# Summary of Three Recent Papers: Deep Reinforcement Learning and Adversarial Attacks

Presentor: Ji Gao

Department of Computer Science, University of Virginia

https://qdata.github.io/deep2Read/

# Paper List

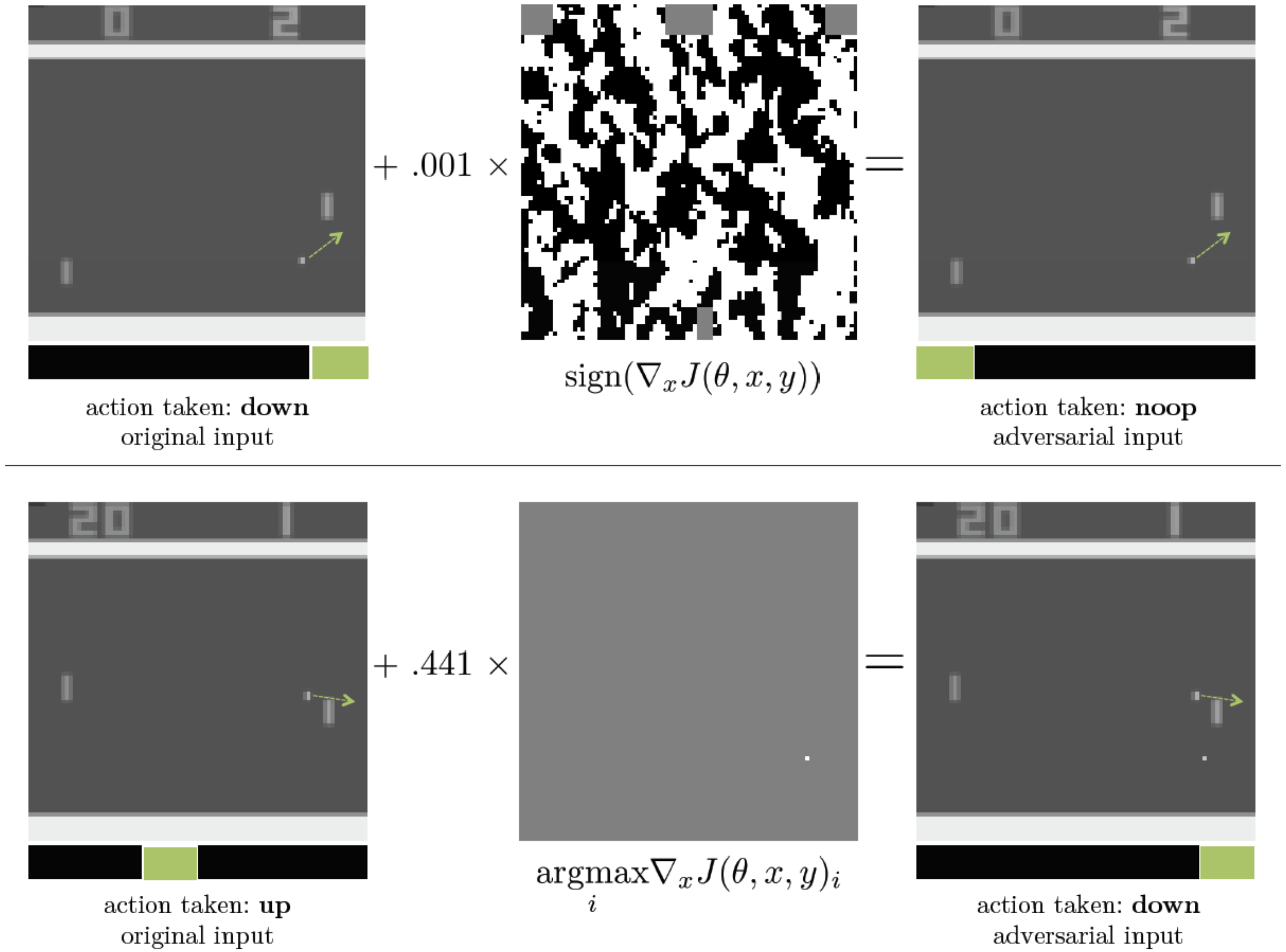
- Adversarial Attacks on Neural Network Policies
- Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks
- Online Robust Policy Learning in the Presence of Unknown Adversaries
- Robust Deep Reinforcement Learning with Adversarial Attacks

# Adversarial Attacks on Neural Network Policies

*Sandy Huangy, Nicolas Papernotz, Ian Goodfellowx, Yan Duanyx, Pieter Abbeely*

- 2017 ICLR Workshop
- Idea: Craft adversarial samples in the input feature (State) of RL algorithm, lead to a large degrade of test-time performance
- Test-time

# Idea



+ .001 ×
=
$\mathrm{sign}(\nabla_x J(\theta, x, y))$
action taken: **down**
original input
action taken: **noop**
adversarial input
+ .441 ×
=
$\underset{i}{\mathrm{argmax}} \nabla_x J(\theta, x, y)_i$
action taken: **up**
original input
action taken: **down**
adversarial input

# RL algorithm

- DQN: Deep Q Network
- TRPO: Trust Region Policy Optimization. Use a whole trajectory rollout on stochastic policy, penalized by the KL divergence between old and new policy.
- A3C: Asynchronous  Advantage Actor-Critic. Asynchronous gradient descent on stochastic policy.

# Apply FGSM on Policy

- FGSM directly use $J(\theta, x, y)$ to generate perturbation.
- In this case, just assume the policy $\pi_\theta$ generated is good, and our good is to flip the predicted policy.
- Note: Use an extra softmax on the DQN

# Different norms

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

# Result

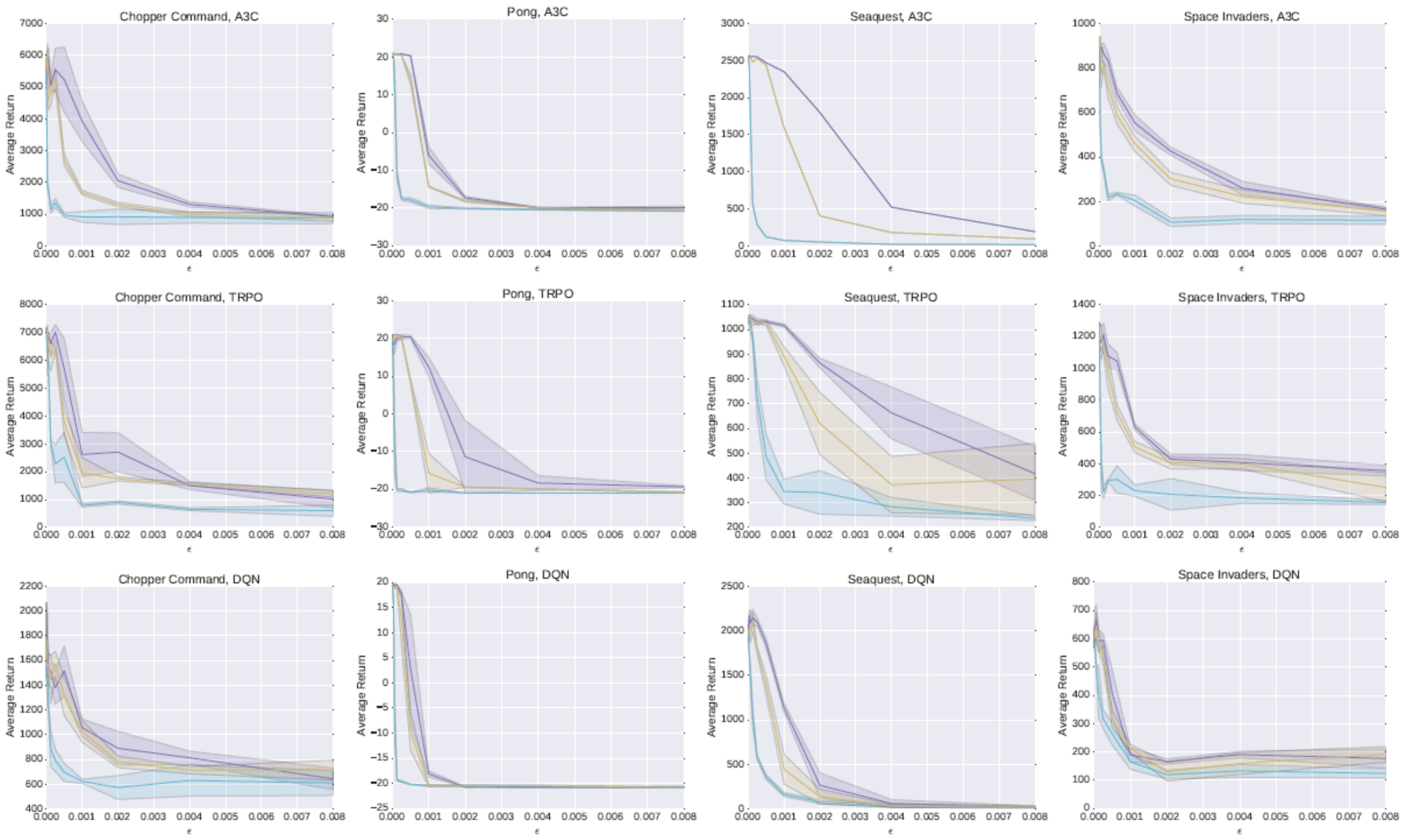



Figure 2: Comparison of the effectiveness of $\ell_\infty$, $\ell_2$, and $\ell_1$ FGSM adversaries on four Atari games trained with three learning algorithms. The average return is taken across ten trajectories. Constraint on FGSM perturbation: $\ell_\infty$-norm, $\ell_2$-norm, $\ell_1$-norm

# Transferability

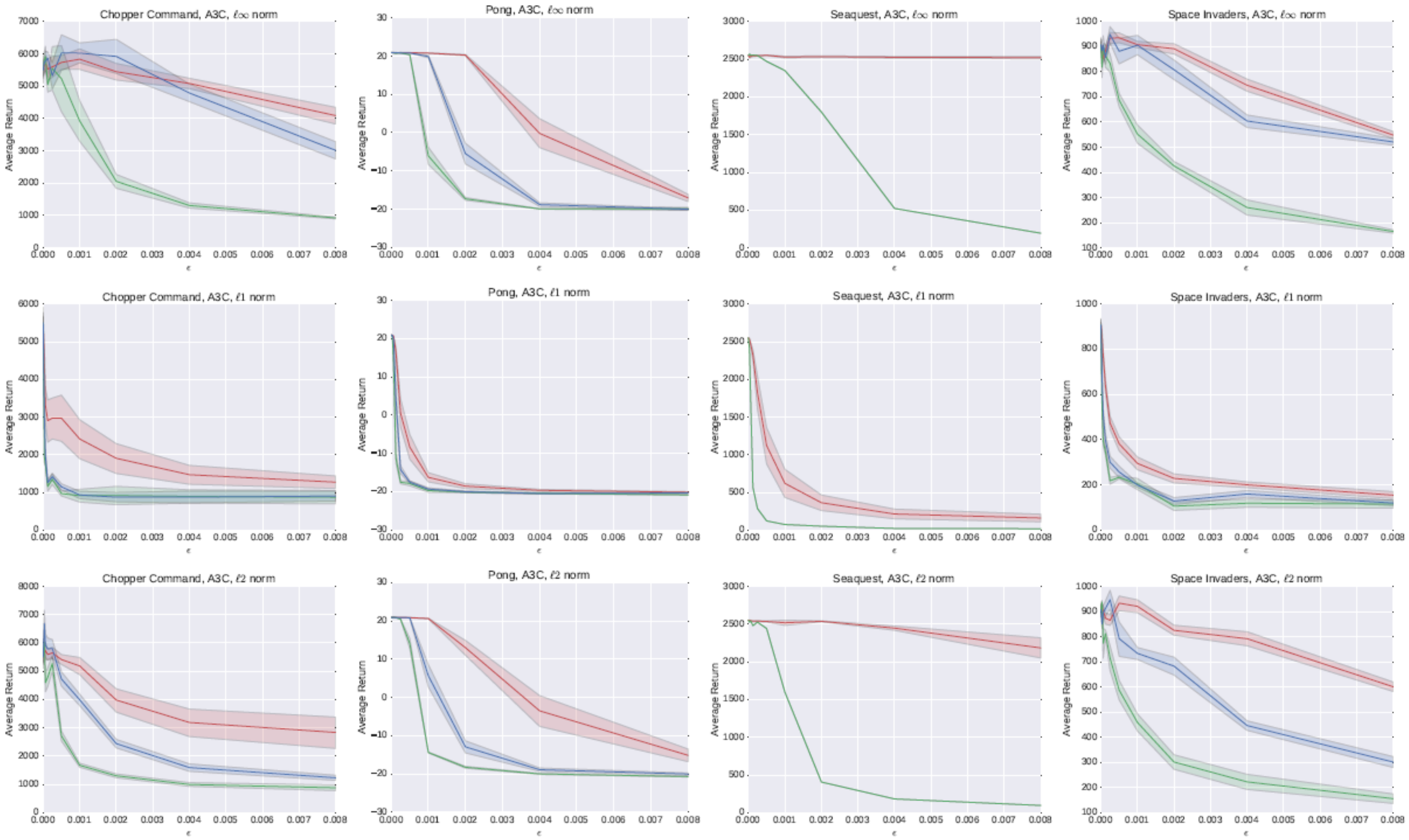
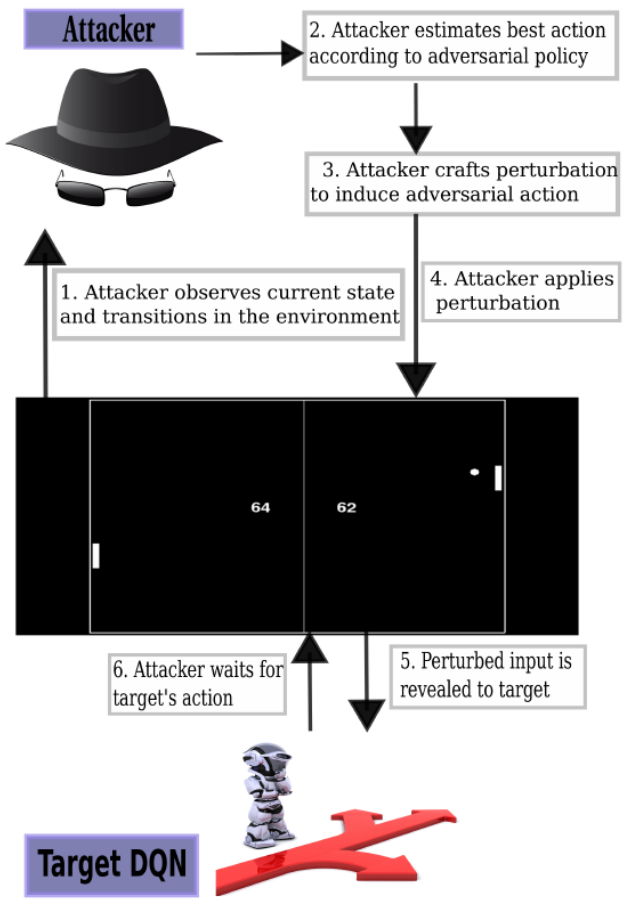



Figure 3: Transferability of adversarial inputs for policies trained with A3C. Type of transfer: algorithm policy none

# Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks

*Vahid Behzadan and Arslan Munir*
*International Conference on Machine Learning and Data Mining in Pattern Recognition* 2017

- Attack DQN



# Threat model

- Priori info: Structure of DQN, reward function R
- Attacker: Work on state, not actions
- Magnitude of perturbation is smaller than $\epsilon$

# Method

- Use another DQN to simulate DQN
- Use FGSM and JSMA

# Procedure

**Algorithm 1:** Exploitation Procedure

**input** : adversarial policy $\pi^*_{adv}$, initialized replica DQNs $Q'$, $\hat{Q}'$, synchronization frequency $c$, number of iterations N

**for** *observation = 1, N* **do**
- Observe current state $s_t$, action $a_t$, reward $r_t$, and resulting state $s_{t+1}$
- **if** $s_{t+1}$ *is not terminal* **then**
  - set $a'_{adv} = \pi^*_{adv}(s_{t+1})$
  - Calculate perturbation vector $\hat{\delta}_{t+1} = Craft(\hat{Q}', a'_{adv}, s_{t+1})$
  - Update $s_{t+1} \leftarrow s_{t+1} + \hat{\delta}_{t+1}$
  - Set $y_t = (r_t + max_{a'} \hat{Q}'(s_{t+1} + \hat{\delta}_{t+1}, a'; \theta'\_)$
  - Perform SGD on $(y_t - Q'(s_t, a_t, \theta'))^2$ w.r.t $\theta'$
- **end**
- Reveal $s_{t+1}$ to target
- **if** *observation* mod $c = 0$ **then** $\theta'\_ \leftarrow \theta'$

**end**

# Experiment result

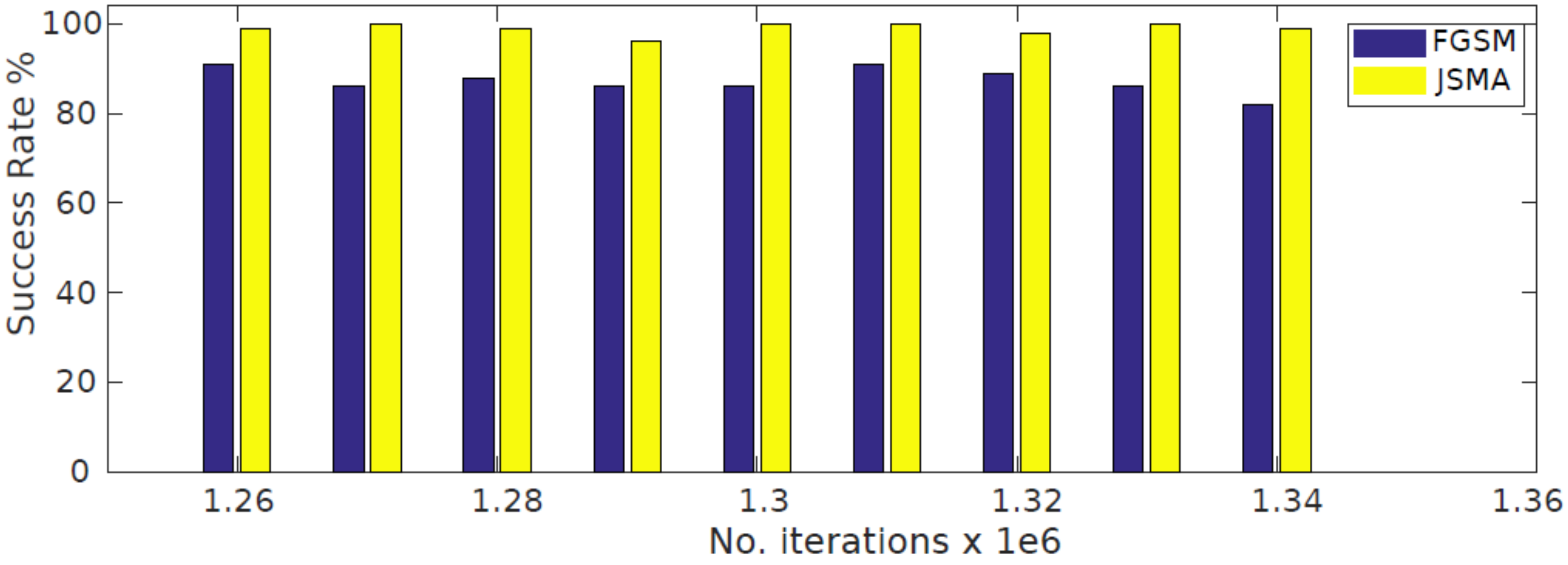
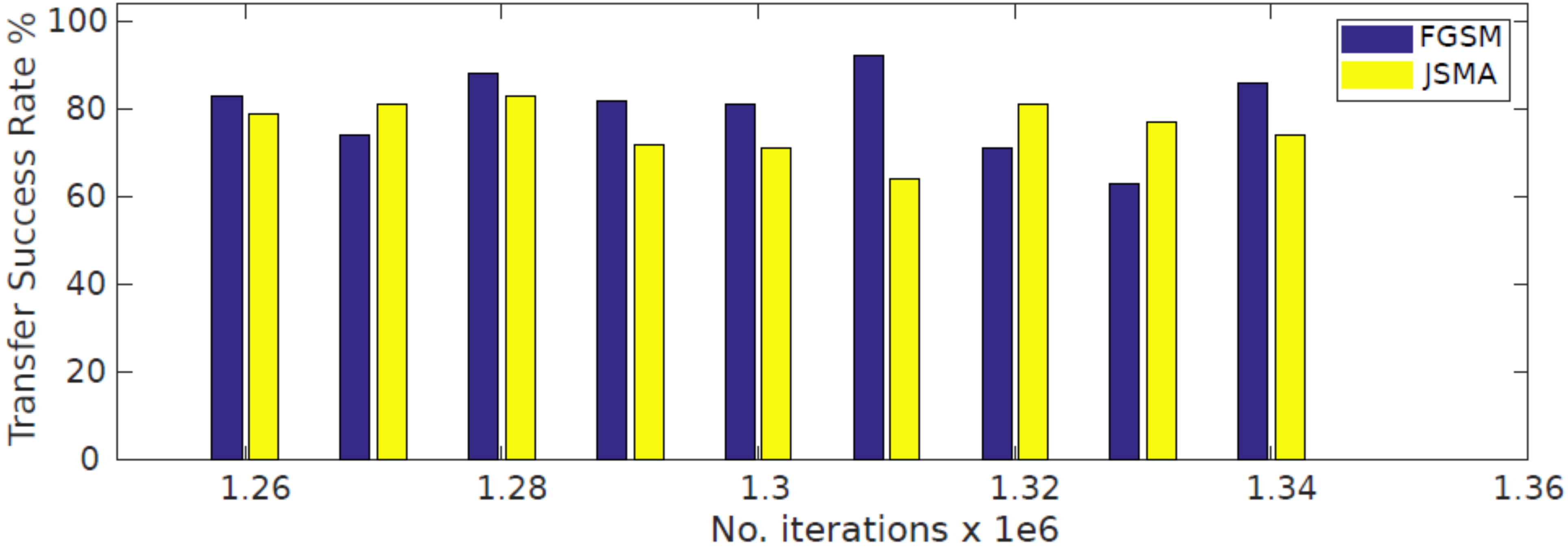



Fig. 4: Success rate of crafting adversarial examples for DQN

# Transferability



# Online Robust Policy Learning in the Presence of Unknown Adversaries

Aaron J. Havens, Zhanhong Jiang, Soumik Sarkar

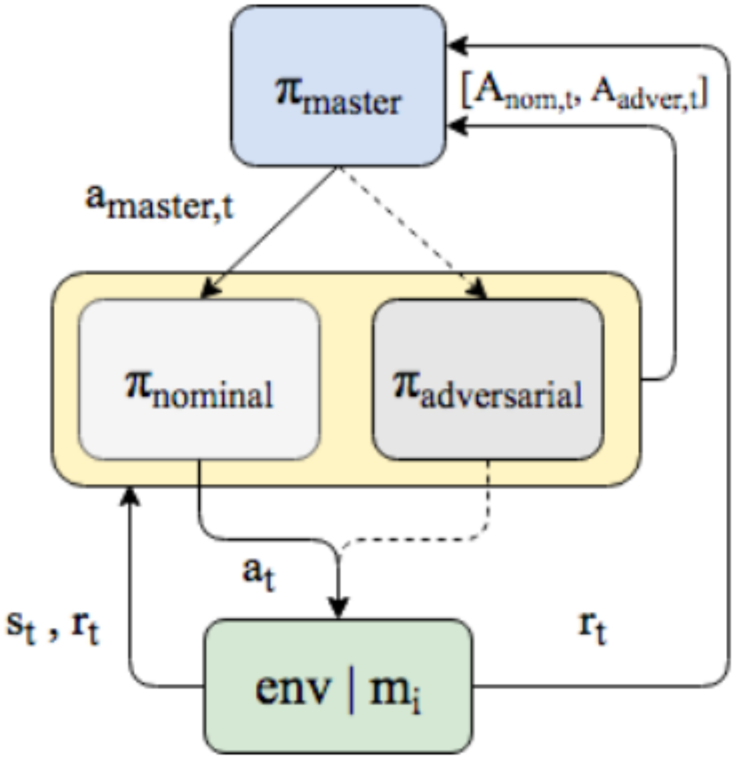
- Setting: Finite-horizon discounted MDP
- Base Algorithm: TRPO
- Goal: Online mitigation of the adversarial perturbation

# Motivation

- In the learning, if the input changed, the expected reward will be different to the actual reward.
- In the online setting, an advantage(expect result – observed return) can be used as a good indicator that whether the input has been perturbed

# Method(General idea)

- Create multiple(2) policies
- A master policy can choose one policy from time to time.
- When it detects a change on the input, it will choose a different policy




(b) MLAH framework

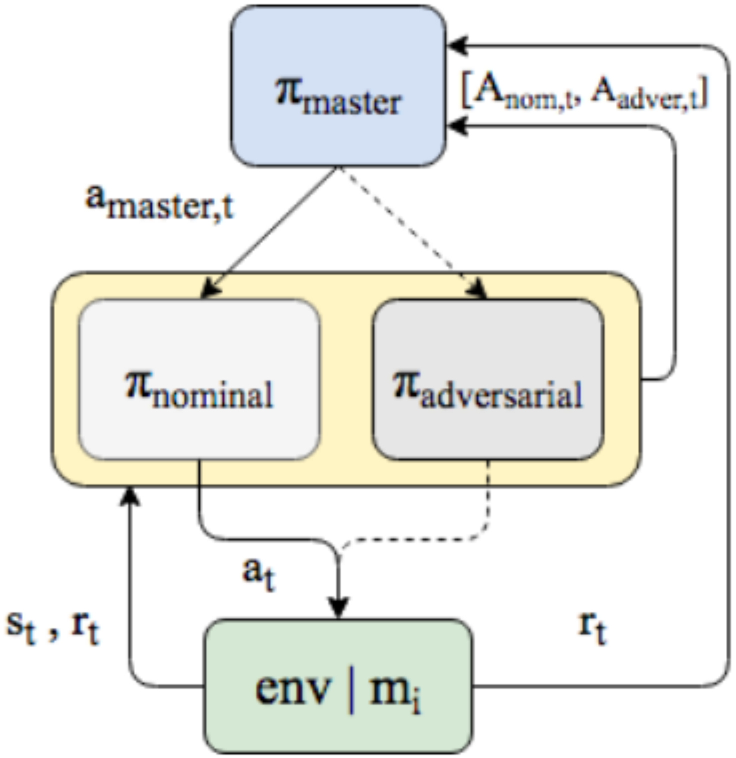
# Method

- Start with random policies
- For a step, estimate the advantage of both policies

$$\mathbf{A}_t = [A_{GAE,t-h}|\pi_{nom}, A_{GAE,t-h}|\pi_{adv}] \in \mathbb{R}^2$$

$$a_{master,t} = \pi_{*,t} = \underset{a}{\mathrm{argmax}}\, \mathbb{E}_{s_t,\pi_i,m_i\ldots}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a) | \, m_i\right] \in \{\pi_{nom}, \pi_{adv}\}$$

- Optimize master policy according to the loss
- Optimize both policies




(b) MLAH framework

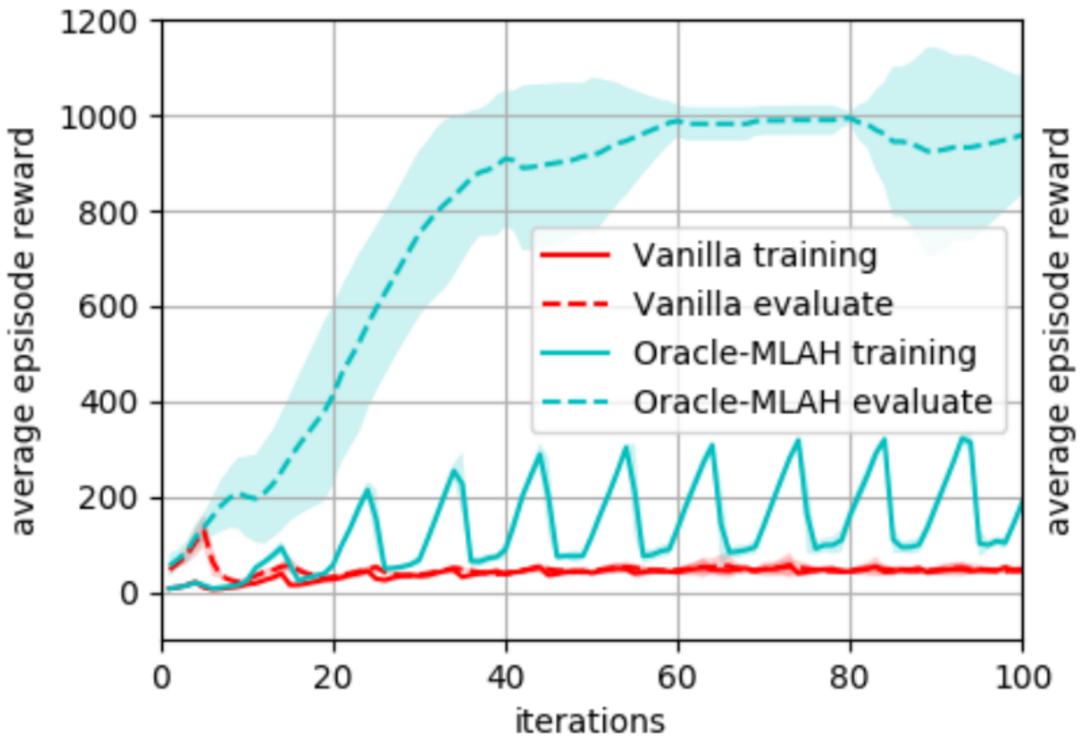
# Method

**Algorithm 1:** MLAH

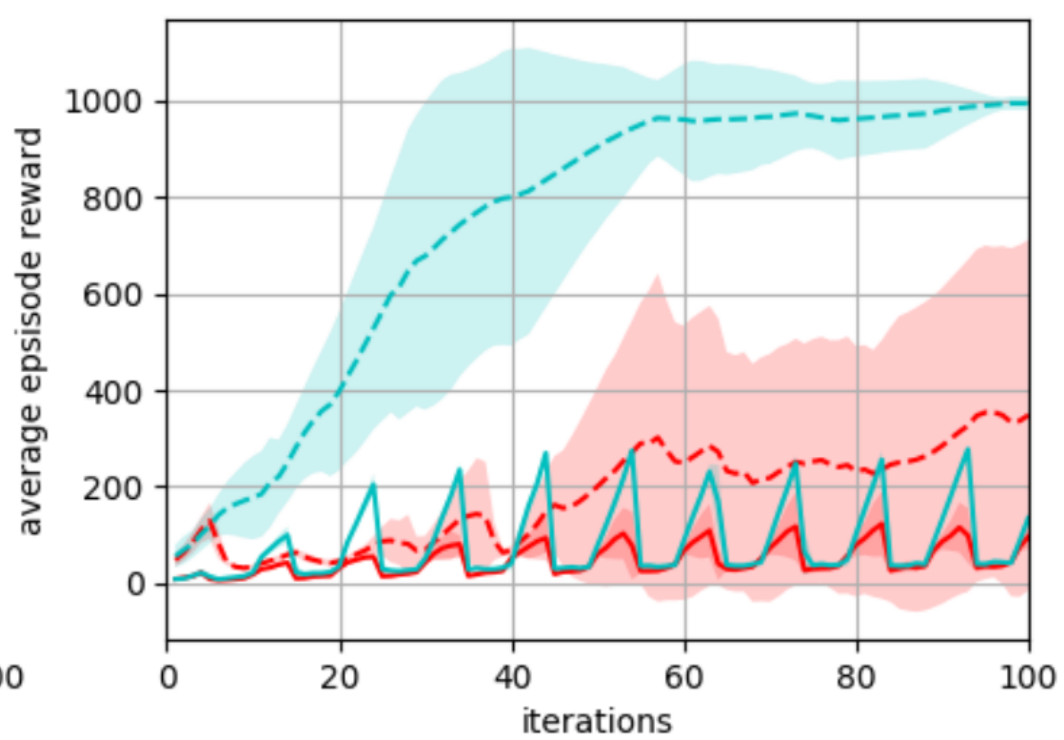
**Input :** $\pi_{nom}$ and $\pi_{adv}$ sub-policies parameterized by $\theta_{nom}$ and $\theta_{adv}$; Master policy $\pi_{master}$ with parameter vector $\phi$.

Initialize $\theta_{nom}$, $\theta_{adv}$, $\phi$
**for** *pre-training iterations [optional]* **do**
    Train $\pi_{nom}$ and $\theta_{nom}$ on only nominal experiences.
**end**
**for** *learning life-time* **do**
    **for** *Time steps $t$ to $t + T$* **do**
        Compute $\mathbf{A}_t$ over sub-policies (see eq. 4)
        select sub-policy to take action with $\pi_{master}$ using $\mathbf{A}_t$ as observations
    **end**
    Estimate all $A_{GAE}$ for $\pi_{nom}$, $\pi_{adv}$ over $T$
    Estimate all $A_{GAE}$ for $\pi_{master}$ over $T$ with respect to $\mathbf{A}_t$ observations
    Optimize $\theta_{nom}$ based on experiences collected from $\pi_{nom}$
    Optimize $\theta_{adv}$ based on experiences collected from $\pi_{adv}$
    Optimize $\phi$ based on all experiences with respect to $\mathbf{A}_t$ observations
**end**

# Result



1200
1000
800
600
400
200
0
0
20
40
60
80
100
iterations
average epsiode reward
Vanilla training
Vanilla evaluate
Oracle-MLAH training
Oracle-MLAH evaluate

1000
800
600
400
200
0
0
20
40
60
80
100
iterations
average epsiode reward

# Robust Deep Reinforcement Learning with Adversarial Attacks

Pattanaik, Anay, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary.
In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*

- Even naïve attack can degrade DRL algorithms
- Adversarial Training leads to significant increase in robustness to parameter variations for RL benchmarks
- Target RL algorithm:
  - DDQN
  - DDPG

# Perturbation

- Naïve attack: Random perturbation
- Gradient attack: Gradient based perturbation(FGSM)

# Adversarial attack

- Use beta distribution to sample noise from some distribution
- Sample multiple times and pick the largest one

---

**Algorithm 2** Naive attack (DDPG)

---

**procedure** NAIVE($Q^{target}, U, s, \epsilon, n, \alpha, \beta$) ▷ Naive attack function takes trained target critic network $Q^{target}$, trained actor network $U$, current state(s), adversarial attack magnitude constraint($\epsilon$), parameters of beta distribution($\alpha, \beta$) and number of times to sample noise($n$) as input

  $a^* = U^{(}s), Q^* = Q^{target}(s, a^*)$ ▷ Determine optimal action and action value function

  **for** $i = 1 : n$ **do** ▷ Sample a few times

    $n_i \sim beta(\alpha, \beta) - 0.5$ ▷ Sample noise

    $s_i = s + \epsilon * n_i$ ▷ Possible adversarial state determined by sampled noise

    $a_{adv} = U(s_i)$ ▷ Determine optimal action in potential adversarial state

    $Q_{adv}^{target} = Q^{target}(s, a_{adv})$ ▷ Determine the value of potential adversarial action corresponding to potential adversarial state for current state

    **if** $Q_{adv}^{target} < Q^*$ **then** ▷ if the potential adversarial state leads to bad action

      $Q^* = Q_{adv}^{target}$ ▷ Store the value function of that potential bad action

      $s_{adv} = s_i$ ▷ Store possible adversarial state

    **else**

      do nothing

    **end if**

  **end for**

  **return** $s_{adv}$ ▷ Adversarial state

**end procedure**

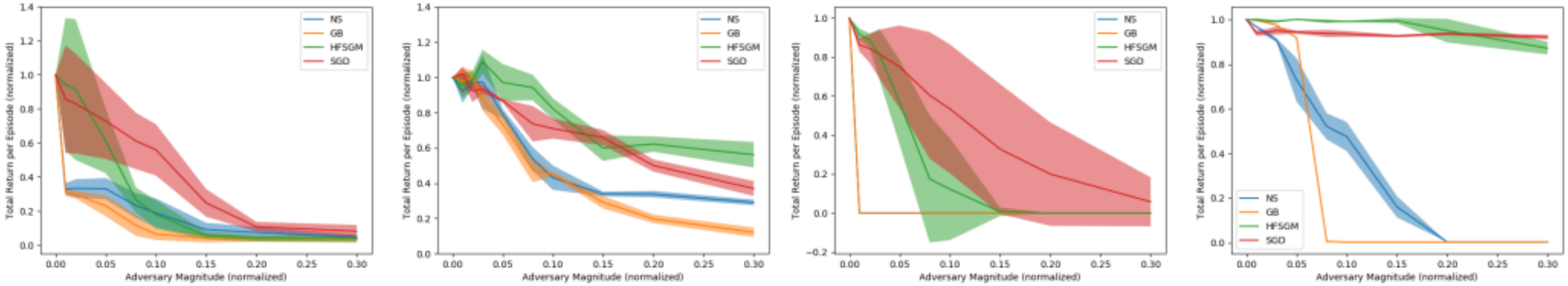
---

# Adversarial training

**Algorithm 5** Training with adversarial perturbation (DDQN)

**procedure** ADV TRAIN ($Q^{target}, Q$) ▷ Gradient based adversarial training method takes pre-trained network
  **for** $i = 1 : iterations$ **do** ▷ Train adversarially for number of timesteps
    Reset the environment and receive observation
    **while** not terminal or not max time steps per episode reached **do**
      $s_{adv} = Grad(Q^{target}, Q, s, \epsilon, n, \alpha, \beta)$ ▷ Fool the agent
      $a = arg \max_{a} Q(s_{adv,a})$ ▷ Fooled agent takes action according to behavior policy
      $s, r = Env(a, s)$ ▷ Environment returns next state and reward corresponding to state $s$ and action $a$
      Update the weights of network according to DDQN algorithm
    **end while**
  **end for**
**end procedure**

# Adversarial Attack performance




(a) DDQN Cart Pole (b) RBF Q Cart Pole (c) DDQN Mountain Car (d) RBF Q mountain car

# Robust

- Test the adversarial trained agent on a wide range of parameters and compared it to “vanilla” DRL