Aalto University

UNIVERSITY OF WATERLOO

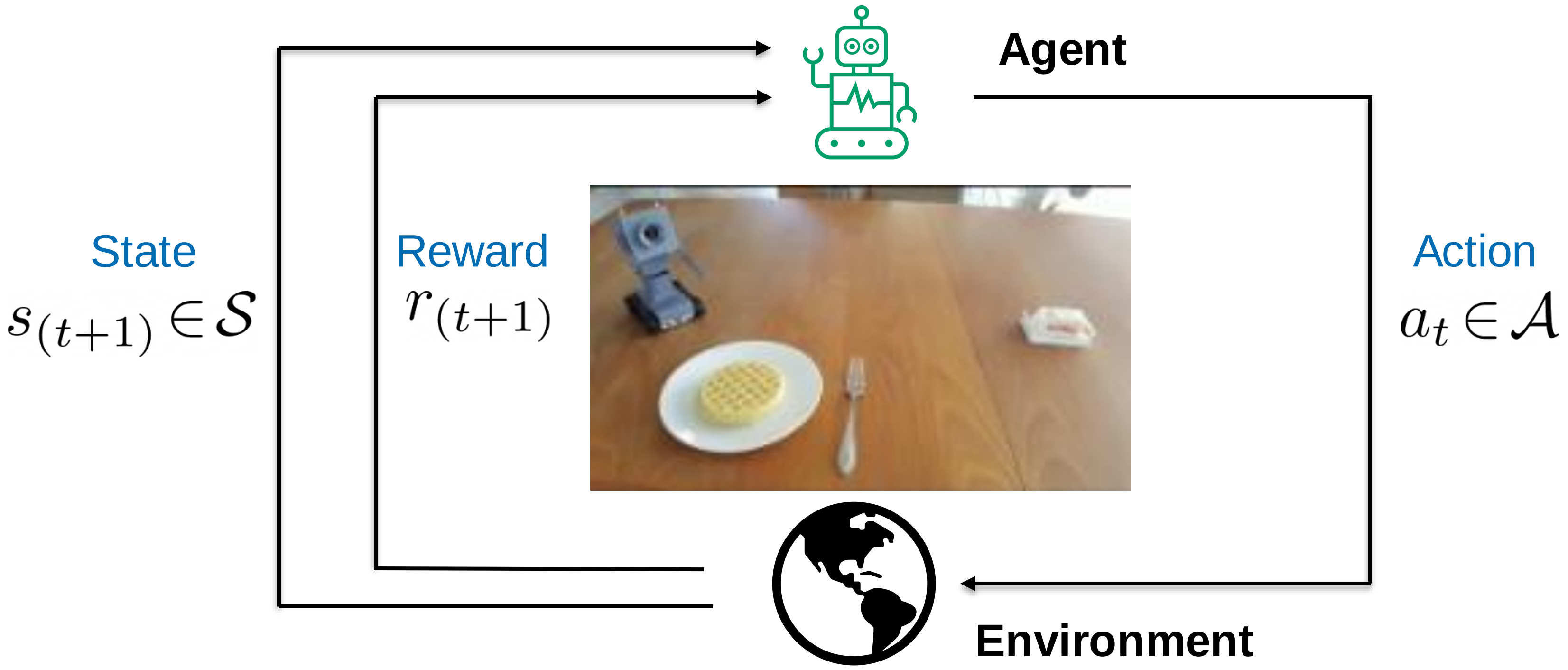
# Real-time Adversarial Perturbations against Deep Reinforcement Learning Policies: Attacks and Defenses

*Buse G. A. Tekgul, Shelly Wang, Samuel Marchal, N. Asokan*

# Reinforcement Learning

**In RL, an agent interacts with an environment to optimize its policy**

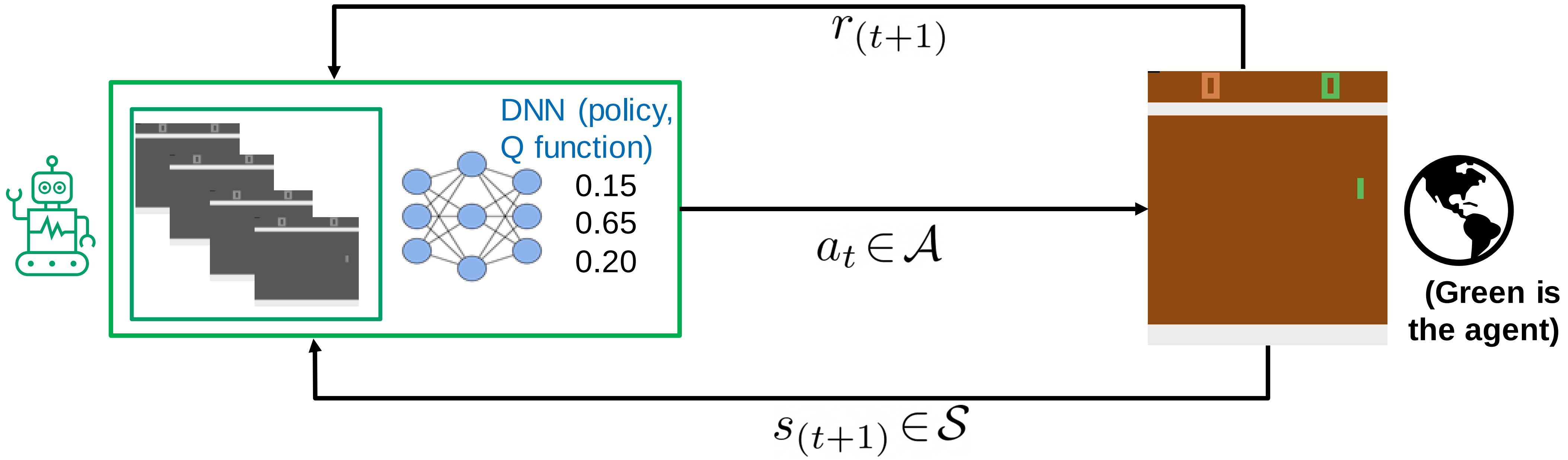
- Policy: Decision making strategy, $\pi(a_t|s_t) : \mathcal{S} \rightarrow \mathcal{A}$
- State-action value function: Helps optimizing the policy in discrete tasks, *Q(s,a)*



# Deep Reinforcement Learning (DRL)

**DRL learns successful policies directly from high-dimensional inputs**

- Reinforcement Learning (RL) defines the objective: maximize future reward
- Deep Neural Networks (DNN) provides the mechanism: approximates policy



# Adversarial Examples in DNN and DRL

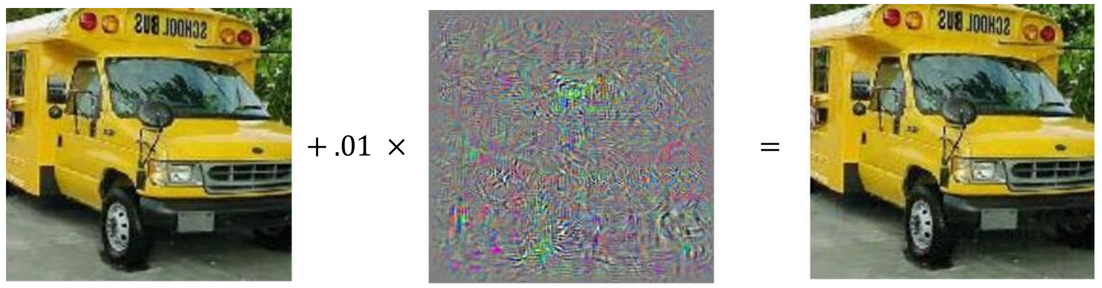
**Adversarial perturbation is added**

- DNNs[1]: ... into clean image → Classifier is victim, wrong label
- DRLs[2] : ... into clean state → Policy is victim, wrong action

**In DRL,**

- no 1-1 mapping between states and actions (no pre-defined labels)
- one successful adversarial example might not affect the task

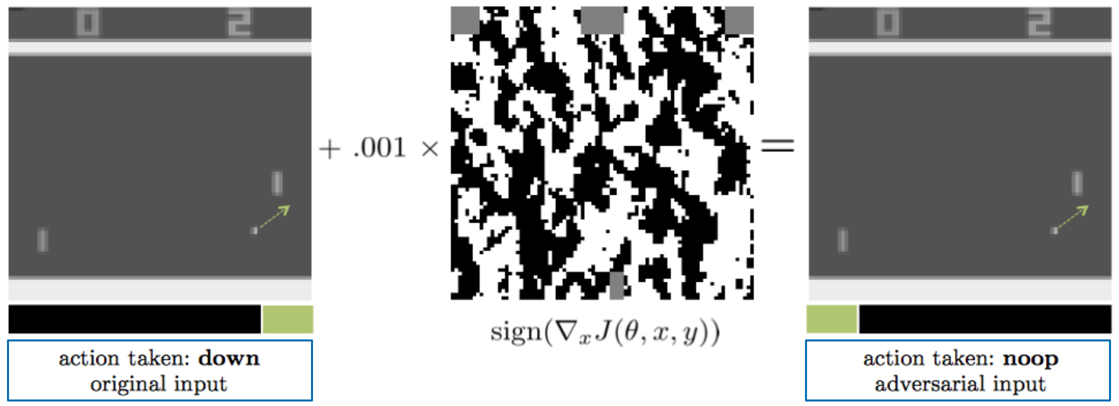
[1]



School bus

Ostrich

[2]



1. Szegedy et al., *"Intriguing Properties of Neural Networks"* arXiv, 2013. https://arxiv.org/abs/1312.6199v4
2. Huang et al. *"Adversarial Attacks on Neural Network Policies"*, arXiv 2017. https://arxiv.org/pdf/1702.02284.pdf

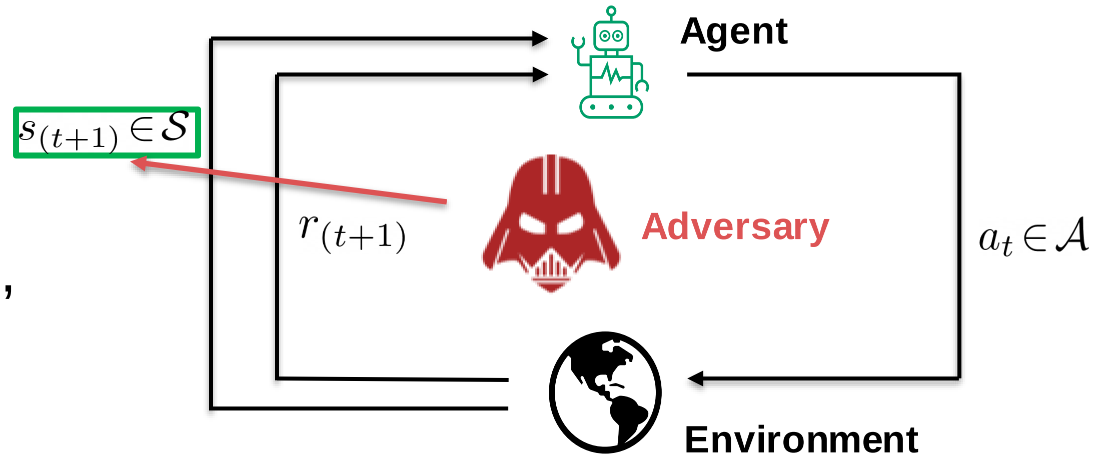
# Adversary Model

**Adversary:**

- wants a reinforcement learning agent to fail its task
- uses *state-action value function Q(s,a)* to generate sub-optimal actions for *discrete* tasks

**Adversarial capabilities:**

- has the knowledge of
  - RL algorithm and
  - DNN model used for victim's policy
- cannot reset environment, replay earlier state, or induce a delay during the task



# Realistic Adversaries in DRL

**A realistic attack**

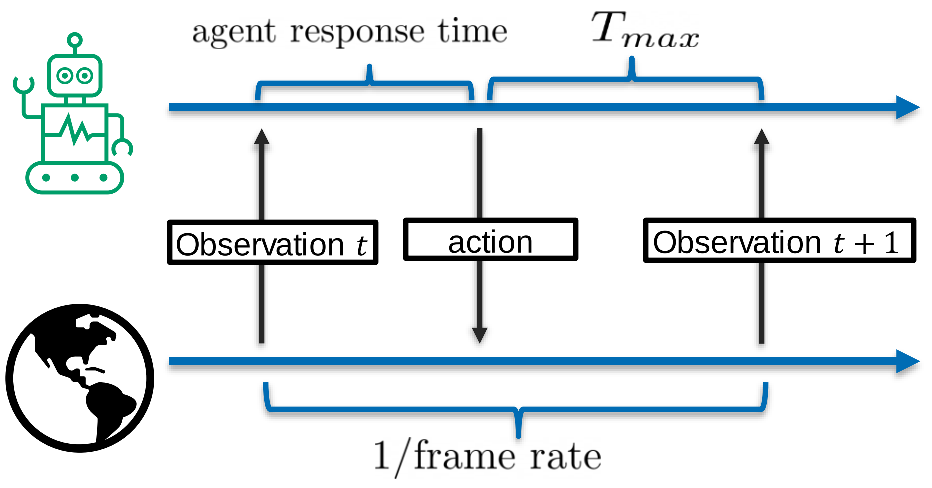
- cannot change the inner workings of victim agent (e.g. short-term memory, received rewards)
- should compute + add the perturbation fast enough to be implemented in real time
- The online cost should be less than

$$T_{max} = 1/\text{frame rate} - \text{agent response time}$$

**Prior attacks are not realistic, they**

- are too slow to be mounted in real time[1,2]
- modify the short term memory of victim[3]

***Can we effectively fool DRL policies in real-time?***



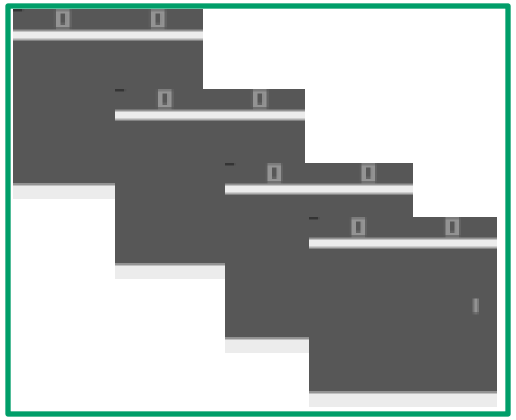
1. Lin, Yen-Chen, et al. *"Tactics of adversarial attack on deep reinforcement learning agents."* IJCAI 2017. https://arxiv.org/abs/1703.06748
2. Pan, Xinlei, et al. *"Characterizing Attacks on Deep Reinforcement Learning."* AAMAS 2022. https://arxiv.org/abs/1907.09470
3. Huang et al. *"Adversarial Attacks on Neural Network Policies"*, arXiv 2017. https://arxiv.org/pdf/1702.02284

# State- and Observation- Agnostic Perturbations

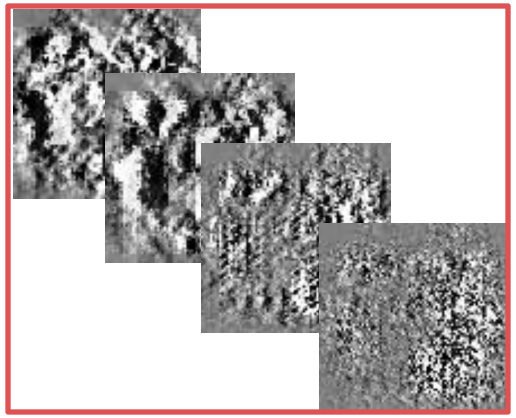
**Universal Adversarial Perturbations (UAP)[1] in DRL settings using**

- Find a sufficiently small perturbation $||r||_p = ||s_{adv(t)} - s_t||_p$ that results in sub-optimal actions for every perturbed state $s_{adv(t)}$
- **State-agnostic (UAP-S):** Perturbation is uniform across different states but is not uniform between the observations within a state
- **Observation-agnostic (UAP-O):** Perturbation is uniform across all observations

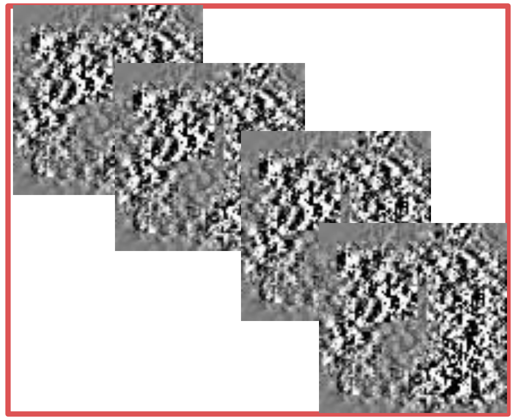
State



UAP-S



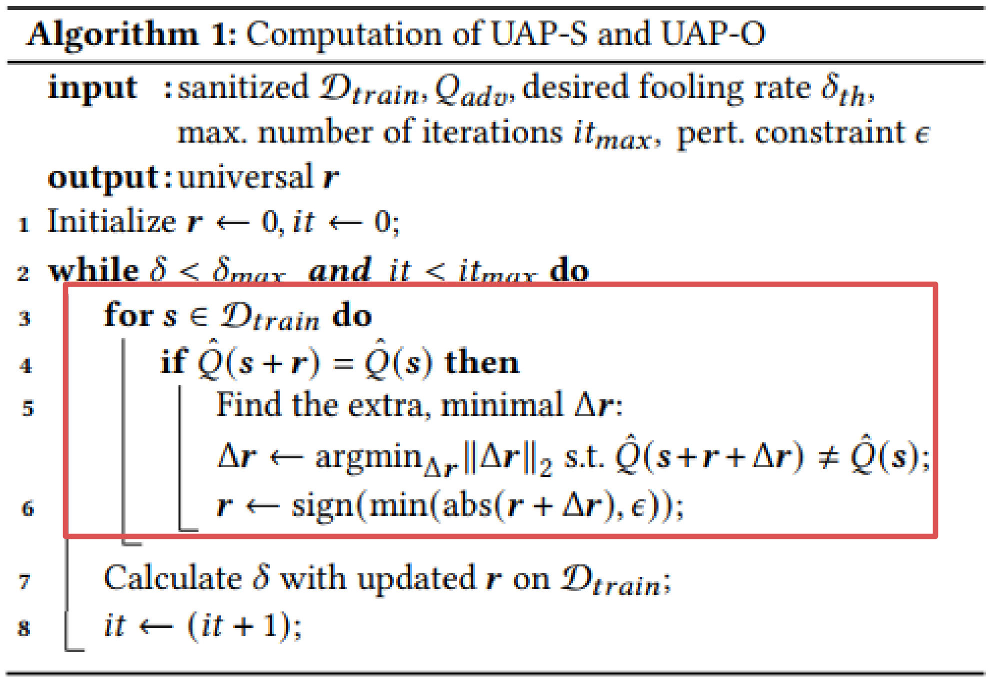
UAP-O



1. Moosavi-Dezfooli, Seyed-Mohsen, et al. *"Universal adversarial perturbations."* CVPR 2017. https://arxiv.org/abs/1610.08401

# State- and Observation- Agnostic Perturbations

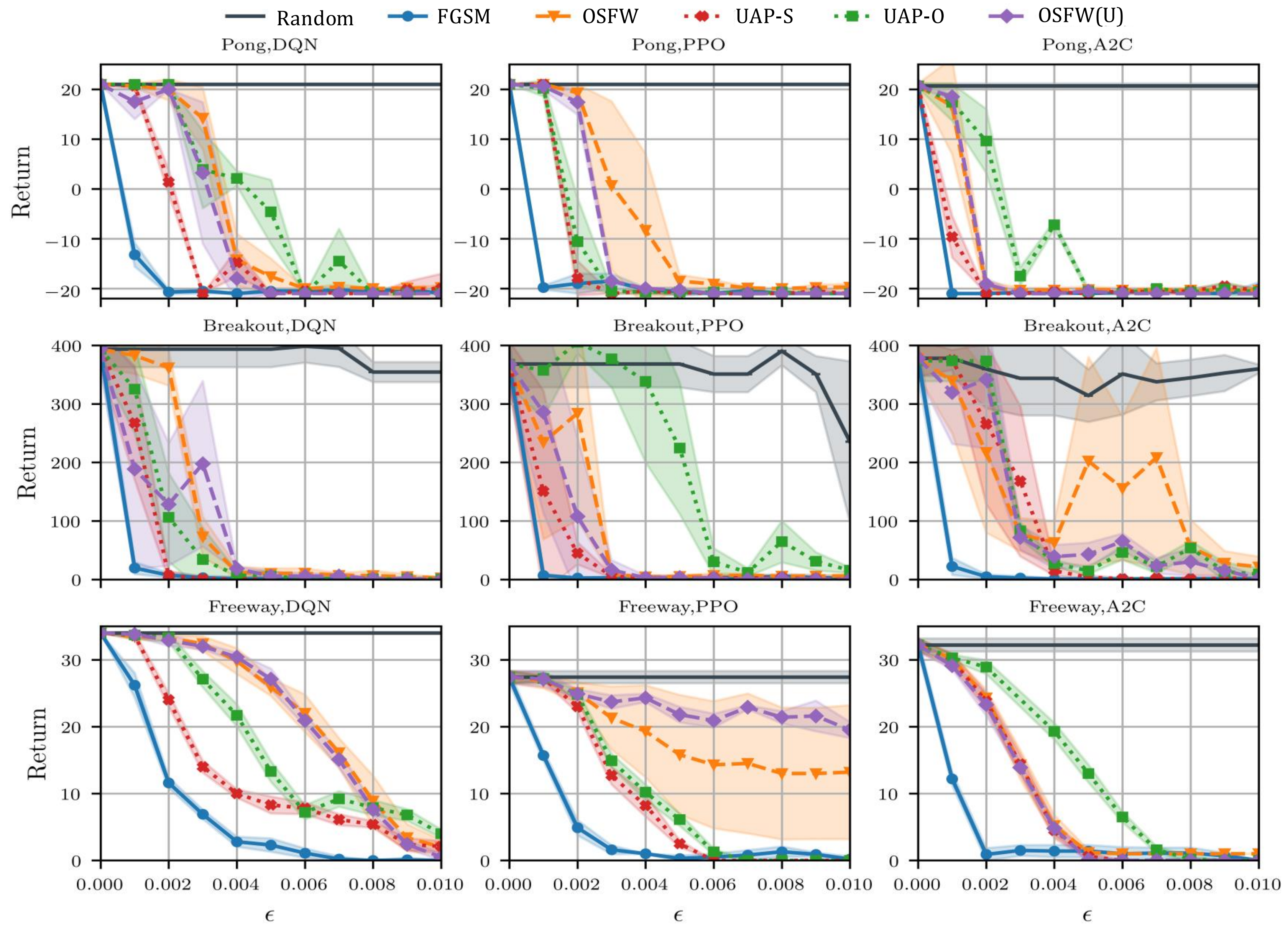
1. Collect training data by observing a full episode
2. Clone DNN (i.e., approximated state-action value function) of victim agent to an adversary's agent
3. Sanitize the training data by choosing only critical states
4. Compute the perturbation using Algorithm 1 in an offline manner
5. Add the perturbation to any other episode during the task



**Algorithm 1:** Computation of UAP-S and UAP-O

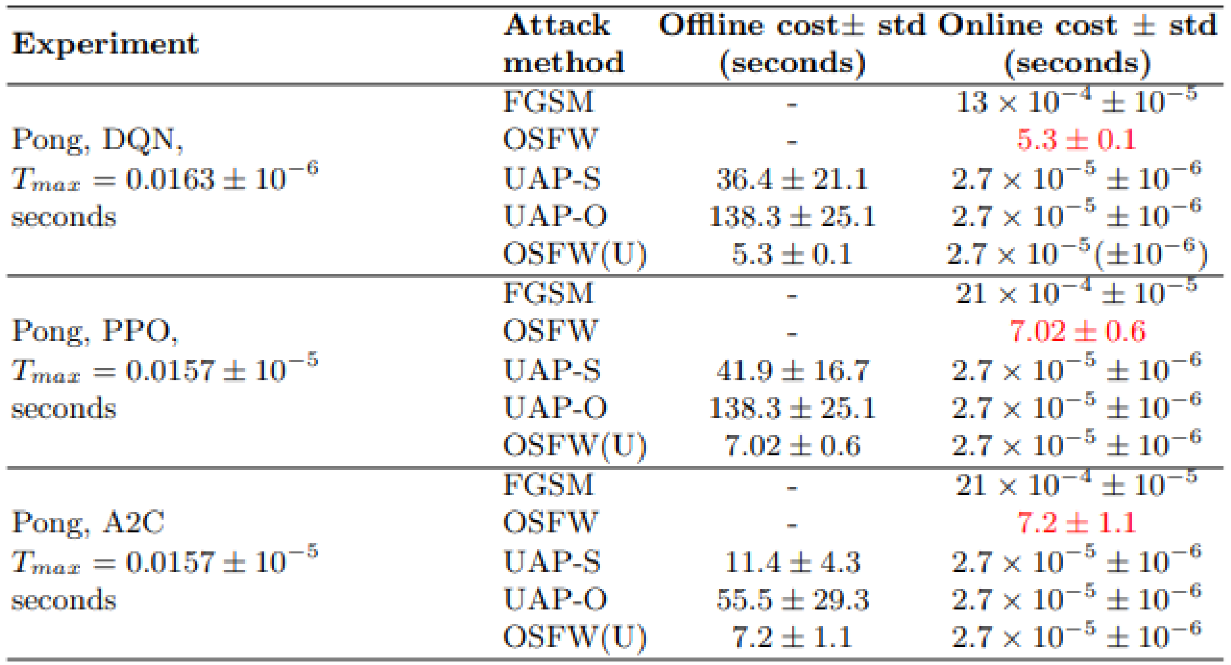
```
input  : sanitized $\mathcal{D}_{train}$, $Q_{adv}$, desired fooling rate $\delta_{th}$,
         max. number of iterations $it_{max}$, pert. constraint $\epsilon$
output : universal $\boldsymbol{r}$
1 Initialize $\boldsymbol{r} \leftarrow 0, it \leftarrow 0$;
2 while $\delta < \delta_{max}$ and $it < it_{max}$ do
3     for $s \in \mathcal{D}_{train}$ do
4         if $\hat{Q}(s + \boldsymbol{r}) = \hat{Q}(s)$ then
5             Find the extra, minimal $\Delta\boldsymbol{r}$:
              $\Delta\boldsymbol{r} \leftarrow \mathrm{argmin}_{\Delta\boldsymbol{r}} \|\Delta\boldsymbol{r}\|_2$ s.t. $\hat{Q}(s + \boldsymbol{r} + \Delta\boldsymbol{r}) \neq \hat{Q}(s)$;
6             $\boldsymbol{r} \leftarrow \mathrm{sign}(\min(\mathrm{abs}(\boldsymbol{r} + \Delta\boldsymbol{r}), \epsilon))$;
7     Calculate $\delta$ with updated $\boldsymbol{r}$ on $\mathcal{D}_{train}$;
8     $it \leftarrow (it + 1)$;
```

## Experimental Results: Performance Degradation



# Experimental Results: Computational Cost

- FGSM has low online cost, but requires rewriting victim agent’s memory
- OSFW has high online cost, so it misses perturbing 102 states on average
- UAP-S, UAP-O have high offline cost, but it does not interfere with the task
- UAP-S, UAP-O and OSWF(U) low online cost, can be implemented in real-time



| **Experiment** | **Attack method** | **Offline cost± std (seconds)** | **Online cost ± std (seconds)** |
|---|---|---|---|
| Pong, DQN, $T_{max} = 0.0163 \pm 10^{-6}$ seconds | FGSM | - | $13 \times 10^{-4} \pm 10^{-5}$ |
| | OSFW | - | $5.3 \pm 0.1$ |
| | UAP-S | $36.4 \pm 21.1$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |
| | UAP-O | $138.3 \pm 25.1$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |
| | OSFW(U) | $5.3 \pm 0.1$ | $2.7 \times 10^{-5}(\pm 10^{-6})$ |
| Pong, PPO, $T_{max} = 0.0157 \pm 10^{-5}$ seconds | FGSM | - | $21 \times 10^{-4} \pm 10^{-5}$ |
| | OSFW | - | $7.02 \pm 0.6$ |
| | UAP-S | $41.9 \pm 16.7$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |
| | UAP-O | $138.3 \pm 25.1$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |
| | OSFW(U) | $7.02 \pm 0.6$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |
| Pong, A2C $T_{max} = 0.0157 \pm 10^{-5}$ seconds | FGSM | - | $21 \times 10^{-4} \pm 10^{-5}$ |
| | OSFW | - | $7.2 \pm 1.1$ |
| | UAP-S | $11.4 \pm 4.3$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |
| | UAP-O | $55.5 \pm 29.3$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |
| | OSFW(U) | $7.2 \pm 1.1$ | $2.7 \times 10^{-5} \pm 10^{-6}$ |

11

# Experimental Results: Continuous Control

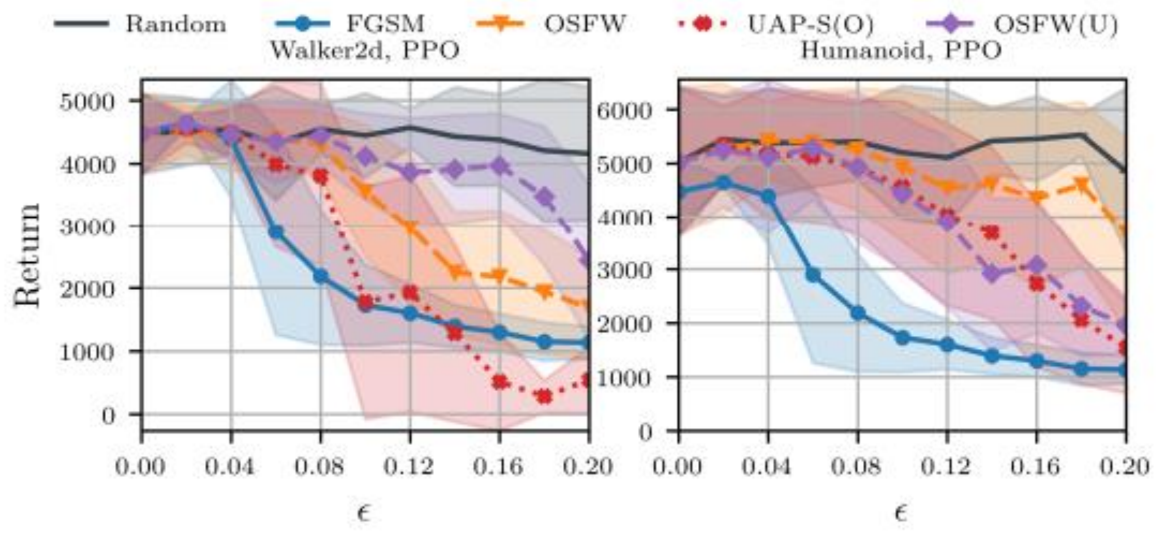
**Challenge:**

No discrete action space (lack of Q(s,a))

**Solution:**

- Exploit value function *V(s)* used in policy
- Modify Algorithm 1 using V(s)
- Goal: Decrease the evaluation of the state

**UAP-S and UAP-O generalize to continuous control**



| Experiment | Attack method | Offline cost± std (seconds) | Online cost ± std (seconds) |
|---|---|---|---|
| Walker2d, PPO, $T_{max} = 0.0079 \pm 10^{-5}$ seconds | FGSM | - | $31 \times 10^{-5} \pm 10^{-5}$ |
| | OSFW | - | $0.02 \pm 0.001$ |
| | UAP-S (O) | $8.75 \pm 0.024$ | $2.9 \times 10^{-5} \pm 10^{-6}$ |
| | OSFW(U) | $0.02 \pm 0.001$ | $2.9 \times 10^{-5} \pm 10^{-6}$ |
| Humanoid PPO, $T_{max} = 0.0079 \pm 10^{-6}$ seconds | FGSM | - | $35 \times 10^{-5} \pm 10^{-5}$ |
| | OSFW | - | $0.02 \pm 0.001$ |
| | UAP-S (O) | $35.86 \pm 0.466$ | $2.4 \times 10^{-5} \pm 10^{-6}$ |
| | OSFW(U) | $0.02 \pm 0.001$ | $2.4 \times 10^{-5} \pm 10^{-6}$ |

## Detection and Mitigation of Adversarial Perturbations

**In tasks that can end with clear negative results:**

- Losing a game
- Ends episode with negative returns

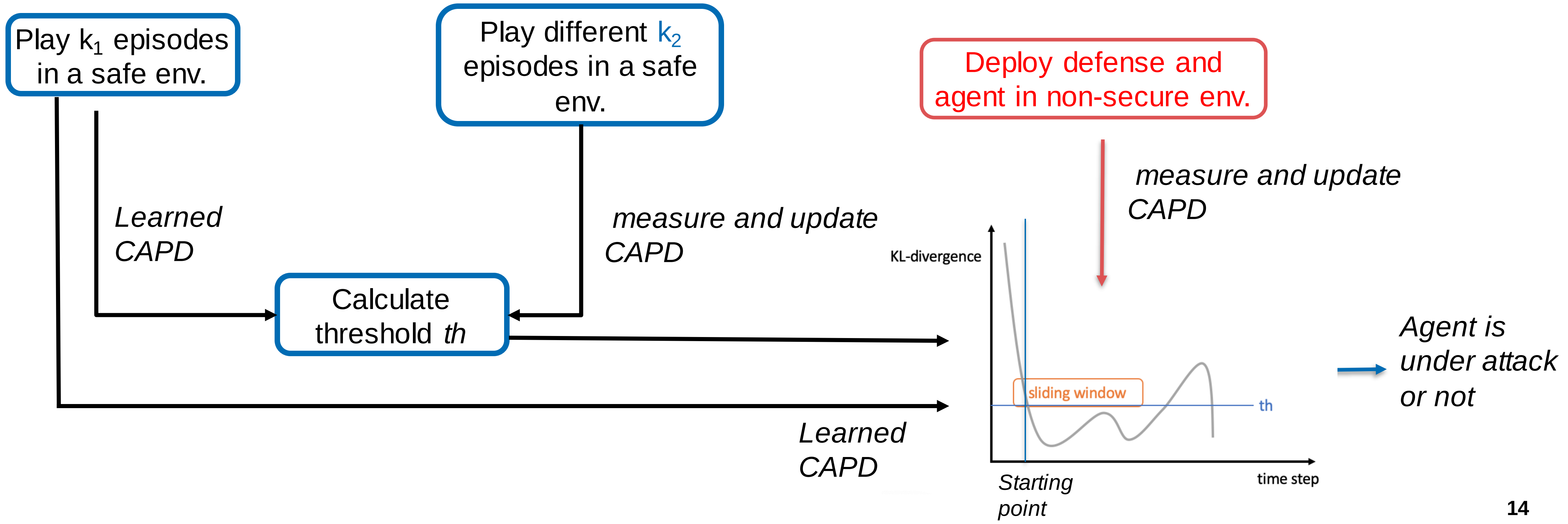
**The victim would be able to suspend/forfeit an episode if the adversary could be detected to prevent the negative outcome**

***Can we develop an effective detection mechanism that can detect the presence of the adversary?***

# AD$^3$ - Action Distribution Divergence Detector

## Threshold-based detection method

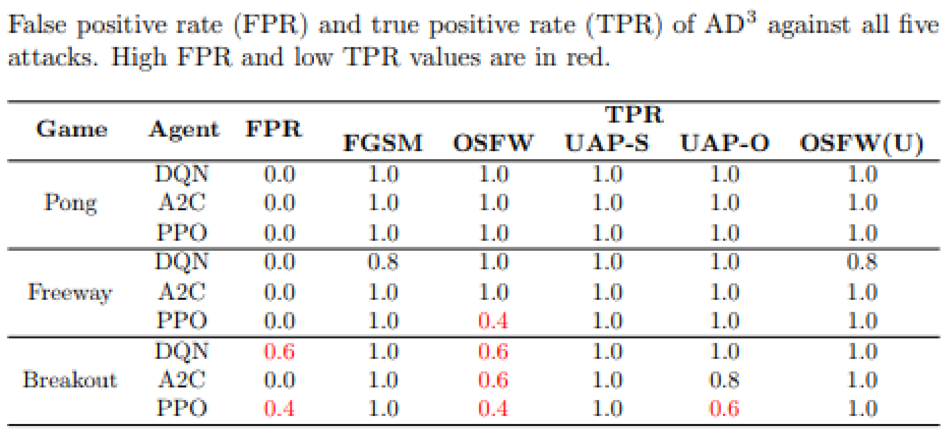
- Measures statistical distance between the *conditional action probability distributions (CAPD)*



Play $k_1$ episodes in a safe env.
Play different $k_2$ episodes in a safe env.
Deploy defense and agent in non-secure env.
Learned CAPD
measure and update CAPD
Calculate threshold *th*
measure and update CAPD
KL-divergence
sliding window
th
time step
Starting point
Learned CAPD
Agent is under attack or not

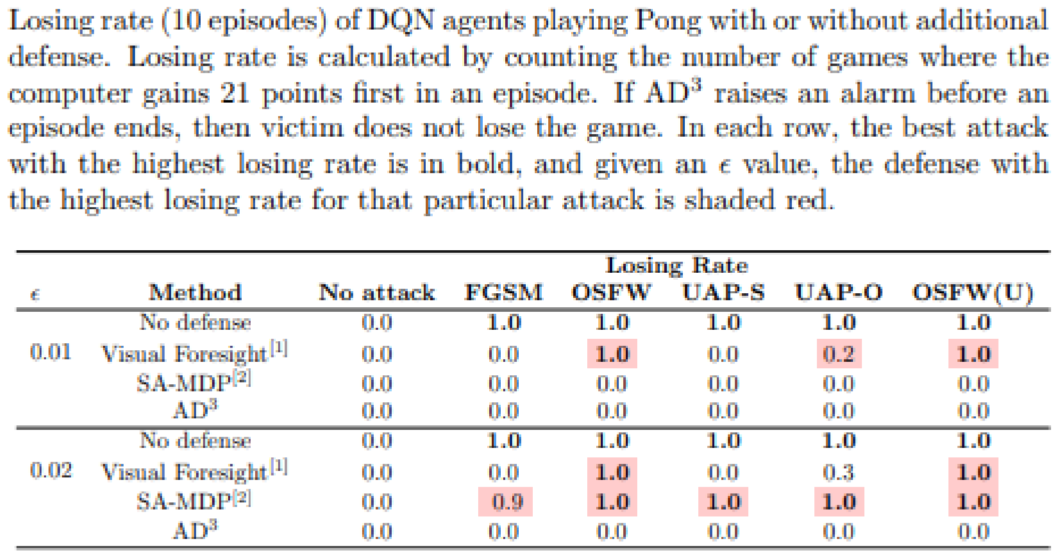
# Effectiveness of AD$^3$

- Effective in Pong for all agents against all attacks
- Less effective in Freeway especially against less effective attacks
- Not effective in Breakout with high false positive rate for DQN and PPO agents
- Useful in raising an alarm when the victim is in the direction of negative return (e.g., losing the game)



False positive rate (FPR) and true positive rate (TPR) of AD$^3$ against all five attacks. High FPR and low TPR values are in red.

| Game | Agent | FPR | TPR | | | | |
|---|---|---|---|---|---|---|---|
| | | | FGSM | OSFW | UAP-S | UAP-O | OSFW(U) |
| Pong | DQN | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | A2C | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | PPO | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Freeway | DQN | 0.0 | 0.8 | 1.0 | 1.0 | 1.0 | 0.8 |
| | A2C | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | PPO | 0.0 | 1.0 | 0.4 | 1.0 | 1.0 | 1.0 |
| Breakout | DQN | 0.6 | 1.0 | 0.6 | 1.0 | 1.0 | 1.0 |
| | A2C | 0.0 | 1.0 | 0.6 | 1.0 | 0.8 | 1.0 |
| | PPO | 0.4 | 1.0 | 0.4 | 1.0 | 0.6 | 1.0 |



Losing rate (10 episodes) of DQN agents playing Pong with or without additional defense. Losing rate is calculated by counting the number of games where the computer gains 21 points first in an episode. If AD$^3$ raises an alarm before an episode ends, then victim does not lose the game. In each row, the best attack with the highest losing rate is in bold, and given an $\epsilon$ value, the defense with the highest losing rate for that particular attack is shaded red.

| $\epsilon$ | Method | Losing Rate | | | | | |
|---|---|---|---|---|---|---|---|
| | | No attack | FGSM | OSFW | UAP-S | UAP-O | OSFW(U) |
| 0.01 | No defense | 0.0 | **1.0** | **1.0** | **1.0** | **1.0** | **1.0** |
| | Visual Foresight[1] | 0.0 | 0.0 | **1.0** | 0.0 | 0.2 | **1.0** |
| | SA-MDP[2] | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | AD$^3$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.02 | No defense | 0.0 | **1.0** | **1.0** | **1.0** | **1.0** | **1.0** |
| | Visual Foresight[1] | 0.0 | 0.0 | **1.0** | 0.0 | 0.3 | **1.0** |
| | SA-MDP[2] | 0.0 | 0.9 | **1.0** | **1.0** | **1.0** | **1.0** |
| | AD$^3$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

1. Lin, Yen-Chen, et al. *"Detecting adversarial attacks on neural network policies with visual foresight."* arXiv 2017. https://arxiv.org/abs/1710.00814
2. Zhang, Huan, et al. *"Robust deep reinforcement learning against adversarial perturbations on state observations."* NeurIPS2020 https://arxiv.org/abs/2003.08938

## Conclusion and Takeaways

**Thwarting DRL agents**

- UAP-S and UAP-O
  - have the same effectiveness as state-of-the-art attacks
  - can be mounted in real time

**Detecting the presence of adversary**

- Action Distribution Divergence Detector, AD$^3$
  - Defense relying on the temporal coherence of actions
  - Useful to combine with other recovery methods/defenses

https://ssg.aalto.fi/research/projects/

Open positions at University of Waterloo: https://asokan.org/asokan/research/SecureSystems-open-positions-Jul2021.php