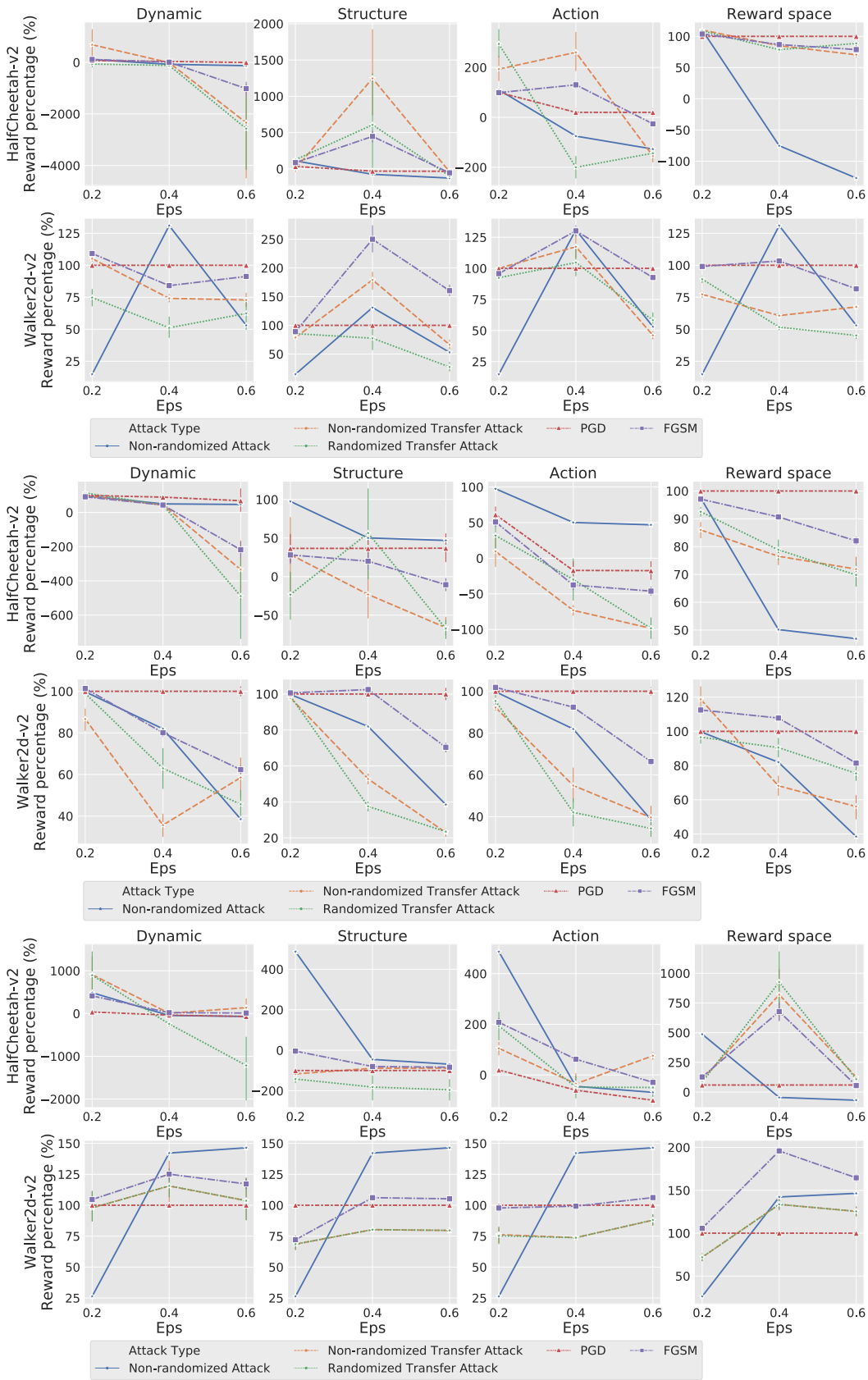

Figure 1. Domain randomization in Dynamic, network structure, action space and reward space. Shown are the results for attacking victim policies trained with DDPG(top), SAC(middle), and PPO(bottom).

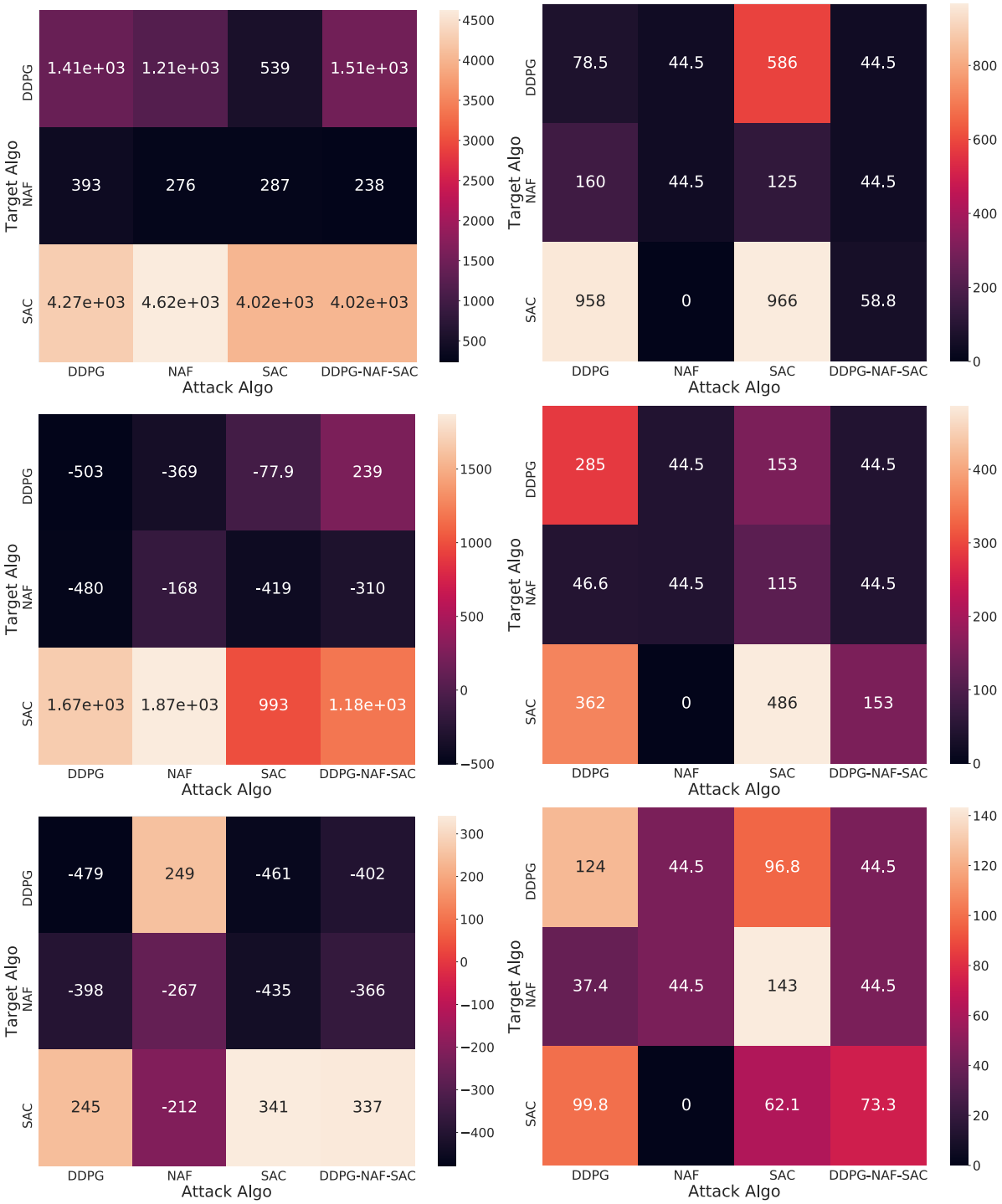

Figure 2. Domain randomization in target algorithms. Shown here are the attack results on HalfCheetah with eps of 0.2 (top left), Walker with eps 0.2 (top right), HalfCheetah with eps of 0.4 (middle left), Walker with eps 0.4 (middle right), HalfCheetah with eps of 0.6 (bottom left) and Walker with eps 0.6 (bottom right).

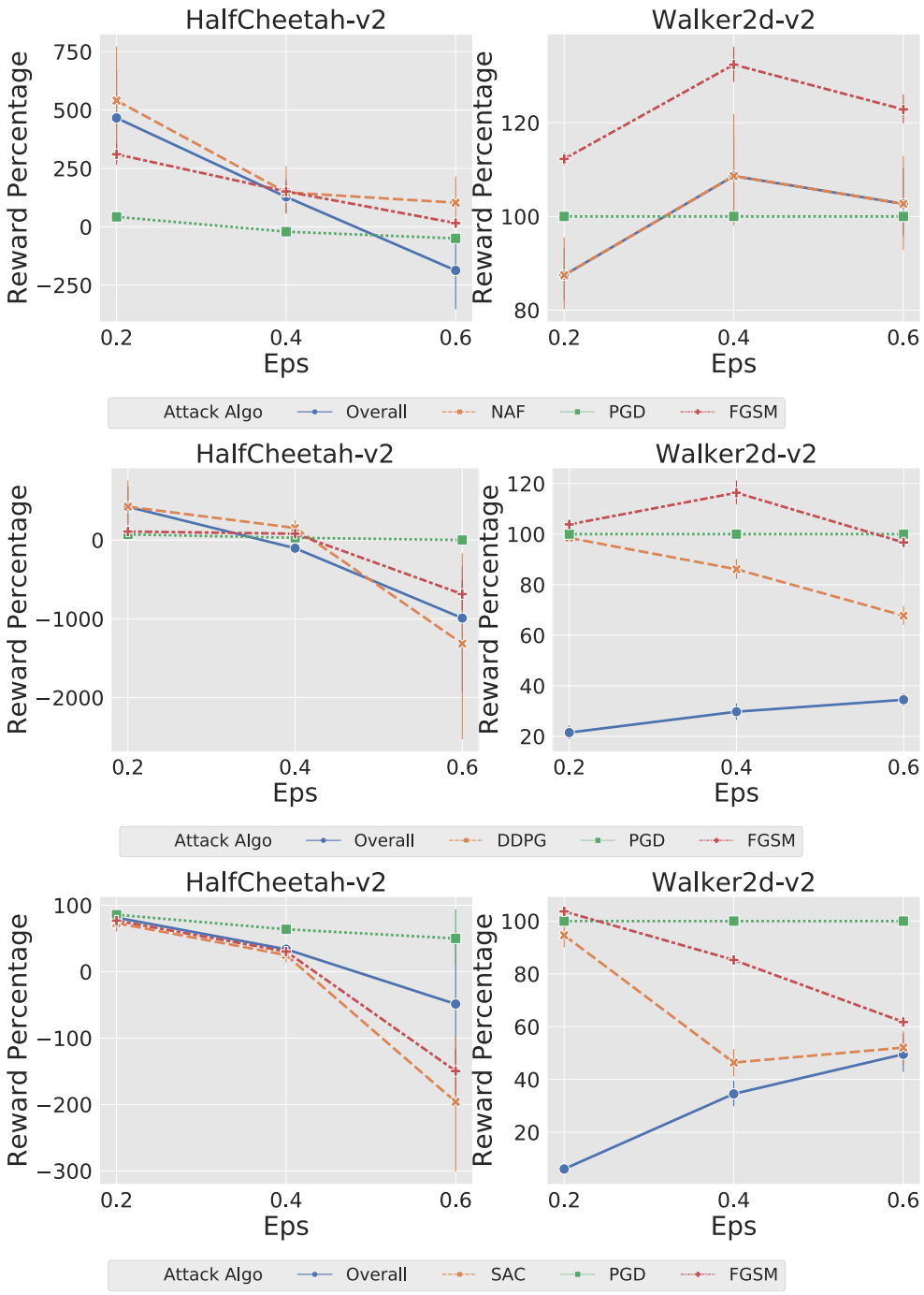

Figure 3. Domain randomization in all domains. Shown are the results for attacking victim policies trained with NAF (top), DDPG (middle) and SAC (bottom).

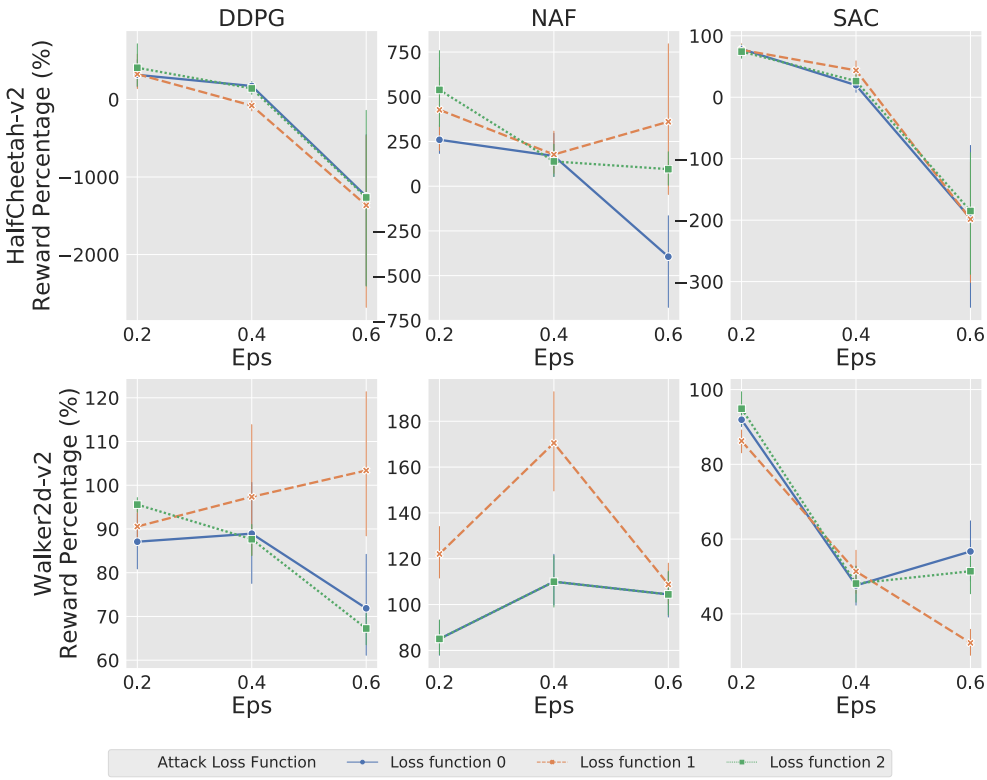

Figure 4. Results on different attack algorithms. There is no randomization in this case, so the attacker is trained on the victim policy directly to minimize the reward. Shown are the results for HalfCheetah (first row) and Walker (second row).