ADVERSARIAL ATTACKS ON NEURAL NETWORK POLICIES

Sandy Huang[†], **Nicolas Papernot**[‡], **Ian Goodfellow**[§], **Yan Duan**[†§], **Pieter Abbeel**[†§]
[†] University of California, Berkeley, Department of Electrical Engineering and Computer Sciences
[‡] Pennsylvania State University, School of Electrical Engineering and Computer Science
[§] OpenAI
shhuang@cs.berkeley.edu, ngp5056@cse.psu.edu,
{ian,rocky,pieter}@openai.com

ABSTRACT

Machine learning classifiers are known to be vulnerable to inputs maliciously constructed by adversaries to force misclassification. Such adversarial examples have been extensively studied in the context of computer vision applications. In this work, we show adversarial attacks are also effective when targeting neural network policies in reinforcement learning. Specifically, we show existing adversarial example crafting techniques can be used to significantly degrade test-time performance of trained policies. Our threat model considers adversaries capable of introducing small perturbations to the raw input of the policy. We characterize the degree of vulnerability across tasks and training algorithms, for a subclass of adversarial-example attacks in white-box and black-box settings. Regardless of the learned task or training algorithm, we observe a significant drop in performance, even with small adversarial perturbations that do not interfere with human perception. Videos are available at http://rll.berkeley.edu/adversarial.

1 INTRODUCTION

Recent advances in deep learning and deep reinforcement learning (RL) have made it possible to learn end-to-end policies that map directly from raw inputs (e.g., images) to a distribution over actions to take. These policies are parametrized by neural networks, which have been shown to be vulnerable to adversarial attacks in supervised learning settings (Szegedy et al., 2014). Unlike supervised learning, where a fixed dataset of training examples is processed during learning, in reinforcement learning these examples are gathered throughout the training process. Thus, policies trained to do the same task could conceivably be significantly different (e.g., in terms of the high-level features they extract from the raw input), depending on how they were initialized and trained.

Our main contribution is to characterize how the effectiveness of adversarial attacks on neural network policies is impacted by two factors: the deep RL algorithm used to learn the policy, and whether the adversary has access to the policy network itself (white-box vs. black-box). We consider a fully trained policy at test time, and allow the adversary to make limited changes to the raw input perceived from the environment before it is passed to the policy (Fig. 1).

2 RELATED WORK

Szegedy et al. (2014) first demonstrated the vulnerability of deep networks to perturbations indistinguishable to humans, leading to a series of follow-up work that showed perturbations could be produced with minimal computing resources (Goodfellow et al., 2015) and/or with access to the model label predictions only (thus enabling black-box attacks) (Papernot et al., 2016), and that these perturbations can also be applied to physical objects (Kurakin et al., 2016; Sharif et al., 2016).

Most work on adversarial examples so far has studied their effect on supervised learning algorithms. A recent technical report studied the scenario of an adversary interfering with the training of an agent, with the intent of preventing the agent from learning anything meaningful (Behzadan & Munir, 2017). Our work is the first to study the ability of an adversary to interfere with the operation of an RL agent by presenting adversarial examples at test time.

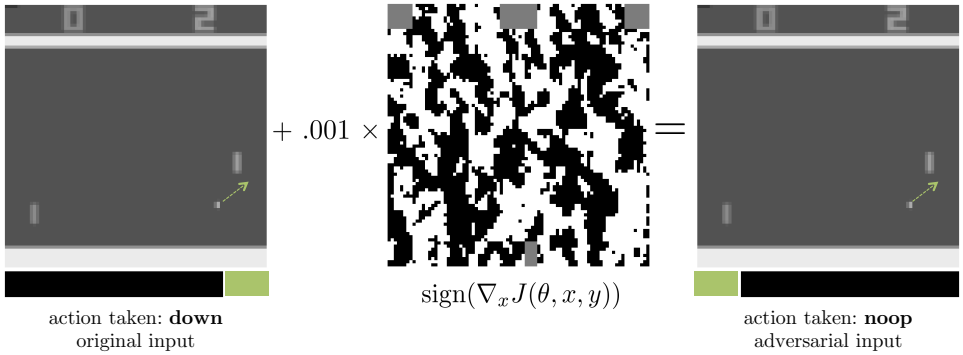

Figure 1: Fooling a policy trained with DQN (Mnih et al., 2013) to play Pong. The policy chooses a good action given the original input, but the adversarial perturbation results in missing the ball and losing the point. (The dotted arrow starts from the ball and denotes the direction it is traveling in, and the green rectangle highlights the action that maximizes the Q-value, for the given input.)

3 Adversarial Attacks

We use the Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2015), an existing method for efficiently generating adversarial examples in the context of computer vision classification. FGSM is fast because it makes a linear approximation of a deep model and solves the maximization problem analytically, to compute the optimal adversarial perturbation under the linear approximation.

3.1 Applying FGSM to Policies

FGSM requires calculating $\nabla_x J(\theta, x, y)$, the gradient of the cost function $J(\theta, x, y)$ with respect to the input x. In the reinforcement learning setting, we assume the output y is a weighting over possible actions (i.e., the policy is stochastic). When computing adversarial perturbations with FGSM for a trained policy π_θ, we assume the action with the maximum weight in y is the optimal action to take: in other words, we assume the policy performs well at the task. Thus, $J(\theta, x, y)$ is the cross-entropy loss between y and the distribution that places all weight on the highest-weighted action in y.[1]

3.2 Choosing a Norm Constraint

FGSM typically restricts the ℓ_∞-norm of the adversarial perturbation η. We additionally consider restriction of the ℓ_1- and ℓ_2-norms, since in certain situations it may be desirable to change all input features by no more than a tiny amount (i.e., constrain $\|\eta\|_\infty$), whereas in others it may be better to change only a small number of input features (i.e., constrain $\|\eta\|_1$). Linearizing the cost function $J(\theta, x, y)$ around the current input x, the optimal perturbation for each type of norm constraint is:

$$\eta = \begin{cases} \epsilon \operatorname{sign}(\nabla_x J(\theta, x, y)) & \text{for constraint } \|\eta\|_\infty \leq \epsilon \\ \epsilon\sqrt{d} * \frac{\nabla_x J(\theta,x,y)}{\|\nabla_x J(\theta,x,y)\|_2} & \text{for constraint } \|\eta\|_2 \leq \|\epsilon \mathbf{1}_d\|_2 \\ \text{maximally perturb highest-impact dimensions with budget } \epsilon d & \\ & \text{for constraint } \|\eta\|_1 \leq \|\epsilon \mathbf{1}_d\|_1 \end{cases} \tag{1}$$

where d is the number of dimensions of input x. Note that the ℓ_2-norm and ℓ_1-norm constraints have ϵ adjusted to be the ℓ_2- and ℓ_1-norm of the vector $\epsilon \mathbf{1}_d$, respectively, since that is the amount of perturbation under the ℓ_∞-norm constraint. The optimal perturbation for the ℓ_1-norm constraint either maximizes or minimizes the feature value at dimensions i of the input, ordered by decreasing $|\nabla_\theta J(\theta, x, y)_i|$. For this norm, the adversary's budget — the total amount of perturbation the adversary is allowed to introduce in the input — is ϵd.

4 Experimental Evaluation

We evaluate our adversarial attacks on four Atari 2600 games in the Arcade Learning Environment (Bellemare et al., 2013) — Chopper Command, Pong, Seaquest, and Space Invaders — and

[1] Functionally, this is equivalent to a technique introduced in the context of image classification, to generate adversarial examples without access to the true class label (Kurakin et al., 2017).

train each with three deep reinforcement learning algorithms: A3C (Mnih et al., 2016), TRPO (Schulman et al., 2015), and DQN (Mnih et al., 2013). We use the same pre-processing and neural network architecture as in Mnih et al. (2013) (Appendix B). The input is a concatenation of the last 4 images, converted from RGB to luminance (Y) and resized to 84×84. Luminance values are rescaled to be from 0 to 1. The output of the policy is a distribution over possible actions.

For each game and training algorithm, we train five policies starting from different random initializations. For our experiments, we focus on the top-performing trained policies, which we define as all policies that perform within 80% of the maximum score, averaged over the last ten training iterations. We cap the number of policies at three for each game and training algorithm. Certain combinations (e.g., Seaquest with A3C) had only one policy meet these requirements.

4.1 Vulnerability to White-Box Attacks

First, we are interested in how vulnerable neural network policies are to white-box adversarial-example attacks, and how this is affected by the type of adversarial perturbation and training algorithm. We find regardless of which game the policy is trained for or how it is trained, it is indeed possible to significantly decrease performance through introducing relatively small perturbations (Supplementary Fig. S2).

Notably, in many cases an ℓ_∞-norm FGSM adversary with $\epsilon = 0.001$ decreases the agent's performance by 50% or more; when converted to 8-bit image encodings, these adversarial inputs are indistinguishable from the original inputs. In contrast, ℓ_1-norm adversaries are able to sharply decrease the agent's performance just by changing a few pixels (by large amounts).

We see that policies trained with A3C, TRPO, and DQN are all susceptible to adversarial inputs. Interestingly, policies trained with DQN are more susceptible, especially to ℓ_∞-norm FGSM perturbations on Pong, Seaquest, and Space Invaders.

4.2 Vulnerability to Black-Box Attacks

In practice, often an adversary does not have complete access to the neural network of the target policy (Papernot et al., 2016), known as the black-box scenario. We investigate how vulnerable neural network policies are to black-box attacks of the following two variants:

1. Transferability across policies: the adversary has access to the training environment and knowledge of the training algorithm and hyperparameters. It knows the architecture of the target policy network, but not its random initialization.
2. Transferability across algorithms: the adversary has no knowledge of the training algorithm.

As one might expect, we find that the less the adversary knows about the target policy, the less effective the adversarial examples are (Supplementary Fig. S3, S4, S5). Transferability across algorithms is less effective at decreasing agent performance than transferability across policies, which is less effective than when the adversary does not need to rely on transferability (i.e., the adversary has full access to the target policy network). However, for most games, transferability across algorithms is still able to significantly decrease the agent's performance, especially for larger values of ϵ. Notably for $\ell 1$-norm adversaries, transferability across algorithms is nearly as effective as no transferability, for most game and algorithm combinations.

5 Discussion and Future Work

This direction of work has significant implications for both online and real-world deployment of neural network policies. Our experiments show it is fairly easy to confuse such policies with computationally-efficient adversarial examples, even in black-box scenarios. Based on Kurakin et al. (2016), these adversarial perturbations could possibly be applied to objects in the real world, for example adding strategically-placed paint to road surfaces to confuse an autonomous car's lane-following policy.

Thus, an important direction of future work is developing defenses against adversarial attacks. This could involve adding adversarially-perturbed examples during training time (as in Goodfellow et al. (2015)), or it could involve detecting adversarial input at test time and dealing with it appropriately.

REFERENCES

V. Behzadan and A. Munir. Vulnerability of deep reinforcement learning to policy induction attacks. *arXiv preprint arXiv:1701.04143*, 2017.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the Thirty-Third International Conference on Machine Learning*, 2016.

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

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *Proceedings of the Fifth International Conference on Learning Representations*, 2017.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Workshop on Deep Learning*, 2013.

V. Mnih, A. Puigdomenech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the Thirty-Third International Conference on Machine Learning*, 2016.

N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*, 2016.

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. In *Proceedings of the Thirty-Second International Conference on Machine Learning*, 2015.

Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1528–1540, 2016.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *Proceedings of the Second International Conference on Learning Representations*, 2014.

A SUPPLEMENTARY FIGURES

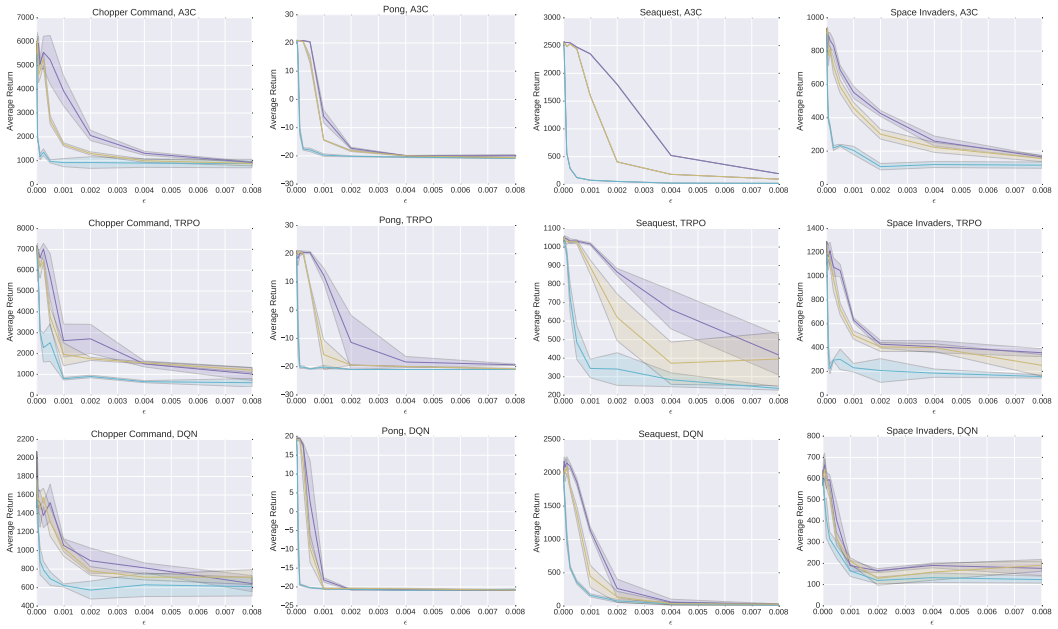

Figure S2: Comparison of the effectiveness of ℓ_∞, ℓ_2, and ℓ_1 FGSM adversaries on four Atari games trained with three learning algorithms. The average return is taken across ten rollouts. Constraint on FGSM perturbation: ℓ_∞-norm ℓ_2-norm ℓ_1-norm

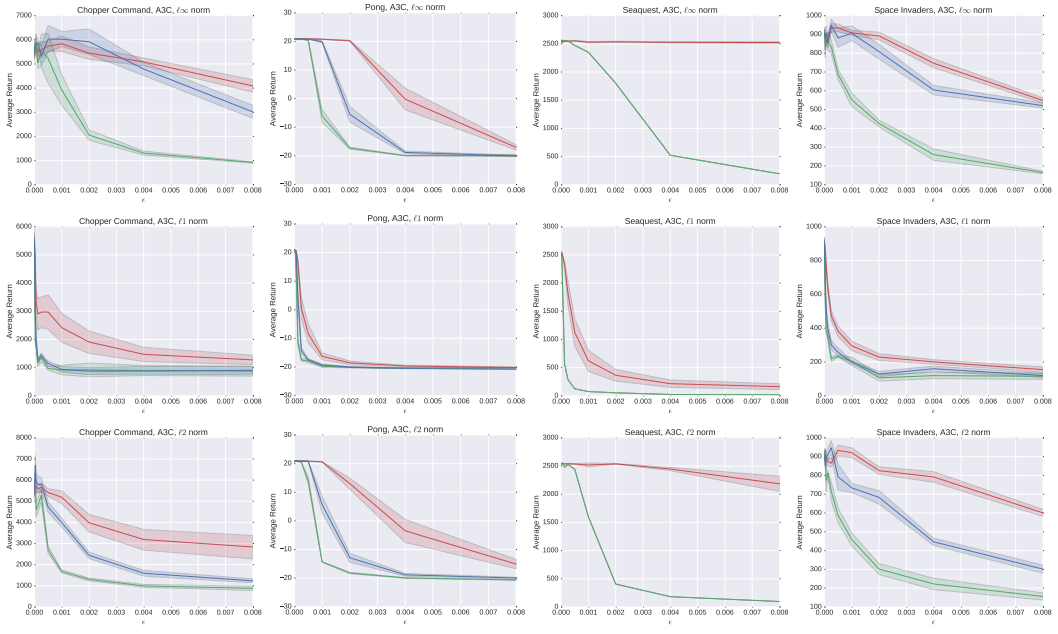

Figure S3: Transferability of adversarial inputs for policies trained with A3C. Type of transfer: algorithm policy none

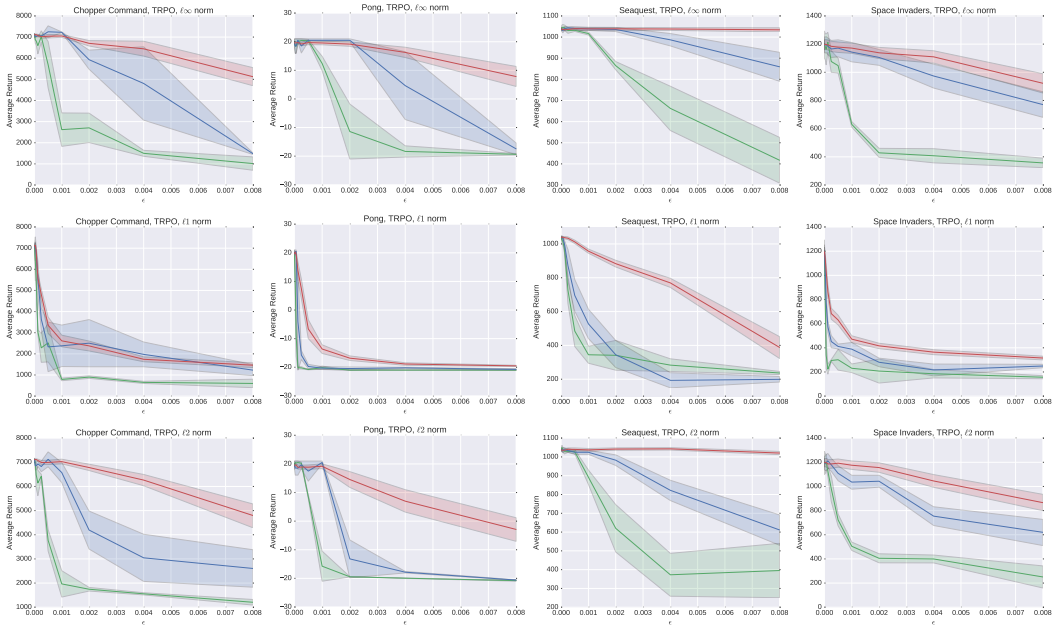

Figure S4: Transferability of adversarial inputs for policies trained with TRPO. Type of transfer: algorithm policy none

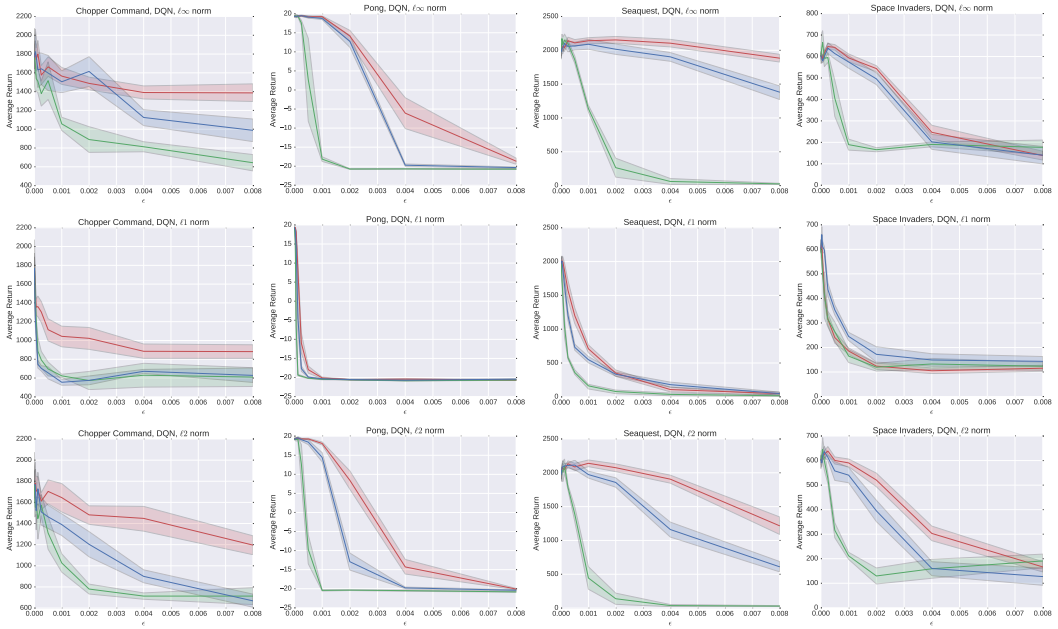

Figure S5: Transferability of adversarial inputs for policies trained with DQN. Type of transfer: algorithm policy none

B EXPERIMENTAL SETUP

We set up our experiments within the rllab (Duan et al., 2016) framework. We use a parallelized version of the rllab implementation of TRPO, and integrate outside implementations of DQN[2] and A3C[3]. We use OpenAI Gym environments (Brockman et al., 2016) as the interface to the Arcade Learning Environment (Bellemare et al., 2013).

The policies use the network architecture from Mnih et al. (2013): a convolutional layer with 16 filters of size 8×8 with a stride of 4, followed by a convolutional layer with 32 filters of size 4×4 with a stride of 2. The last layer is a fully-connected layer with 256 hidden units. All hidden layers are followed by a rectified nonlinearity.

For all games, we set the frame skip to 4 as in Mnih et al. (2013). The frame skip specifies the number of times the agent's chosen action is repeated.

B.1 TRAINING

We trained policies with TRPO and A3C on Amazon EC2 c4.8xlarge machines. For each policy, we ran TRPO for 2,000 iterations of 100,000 steps each, which took 1.5 to 2 days. We set the bound on the KL divergence to 0.01, as in Schulman et al. (2015).

For A3C, we used 18 actor-learner threads and a learning rate of 0.0004. As in Mnih et al. (2016), we use an entropy regularization weight of 0.01, use RMSProp for optimization with a decay factor of 0.99, update the policy and value networks every 5 time steps, and share all weights except the output layer between the policy and value networks. For each policy, we ran A3C for 200 iterations of 1,000,000 steps each, which took 1.5 to 2 days.

For DQN, we trained policies on Amazon EC2 p2.xlarge machines. We used 100,000 steps per epoch and trained for two days.

[2] `github.com/spragunr/deep_q_rl`

[3] `github.com/muupan/async-rl`