black hat USA 2020

AUGUST 5-6, 2020

BRIEFINGS

Ruling StarCraft Game Spitefully – Exploiting the Blind Spot of AI-Powered Game Bots

Xinyu Xing, **Wenbo Guo, Xian Wu**, Jimmy Su

Deep Reinforcement Learning

- Deep learning has dominated many fields:
  - Computer vision [Krizhevsky, et al, 2012], natural language process [Socher, et al, 2011].
  - Malware detection [Wang, et al, 2017], intrusion detection [Javaid, et al, 2016].

- Integrating DL into reinforcement learning – DRL:
  - Extraordinary performance on many decision-making tasks.
    - Robotic control [Kober, et al, 2009].
    - Autonomous vehicles [O'Kelly, et al, 2019].
    - Finance and business management [Cai, et al, 2018].

DRL in Games

- Board games:
  - Go: DeepMind – AlphaGo [1], Facebook – OpenGo [2].
  - Poker games: Texas hold'em [3].
  - DeepMind – OpenSpiel [4].
    - An open source collection of game environments.
    - Single- and multi- players.

Credit: https://www.irishtimes.com/business/technology/podcast-dives-deep-into-science-and-ai-behind-deepmind-1.3997759

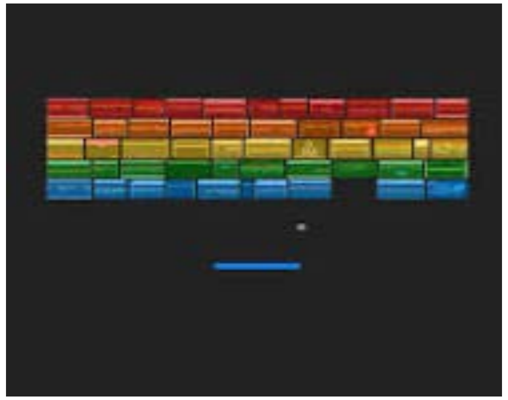
Credit: https://github.com/deepmind/open_spiel/

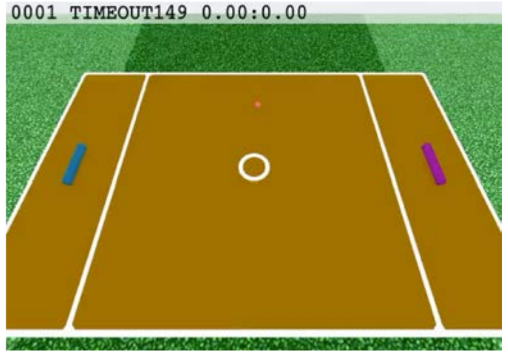
Credit: https://www.nature.com/articles/d41586-019-02156-9

DRL in Games

- Simulation games:
  - OpenAI – Gym [1].
    - Open source toolkit for developing DRL method to control robots, play games, etc.
    - Atari [2], Roboschool [3], and **MuJoCo** [4].

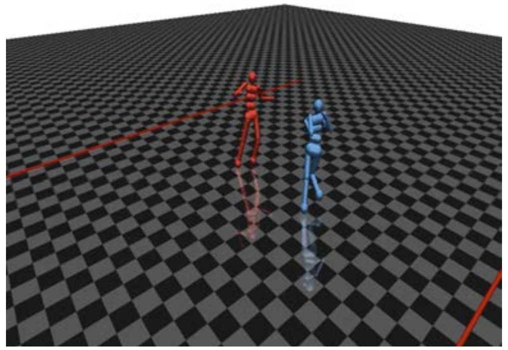
- Real-time strategy games:
  - StarCraft II [5].
  - Dota 2 – OpenAI Five [6].
  - Single- and multi- players.


Atari breakout. Credit: Wikipedia



Roboschool Pong. Credit: [3]


Dota 2 -- defending base. Credit [6]


MuJoCo You-Shall-Not-Pass.


StarCraft II

Attacks on DRL

- Adversarial attacks on deep neural networks:
  - Training phase – data poisoning attacks.
    - Injecting a backdoor into a DNN [Liu, et al, 2017].
  - Testing phase – adversarial samples.
    - An imperceptible perturbation on the input cause a dramatic change to the output [Goodfellow, et al, 2015].

- Deep reinforcement learning is also vulnerable to adversarial attacks.
  - Perturbing an agent's observation, action, or reward and force it to fail the task.
    - Involving hijacking a game system – **not practical.**

Enabling a practical adversarial attack against a master agent in a two-agent game environment.

Agenda

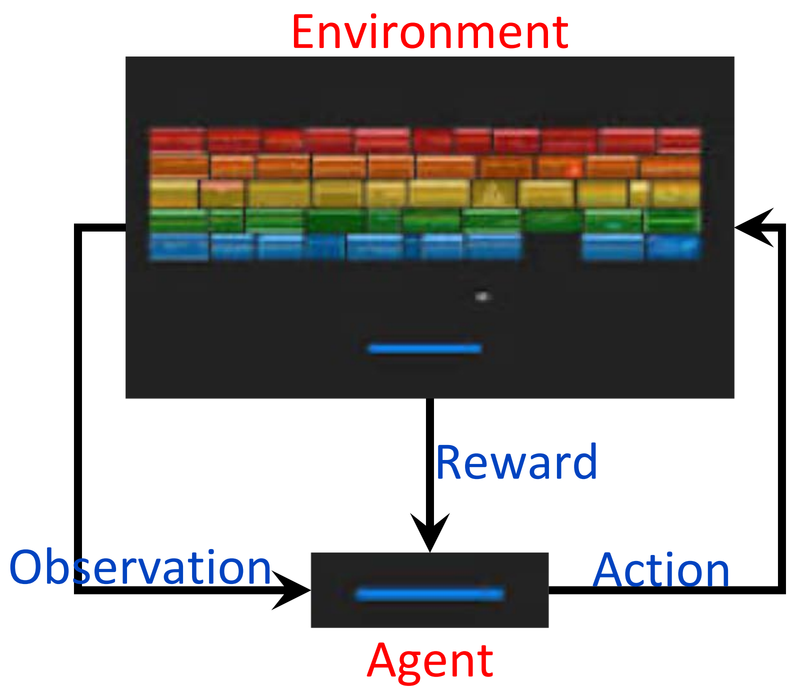
- DRL basics.
  - Modeling an RL problem.
  - Solving an RL problem – training an DRL agent.
- DRL-powered games.
  - Two-agent games: MuJoCo, StarCraft II.
  - Code structure of training an DRL bot.
- Existing attacks on DRL.
  - Perturbation attacks (Extension of attacks on DNN)
  - Practical adversarial agent attack.
- Our attack methodology.
- Evaluation.
- Conclusion.

Agenda

- DRL basics.
  - Modeling an RL problem.
  - Training an DRL agent.
- DRL-powered games.
  - Two-agent games: MuJoCo, StarCraft II.
  - Code structure of training an DRL bot.
- Existing attacks on DRL.
  - Perturbation attacks (Extension of attacks on DNN)
  - Practical adversarial agent attack.
- Our attack methodology.
- Evaluation.
- Conclusion.

Modeling an RL Problem

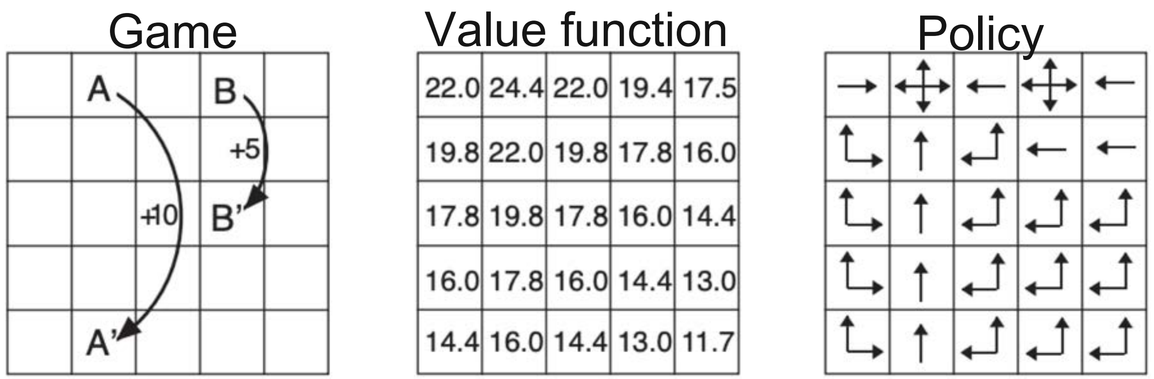
- An RL problem – a sequential decision making problem.
  - An agent observes and interacts with an environment through a series of actions.
  - The agent receives a reward each time taking an action.



- At each time step, system is at a certain state.
  - Agent
    - Receive an observation.
    - Executes an action.
  - Environment
    - Receive this action.
    - Transit to the next state based on the transition dynamics.
    - Emit an reward.
    - Emit the next observation.

Solving an RL Problem

- An RL problem – a sequential decision making problem.
  - The goal of an agent is to maximize its total amount of rewards.
  - The goal of an RL algorithm is to learn an optimal policy, following which the agent could receive a maximum amount of rewards over time.

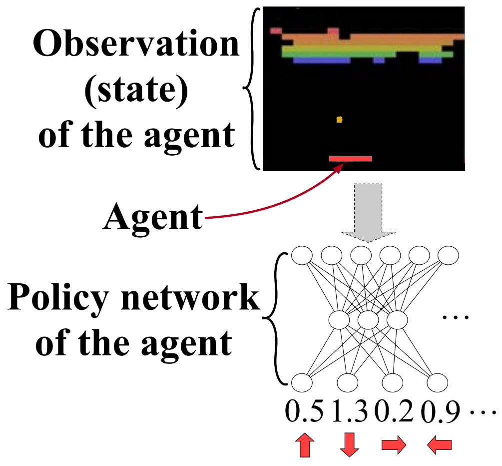


Demonstrations of resolving an optimal policy. Credit: David Silver [1]

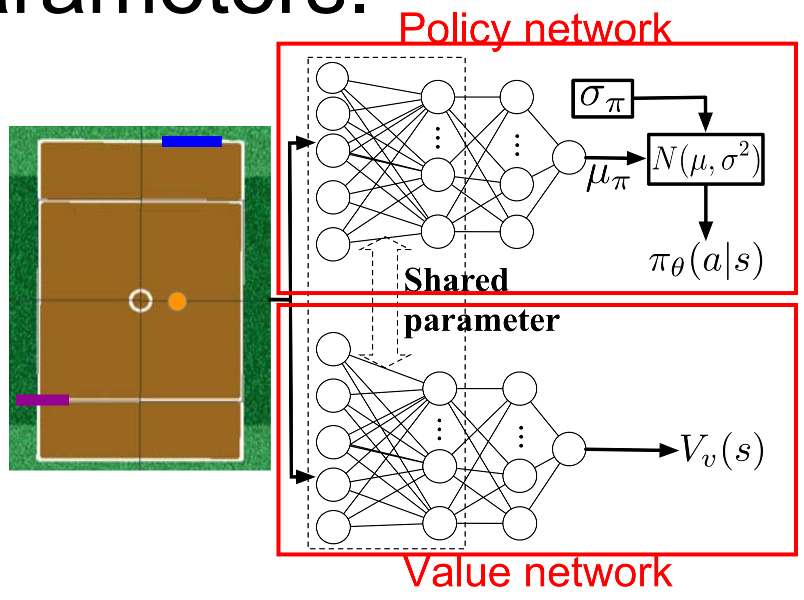
- In RL, the total reward of an agent is formulated as value functions.
  - State-value function: the expected total reward for an agent starting from a state and taking actions by following its policy.
  - Action-value function: the expected total reward for an agent starting from a state and taking a specific action by following its policy.
  - An optimal policy can be obtained by maximizing the value functions.

Training an DRL Agent

- In DRL, an agent is usually modeled as an DNN.
  - Policy network.
  - Taking as input the observation, and output the corresponding action.
  - Learning a policy is to solving the parameters of this neural network.



- Policy gradient methods - solving the network parameters.
  - Goal: optimizing the value-function.
  - Using another network to approximate the value-function.
  - In each iteration:
    - Updating the value network by minimizing the approximation errors.
    - Updating the policy network by maximizing the value function.
    - They usually share parameters.

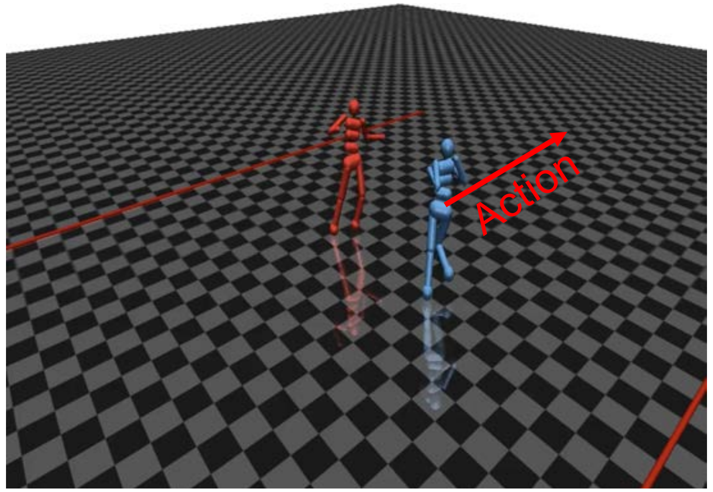


Agenda

- DRL basics.
  - Modeling an RL problem.
  - Training an DRL agent.
- DRL-powered games.
  - Two-agent games: MuJoCo, StarCraft II.
  - Code structure of training an DRL bot.
- Existing attacks on DRL.
  - Perturbation attacks (Extension of attacks on DNN)
  - Practical adversarial agent attack.
- Our attack methodology.
- Evaluation.
- Conclusion.

DRL-powered Games

- Two-party MuJoCo games.
  - Observation: the current status of the environment: the agent's and its opponent's status.
  - Action: the agent's movement (moving direction and velocity).
  - Reward: the agent's status and win/lose.
- StarCraft II games.
  - Observation: spatial condition of the map and the amount of resources.
  - Action: building, producing, harvesting, attacking.
  - Reward: game statistics and win/lose.



Training an DRL Bot

- Overall workflow.
  - Taking proximal policy optimization (PPO) as an example.

Initialize the network parameters

Number of iterations

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Credit: [1]

Collecting training trajectories based on the current policy

Updating the policy network

Updating the value function network

Training an DRL Bot

- Take StarCraft II as an example.
  - Pysc2 [1]
  - Pysc2Extension [2]
  - TStarBot1[3]
- Code structure
  - agents: defining and constructing two networks.
    - Taking as input observation and output action.
  - envs: environment wrapper.
    - Taking action as input and output observation and reward.
  - bin: running the agent in the environment and training the agent.

Training an DRL Bot

- Training an DRL bot for a game.
  - Usually adopting the self-play mechanism.
- The structure of the main file.
  - Defining an environment using the environment wrapper.
  - Defining an actor to collect the trajectories.
    - Running the current agent in the environment.
  - Defining a learner to train the agent.
    - Receiving the collected trajectories and updating the networks.

```python
def start_actor():
  tf_config(ncpu=2)
  random.seed(time.time())
  game_seed =  random.randint(0, 2**32 - 1)
  print("Game Seed: %d" % game_seed)
  env = create_selfplay_env(game_seed)
  policy = {'lstm': LstmPolicy,
            'mlp': MlpPolicy}[FLAGS.policy]
  actor = PPOSelfplayActor(
      env=env,
      policy=policy,
      unroll_length=FLAGS.unroll_length,
      gamma=FLAGS.discount_gamma,
      lam=FLAGS.lambda_return,
      model_cache_size=FLAGS.model_cache_size,
      model_cache_prob=FLAGS.model_cache_prob,
      prob_latest_opponent=0.0,
      init_opponent_pool_filelist=FLAGS.init_oppo_pool_filelist,
      freeze_opponent_pool=False,
      learner_ip=FLAGS.learner_ip,
      port_A=FLAGS.port_A,
      port_B=FLAGS.port_B)
  actor.run()
  env.close()
```

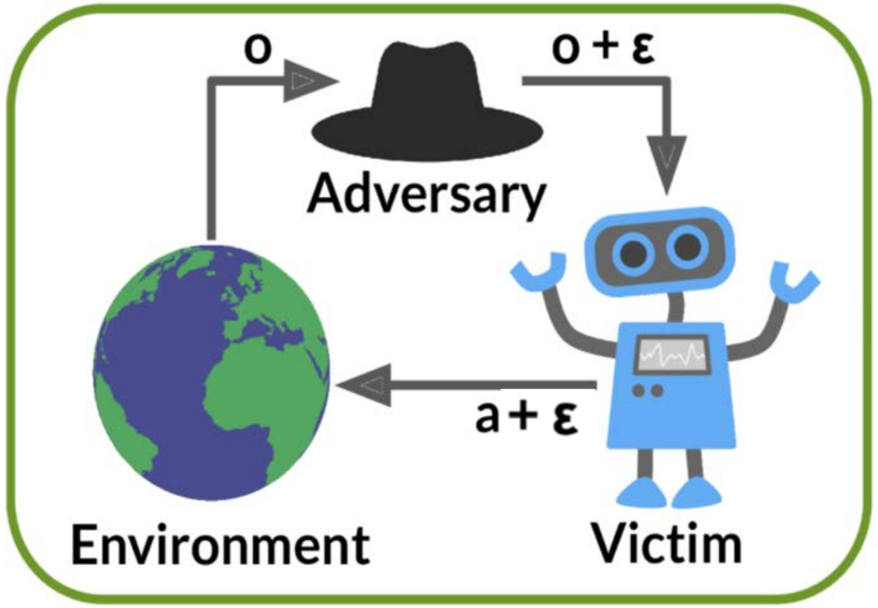
```python
learner = PPOLearner(env=env,
                     policy=policy,
                     unroll_length=FLAGS.unroll_length,
                     lr=FLAGS.learning_rate,
                     clip_range=FLAGS.clip_range,
                     batch_size=FLAGS.batch_size,
                     ent_coef=FLAGS.ent_coef,
                     vf_coef=FLAGS.vf_coef,
                     max_grad_norm=0.5,
                     queue_size=FLAGS.learner_queue_size,
                     print_interval=FLAGS.print_interval,
                     save_interval=FLAGS.save_interval,
                     learn_act_speed_ratio=FLAGS.learn_act_speed_ratio,
                     save_dir=FLAGS.save_dir,
                     init_model_path=FLAGS.init_model_path,
                     port_A=FLAGS.port_A,
                     port_B=FLAGS.port_B)
learner.run()
```

Agenda

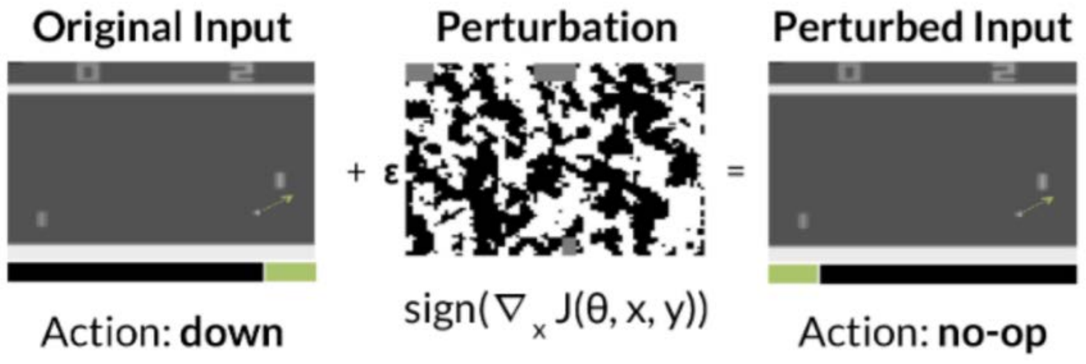
- DRL basics.
  - Modeling an RL problem.
  - Training an DRL agent.
- DRL-powered games.
  - Two-agent games: MuJoCo, StarCraft II.
  - Code of training an DRL bot.
- Existing attacks on DRL.
  - Perturbation-based attacks (Extension of attacks on DNN).
  - Practical adversarial agent attack.
- Our attack methodology.
- Evaluation.
- Conclusion.

Perturbation-based Attacks

- Threat model.
  - Perturbating the observations and force the policy network to output a series of sub-optimal actions.
  - Perturbating the output actions of the policy network.
- Example.
  - Generating perturbations by using the existing attacks on DNN.
  - Adding it to the observation (snapshot of the environment)



Credit: [1]



Credit: Huang, et al, 2017.

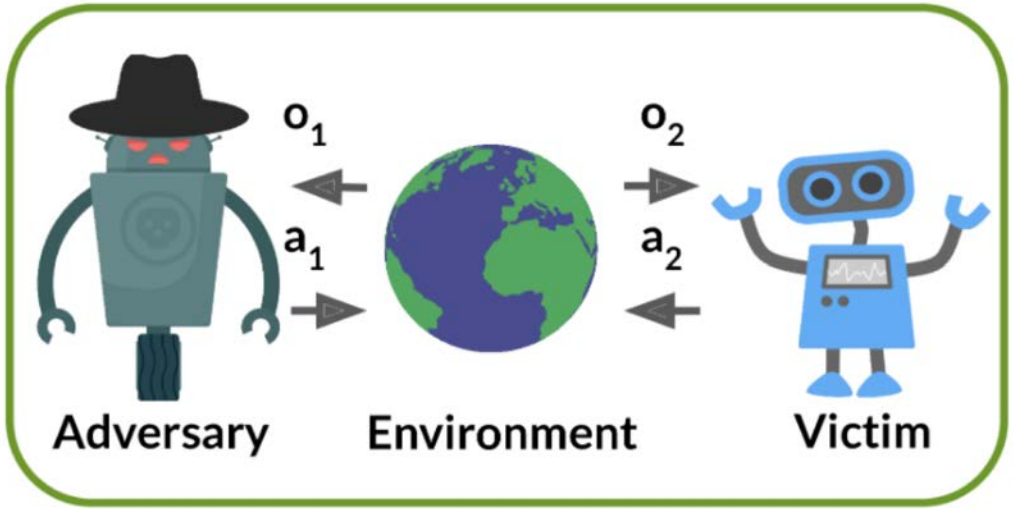
Perturbation-based Attacks

- In the game setup, requiring the attacker to hijack the game server.
  - Identifying and exploiting the vulnerabilities of the server.
  - Bypassing the defense mechanism in the server.
  - Requiring professional hackers tremendous effort and time.
  - Not a practical setup for beating an master agent of a two-party game.

Adversarial Agent Attack

- Threat model.
  - Attacker is not allowed to hijack the information flow of the victim agent.
    - Manipulating the observation, action, and reward.
  - Attacker could train an adversarial agent by playing with the victim agent.

- More practical in games.
  - No need to hack the game system.
  - Any player could play with a master agent freely.



Credit: [1]

Adversarial Agent Attack

- Existing technique [Gleave, et al, 2020].
  - Treating the victim agent as a part of the environment.
  - Training an agent to collect maximum rewards in the environment embedded with the victim.
  - Maximizing the training agent's value function by using the PPO algorithm.
  - Expecting to obtain a policy that could beat the victim.

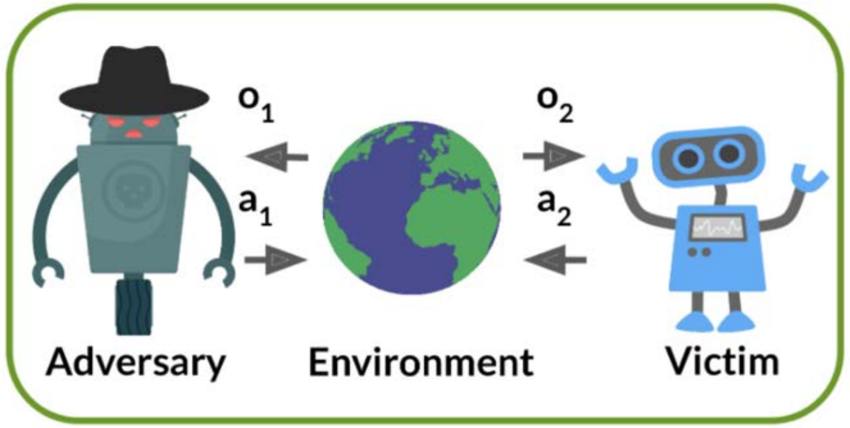
- Limitations
  - Do not explicitly disturbing the victim agent.
  - The training algorithm has less guidance for identifying the weakness of the victim.
  - Cannot establishing a high game winning rate.

Agenda

- DRL basics.
  - Modeling an RL problem.
  - Training an DRL agent.
- DRL-powered games.
  - Two-agent games: MuJoCo, StarCraft II.
  - Code of training an DRL bot.
- Existing attacks on DRL.
  - Perturbation attacks (Extension of attacks on DNN)
  - Practical adversarial agent attack.
- **Our attack methodology.**
- Evaluation.
- Conclusion.

Our attack

- Threat model.
  - Practical threat model.
  - Not allow to manipulate environment, opponent network.



Credit: [1]

- High-level idea.
  - Adversarial agent learns to disturb its opponent.
    - Training an adversarial agent to not only maximizing its reward but also minimizing its opponent's reward.
    - Letting the adversarial agent take an action that deviates the victim's next action.

Our attack

- Achieving the first goal.
  - Approximating the victim value function.
  - Augmenting the objective function with a term that minimizes the victim reward.

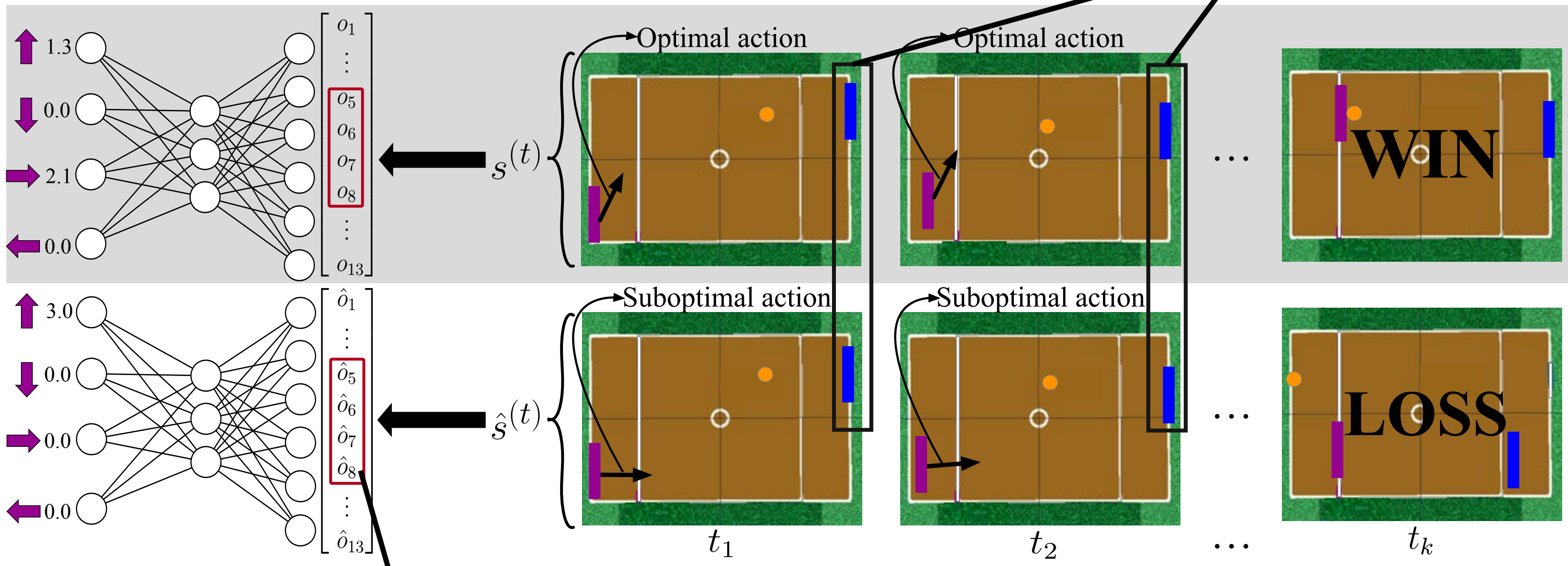
- Example – Collecting resources.
  - Without the added term.
    - The adversarial agent focuses only on optimizing its strategy to correct more resources.
  - With the added term.
    - The adversarial agent learns to block the victim from collecting resources.

Our attack

- Achieving the second goal.
  - Explaining the actions of the victim and find out the time steps when victim takes action based on the adversarial agents.
  - The adversarial agent takes an action that introduce maximum deviation to the victim's action at these critical time steps.

Adversarial agent changes its action and introduce an change to the environment.

- Example.



This change forces the victim to choose a sub-optimal action.

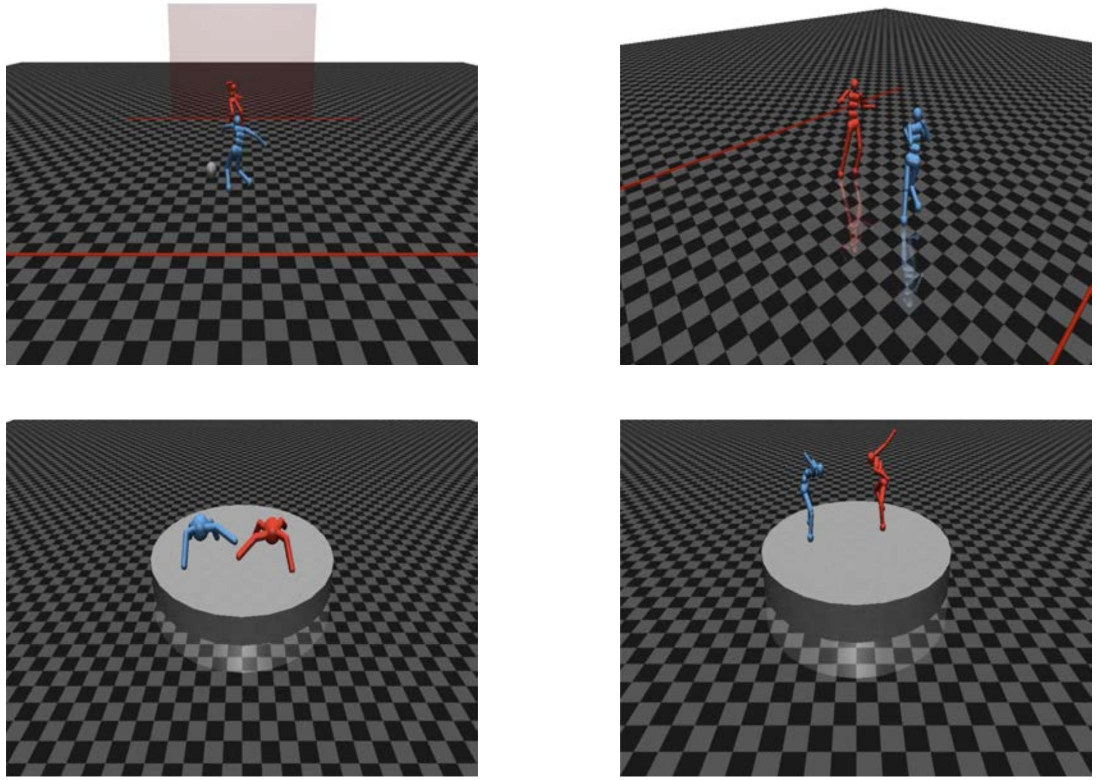
The part of the victim agent observation that represents the adversary.

Agenda

- DRL basics.
  - Modeling an RL problem.
  - Training an DRL agent.
- DRL-powered games.
  - Two-agent games: MuJoCo, StarCraft II.
  - Code of training an DRL bot.
- Existing attacks on DRL.
  - Perturbation attacks (Extension of attacks on DNN)
  - Practical adversarial agent attacks.
- Our attack methodology.
- Evaluation.
- Conclusion.

Evaluation setup

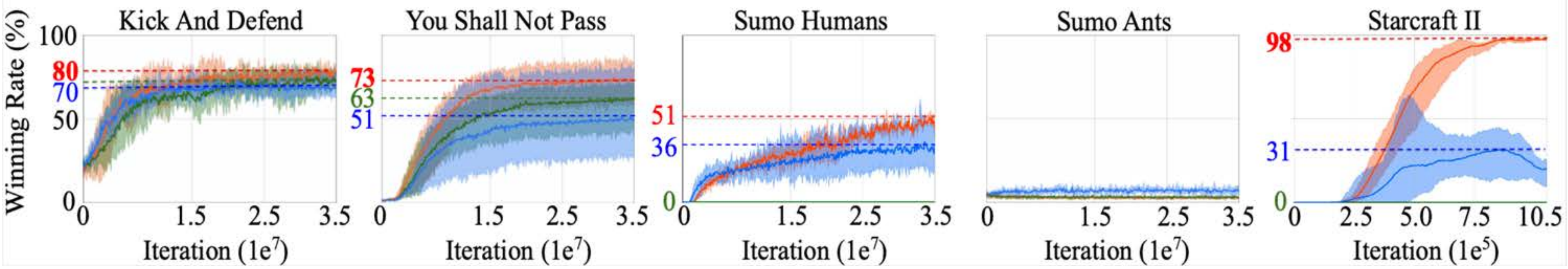
- Selected games.
  - MuJoCo – victim (blue), adversary (red).
    - Kick-And-Defend
    - You-Shall-Not-Pass
    - Sumo-Ants
    - Sumo-Humans
  - StarCraft II – Zerg vs. Zerg.

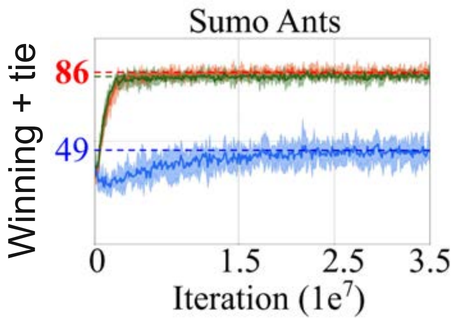


- Measuring and Reporting the winning rate of the adversarial agent each time its policy is updated during the training process.

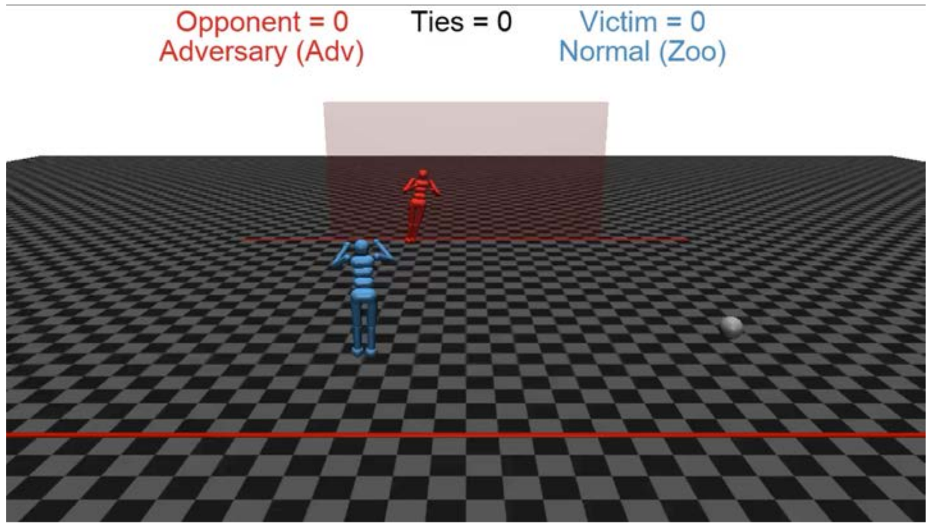
Quantitively Evaluation

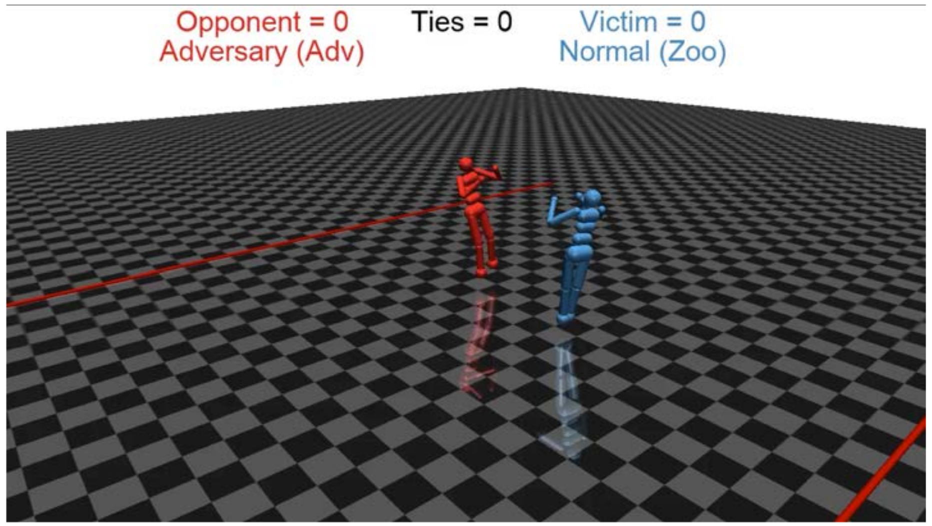
- Comparison of winning rates.
  - Red: our attack; blue: existing adversarial agent attack [Gleave, et al, 2020].
  - Our attack outperforms the existing attack on most games.
  - Sumo-Ants: Improve the non-lose rate.
    - Almost cannot win.
    - Low observation dimensions
    - Hard to disturb the victim via the adversarial actions.





Demo Examples



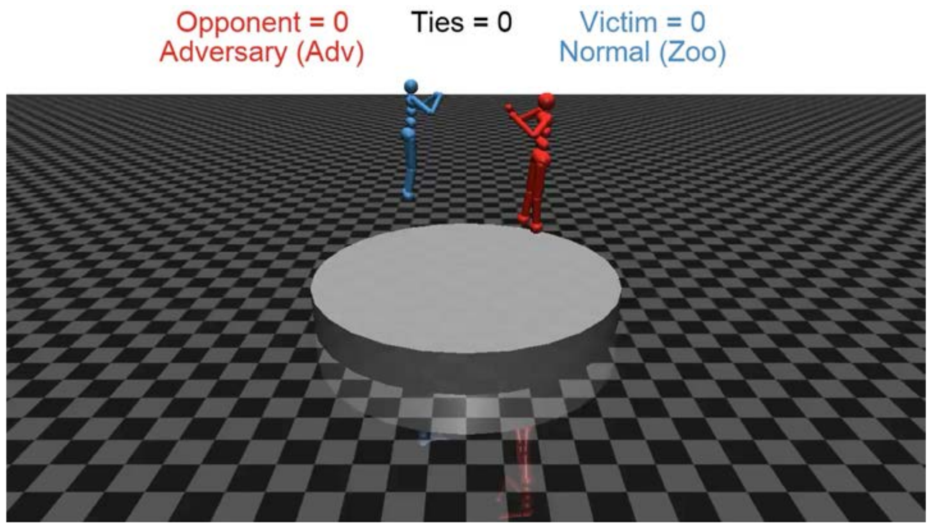


Kick-And-Defend and You-Shall-Not-Pass

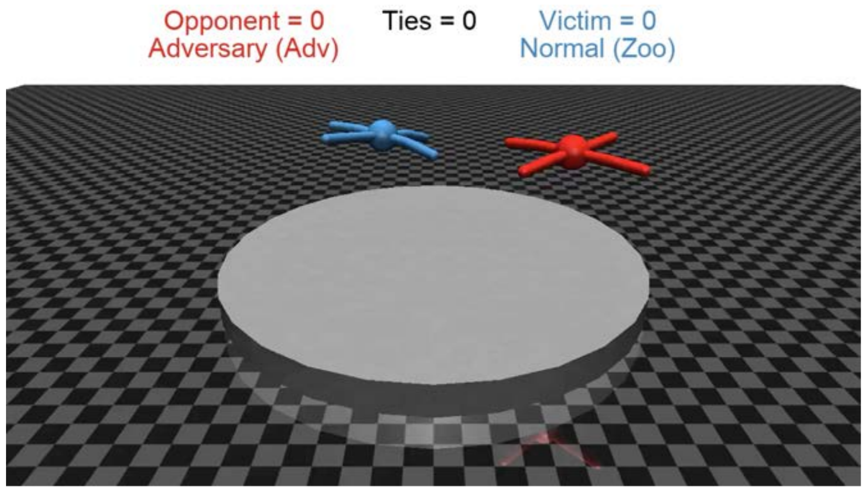
- Establishing weird behaviors that fail the victim.

Sumo-Humans

- Learn a better strategy – initializing itself near the boundary and luring the victim to fall from the arena.



Demo Examples



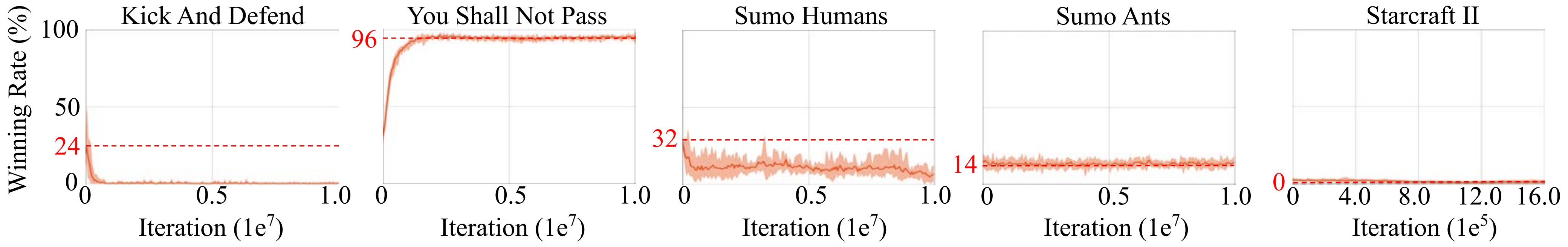
Sumo-ants

- Exploring the weakness of the game rule.
- If one player falls from the arena without touching its opponent, the game ends up with a draw.
- The adversarial agent (red one) intentional falls from the arena after the game begins.

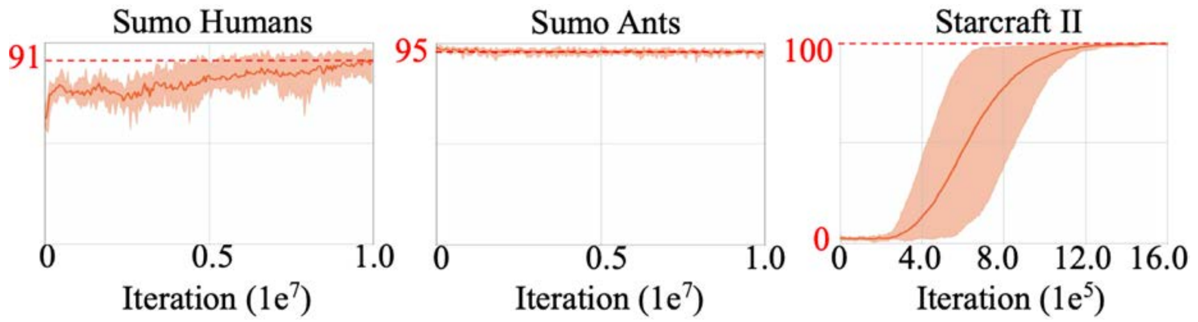
Video of the StarCraft II: https://tinyurl.com/ugun2m3

A Potential Defense

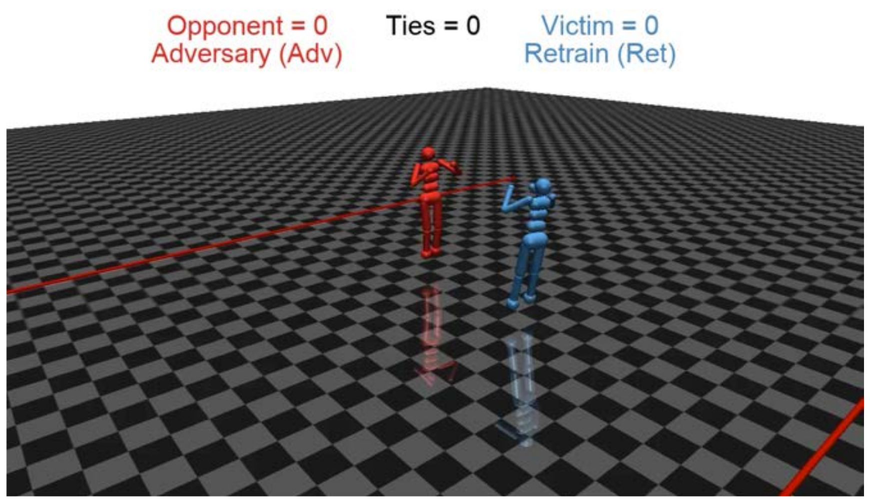
- Retraining the victim agent against the adversarial agent with our proposed attack.



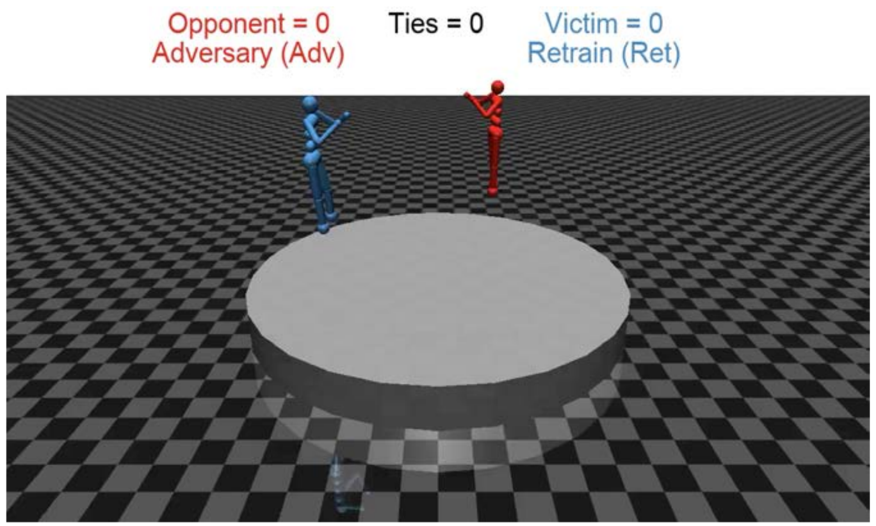
- Improving the performance of the victim.
  - Winning the You-Shall-Not-Pass.
  - Achieving a draw on three games.
- Kick-And-Defend – adversarial retraining does not work.
  - The unfairness of the game design – Its hard for the kicker to win.



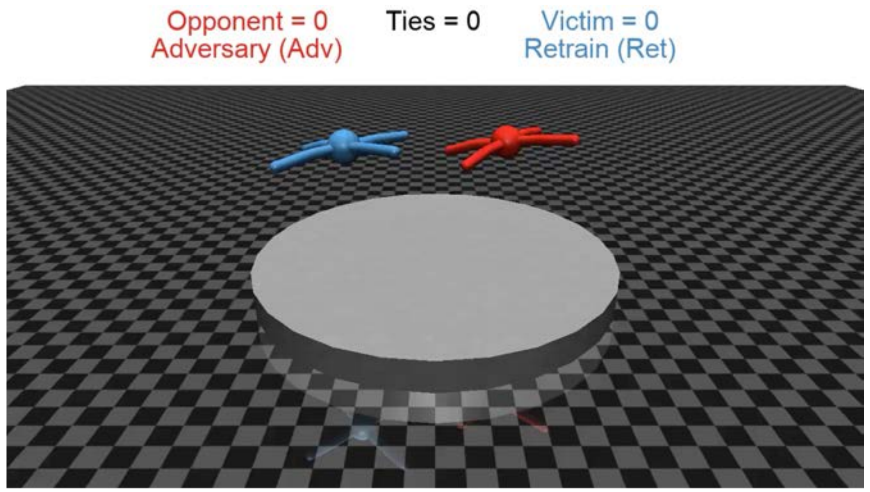
A Potential Defense



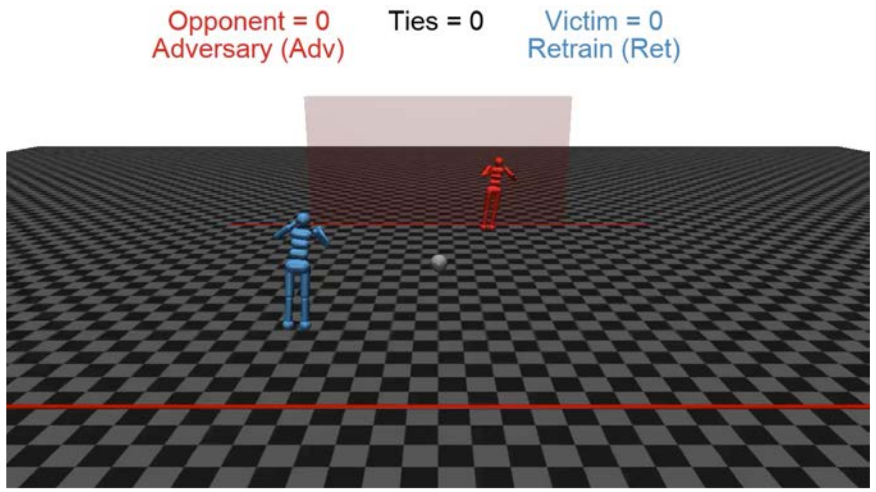
The victim learns to ignore the adversary and directly go for the finish line.



The victim recognizes the trick of the adversary and stay where it is.
Tie game!



Victim cannot change the intentional behaviors of the adversary.
Staying Tie games!



Victim acts ever worse.
Fall into the ground!

Agenda

- DRL basics.
  - Modeling an RL problem.
  - Training an DRL agent.
- DRL-powered games.
  - Two-agent games: MuJoCo, StarCraft II.
  - Code of training an DRL bot.
- Existing attacks on DRL.
  - Perturbation attacks (Extension of attacks on DNN)
  - Practical adversarial agent attacks.
- Our attack methodology.
- Evaluation.
- Conclusion.

Conclusion

- Attacker could train an adversarial agent to defect a bot of an AI-powered game.

- By disturbing the victim actions, the adversarial agent could exploit the vulnerabilities of the victim/game rules and thus fail the victim agent.

- Adversarial retraining does not always succeed; more advanced techniques are needed to protect the game bots (master agent).

black hat USA 2020
AUGUST 5-6, 2020
BRIEFINGS

Thank You !

Wenbo Guo

wzg13@ist.psu.edu   @Henrygwb   @WenboGuo4   http://www.personal.psu.edu/wzg13/