CS 7775

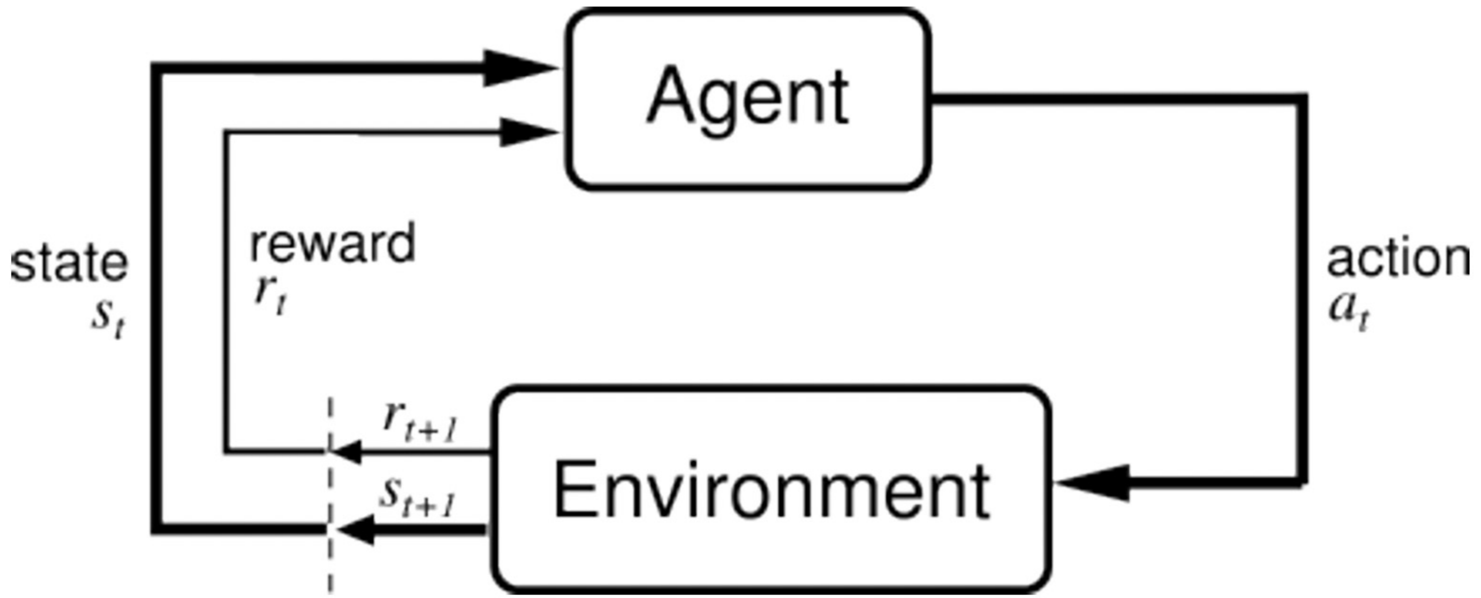
Seminar in Computer Security: Machine Learning Security and Privacy
Fall 2023

Alina Oprea
Associate Professor
Khoury College of Computer Science

October 12 2023

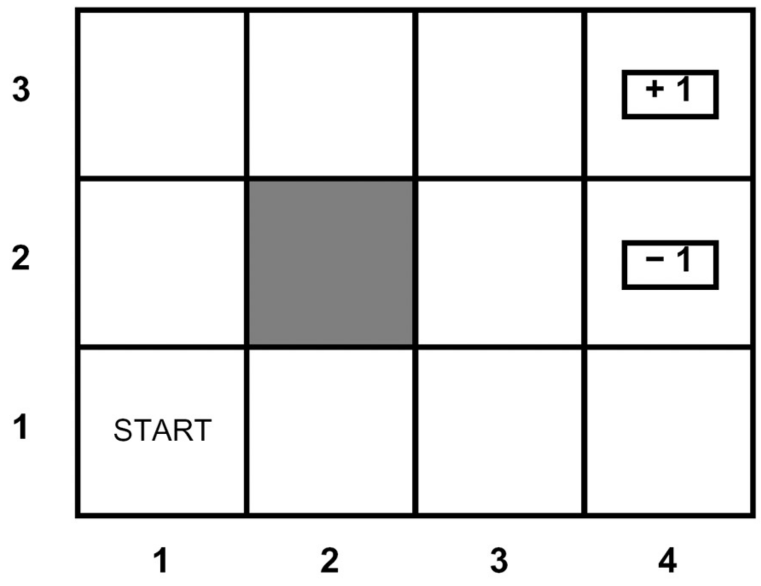
Reinforcement Learning

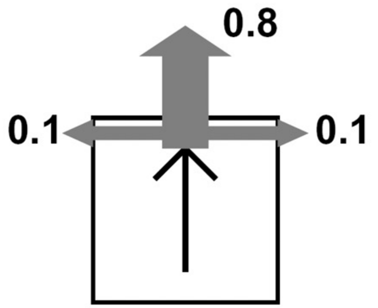
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards



Grid World

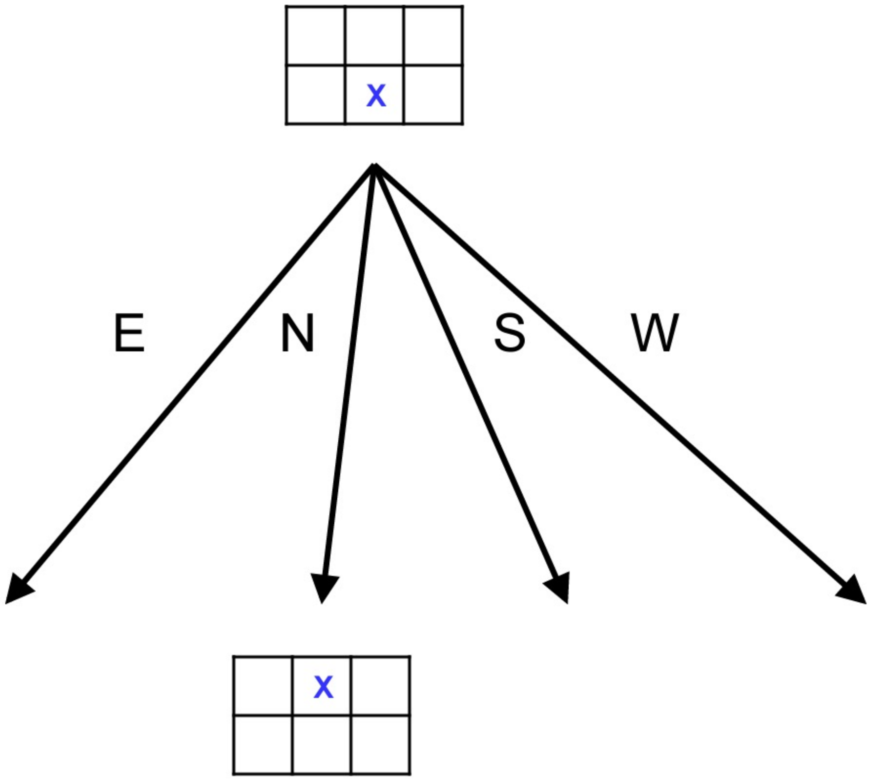
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards*



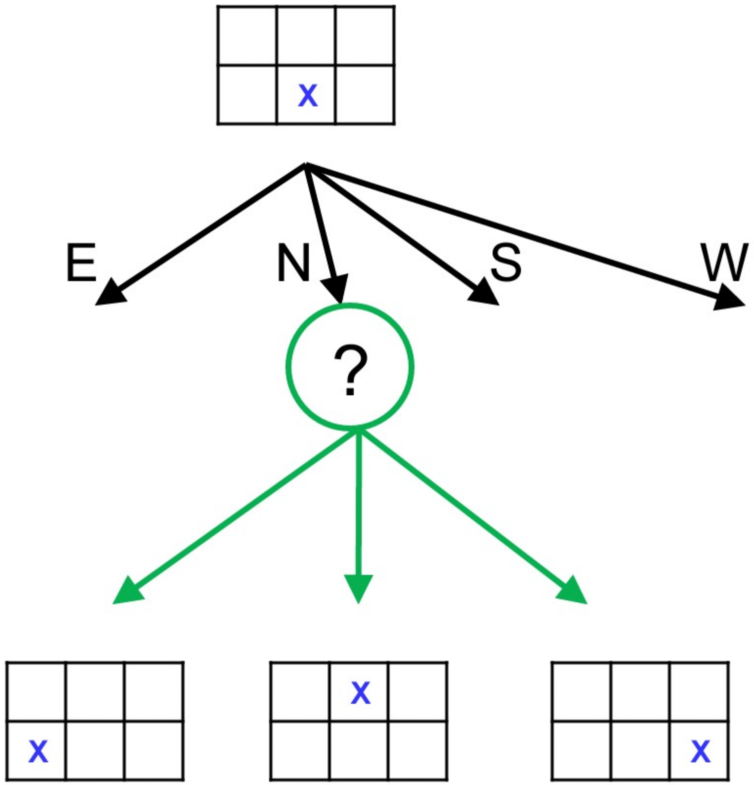


Deterministic vs Stochastic Actions

Deterministic Grid World



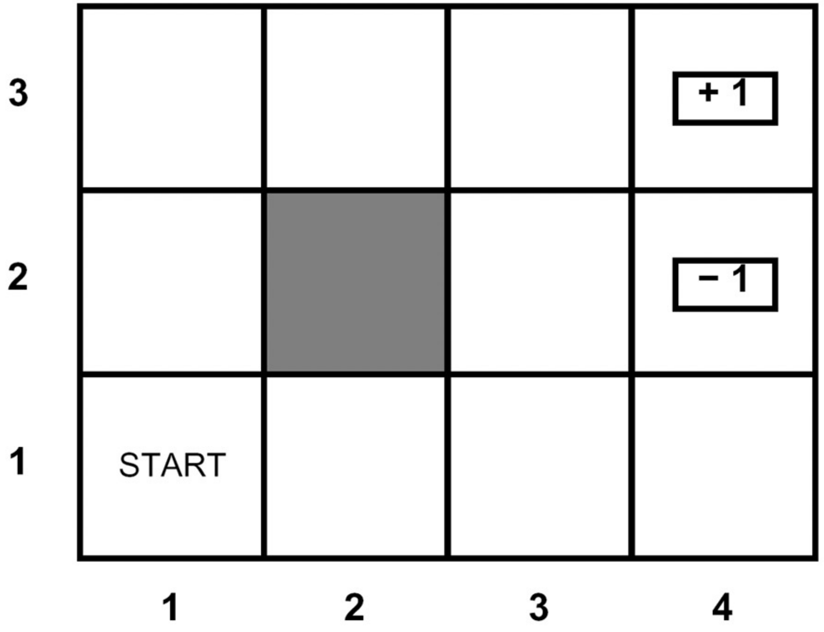
Stochastic Grid World

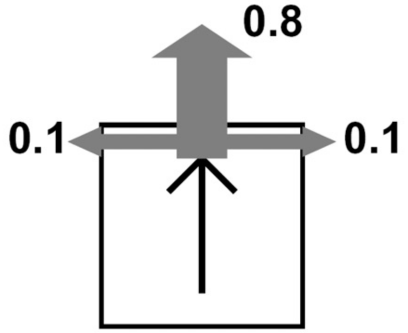


Markov Decision Processes (MDP)

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s,a,s’)
    - Prob that a from s leads to s’
    - i.e., P(s’ | s,a)
    - Also called the model
  - A reward function R(s, a, s’)
    - Sometimes just R(s) or R(s’)
  - A start state (or distribution)
  - Maybe a terminal state

- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don’t know the transition or reward functions





The Markov Property

- Andrey Markov (1856-1922)

- “Markov” generally means that given the present state, the future and the past are independent

- For Markov decision processes, “Markov” means:

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

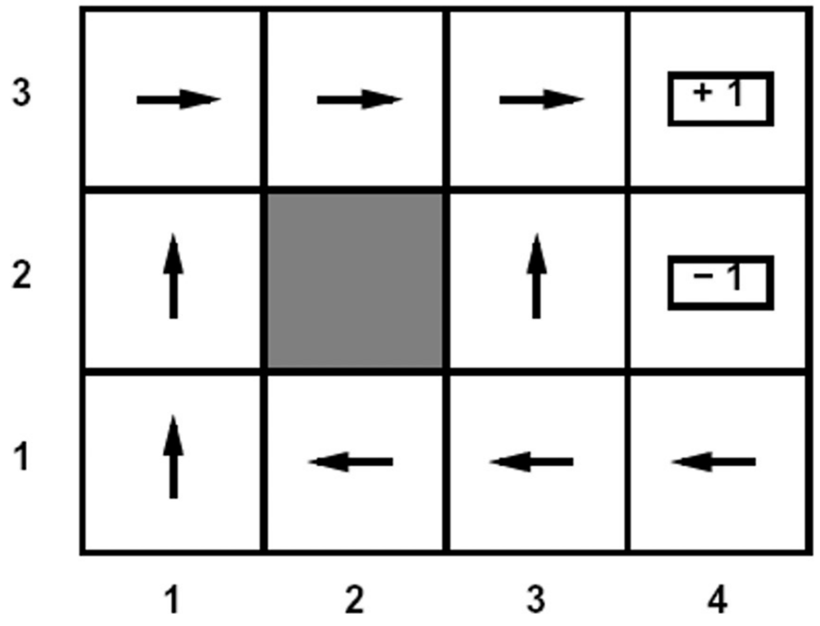
$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

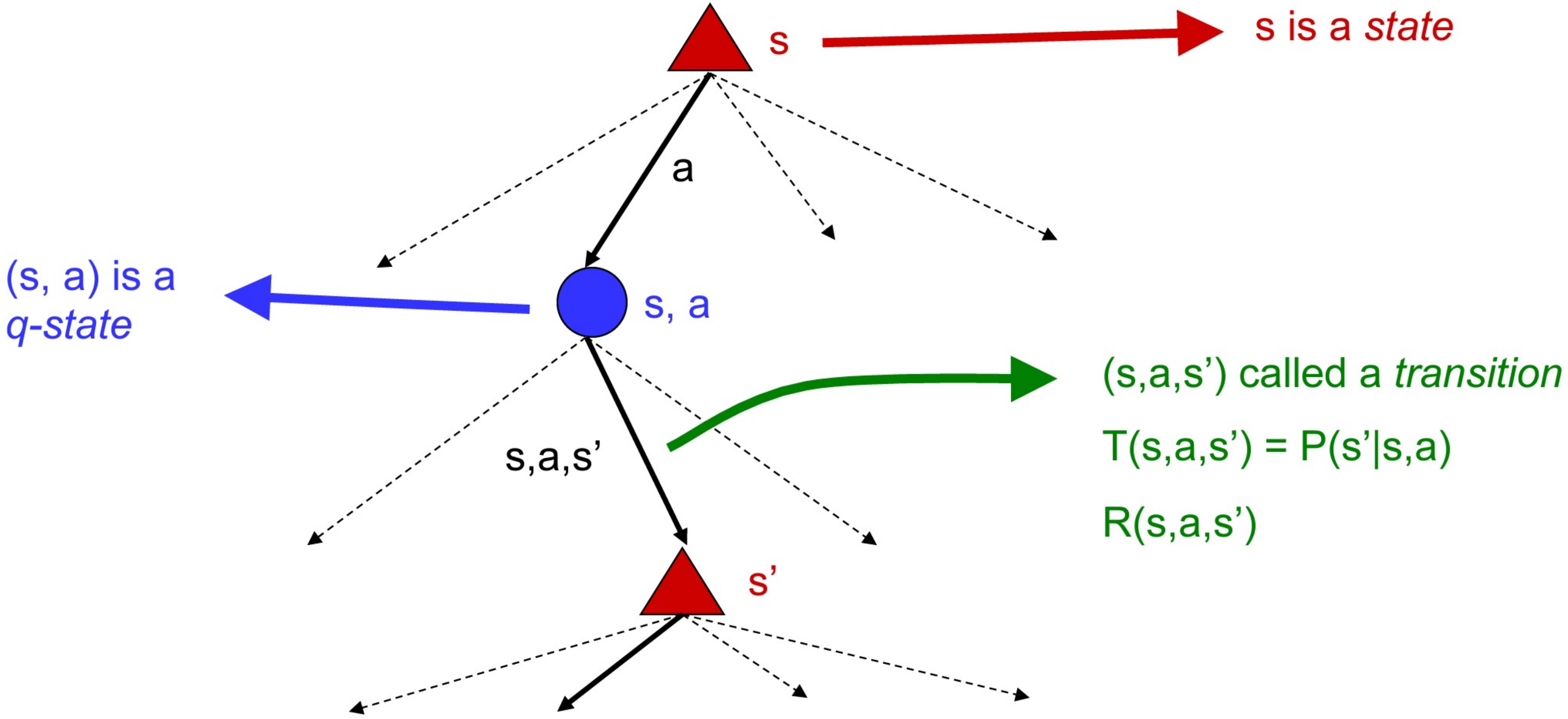
Optimal Policies

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

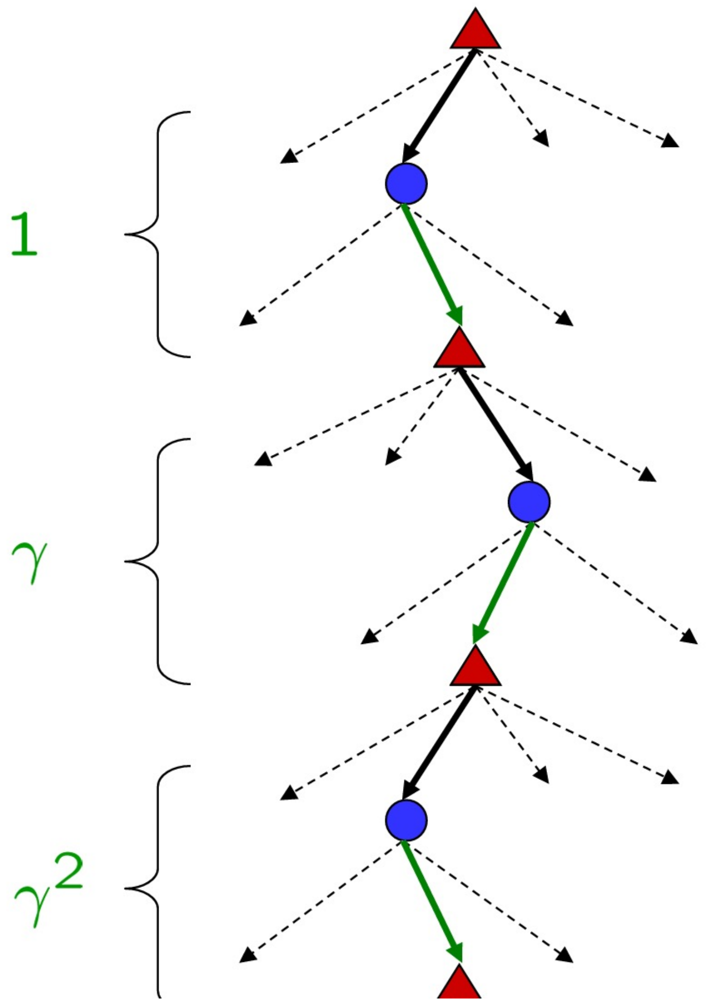


Maximize Reward



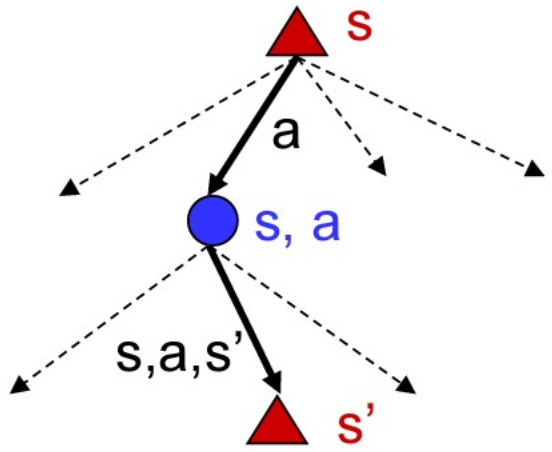
Discounting Rewards

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge



Recap: Defining MDPs

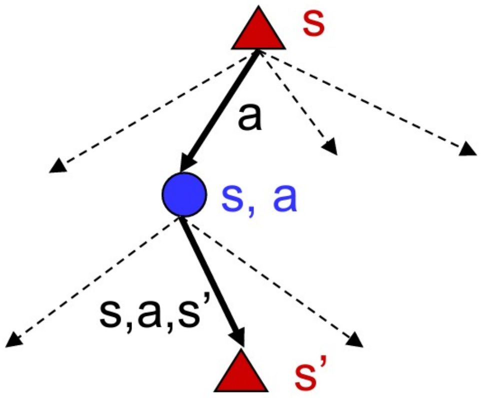
- Markov decision processes:
  - States S
  - Start state s_0
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)

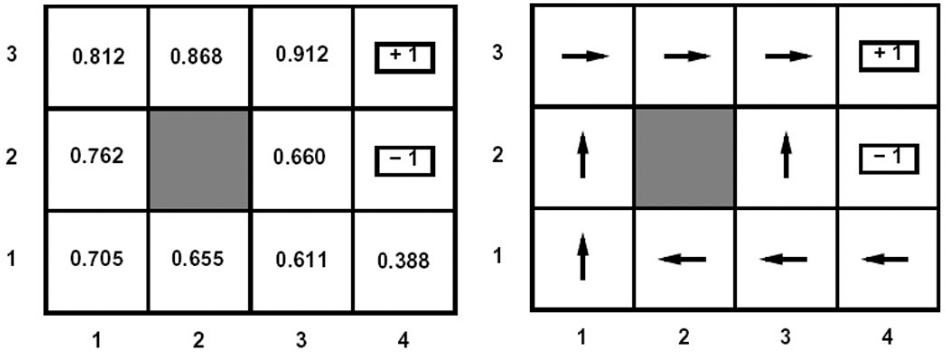


- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Optimal Utilities

- Fundamental operation: compute the values (optimal expectimax utilities) of states s
- Why? Optimal values define optimal policies!
- Define the value of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally
- Define the value of a q-state (s,a):
  $Q^*(s,a)$ = expected utility starting in s, taking action a and thereafter acting optimally
- Define the optimal policy:
  $\pi^*(s)$ = optimal action from state s

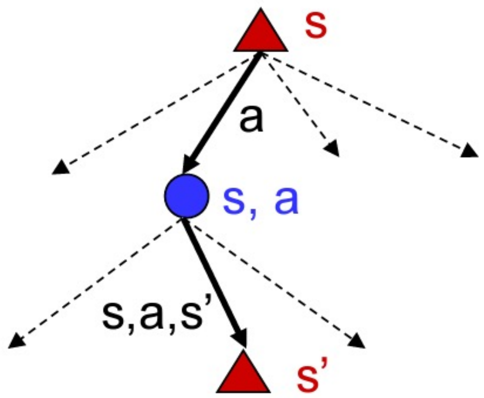




Bellman Equations

- Definition of "optimal utility" leads to a simple one-step lookahead relationship amongst optimal utility values:

  Optimal rewards = maximize over first action and then follow optimal policy



- Formally:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

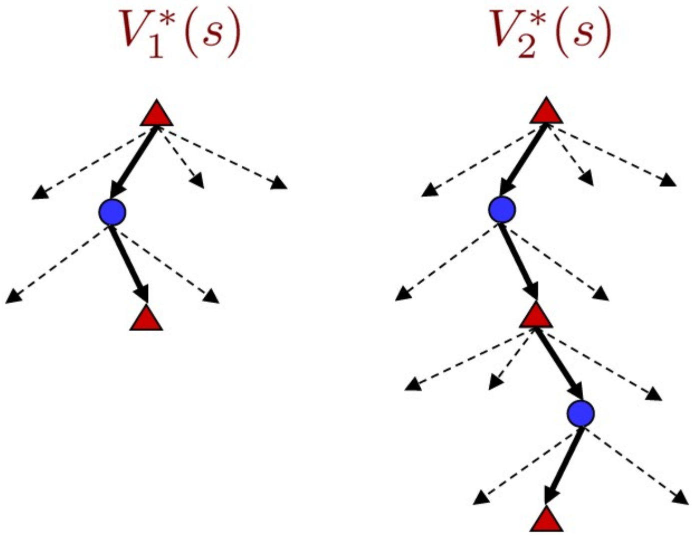
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

Dynamic Programming Methods

- Value Iteration
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

  - Slow, as it considers all actions in every iteration



- Policy Iteration
  - Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence (fast)
  - Step 2: Policy improvement: update policy using one-step lookahead with resulting converged (but not optimal!) utilities (slow but infrequent)
  - Repeat steps until policy converges

Q-Learning

- Q-Learning: sample-based Q-value iteration
- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]$$

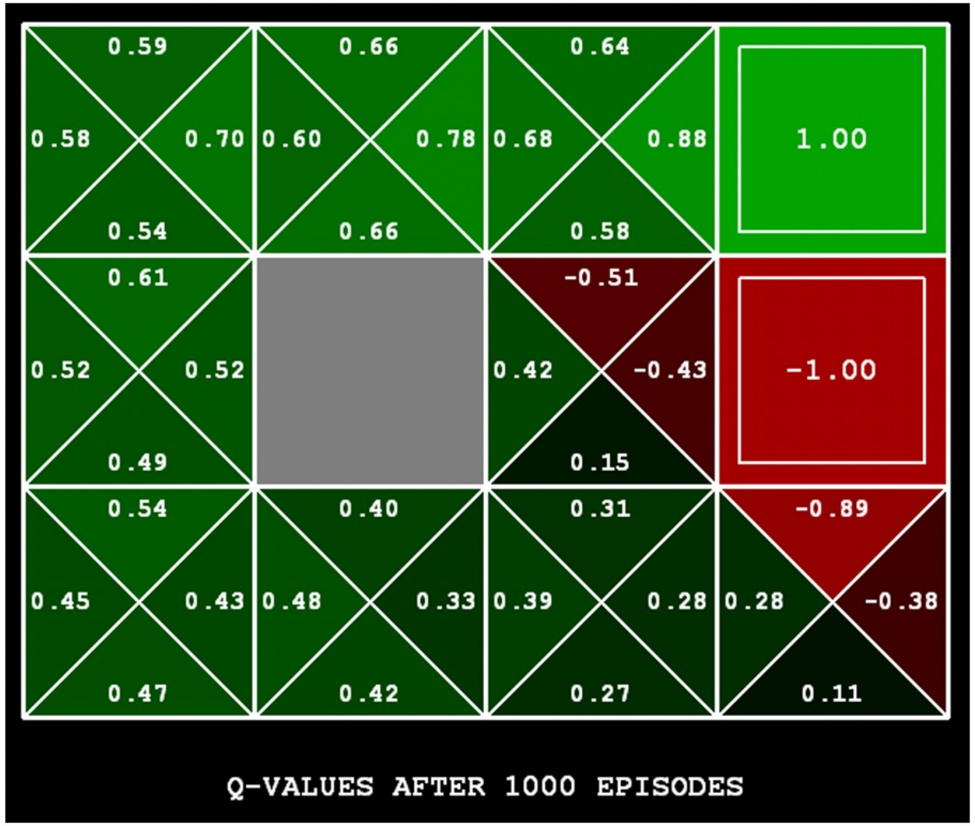
$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

Q-Learning

- Q-learning produces tables of q-values:



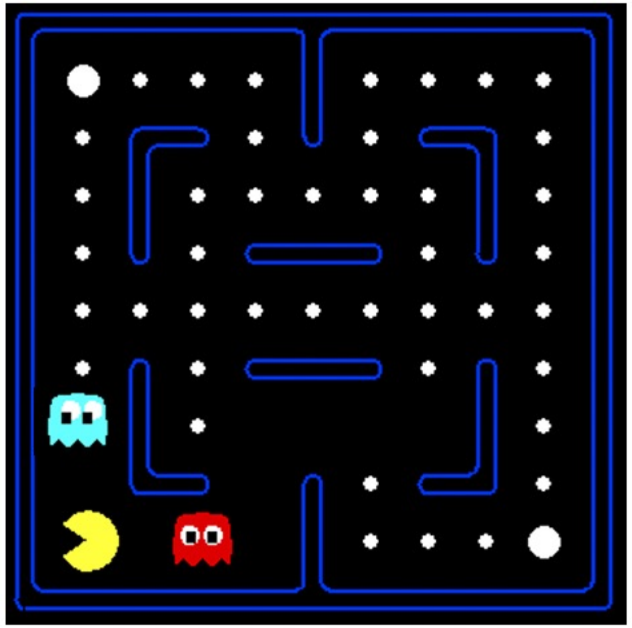
Q-Learning: Challenges

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

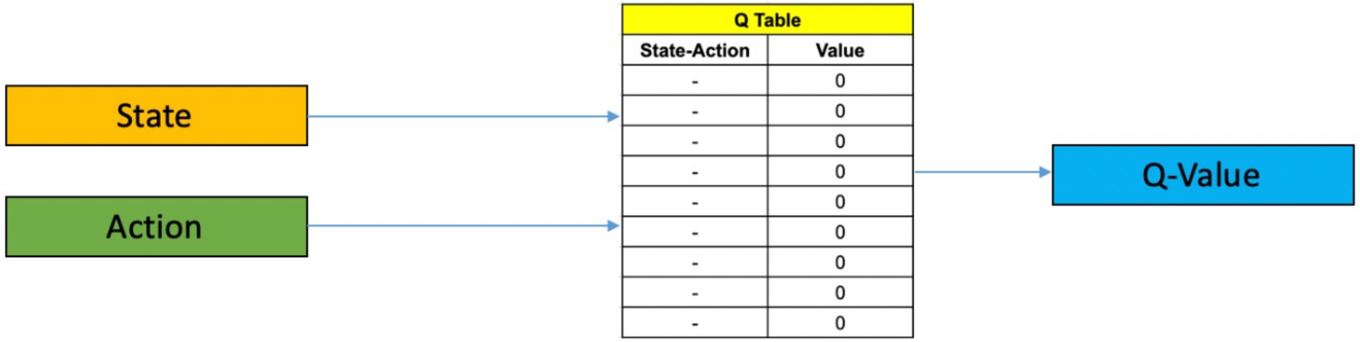
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Feature-Based Representations

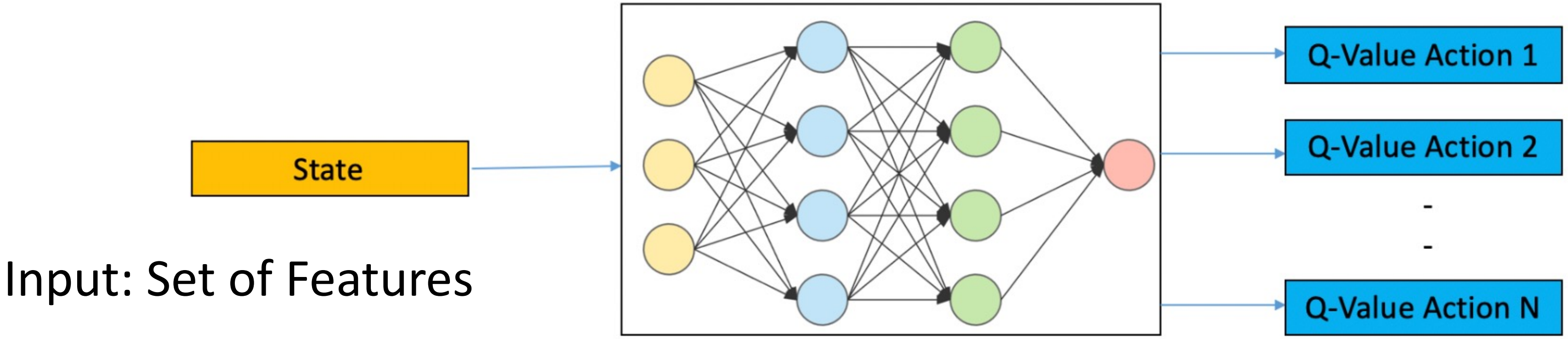
- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.



Deep Q Learning



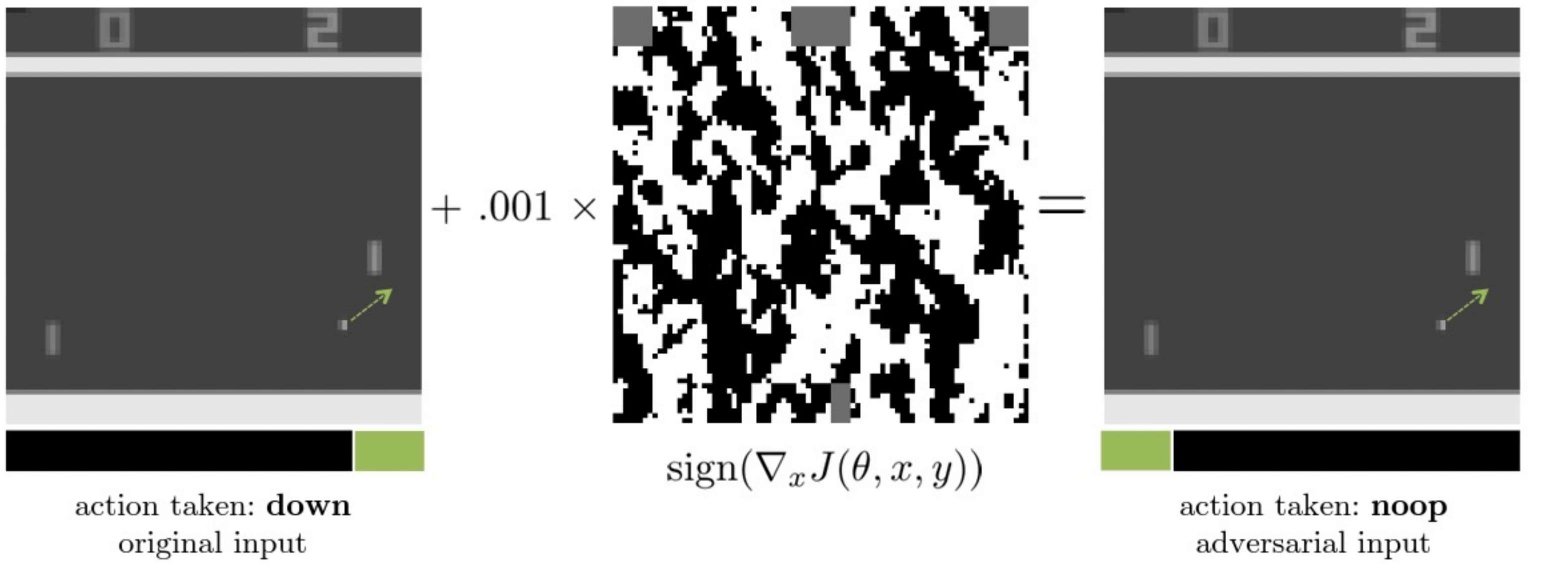
Q Learning



Deep Q Learning

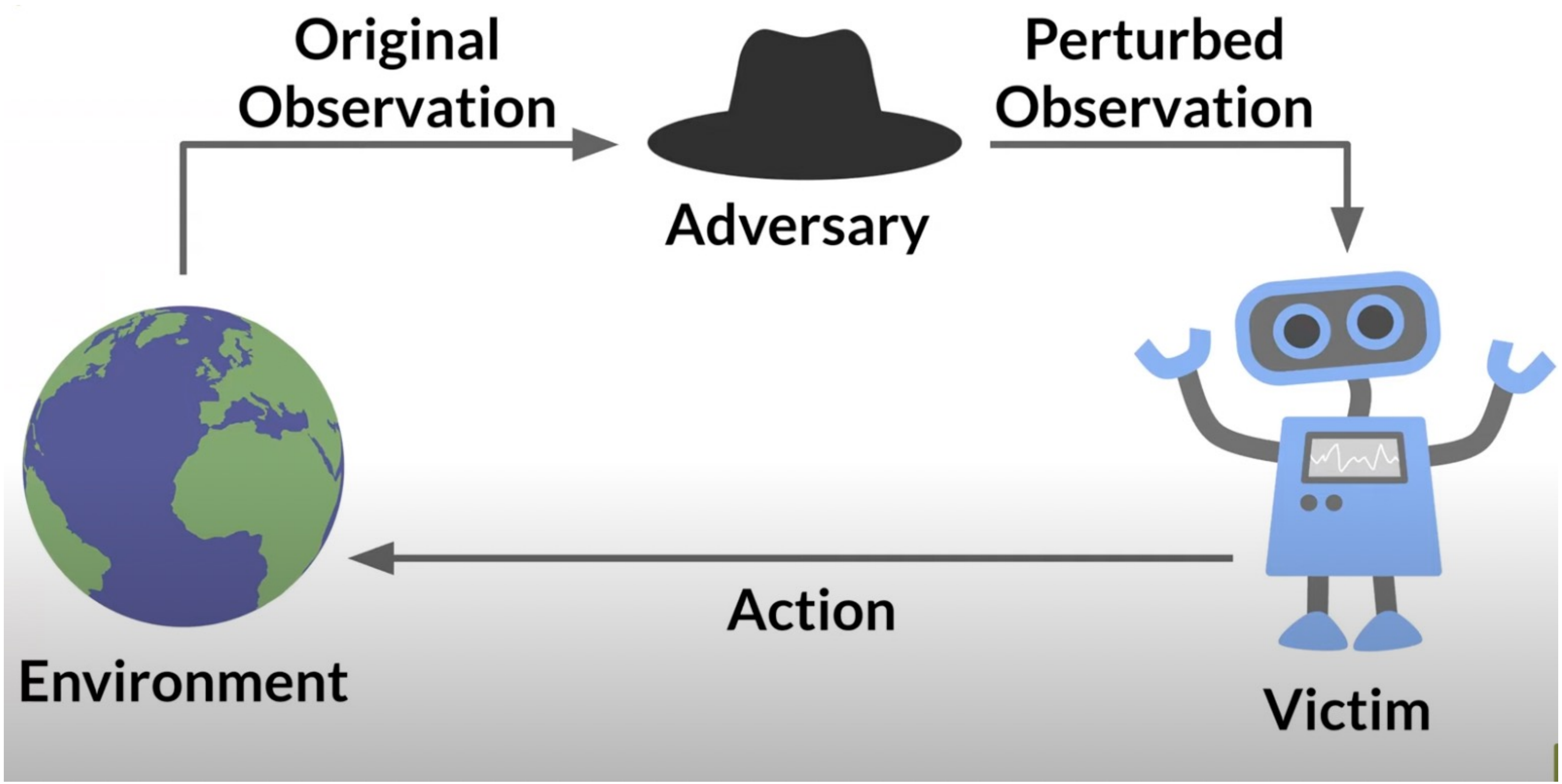

Attacking Deep Reinforcement Learning
Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, Stuart Russell
Victim
Normal
Opponent
Victim
Adversarial
Opponent

RL is Vulnerable to Adversarial Examples

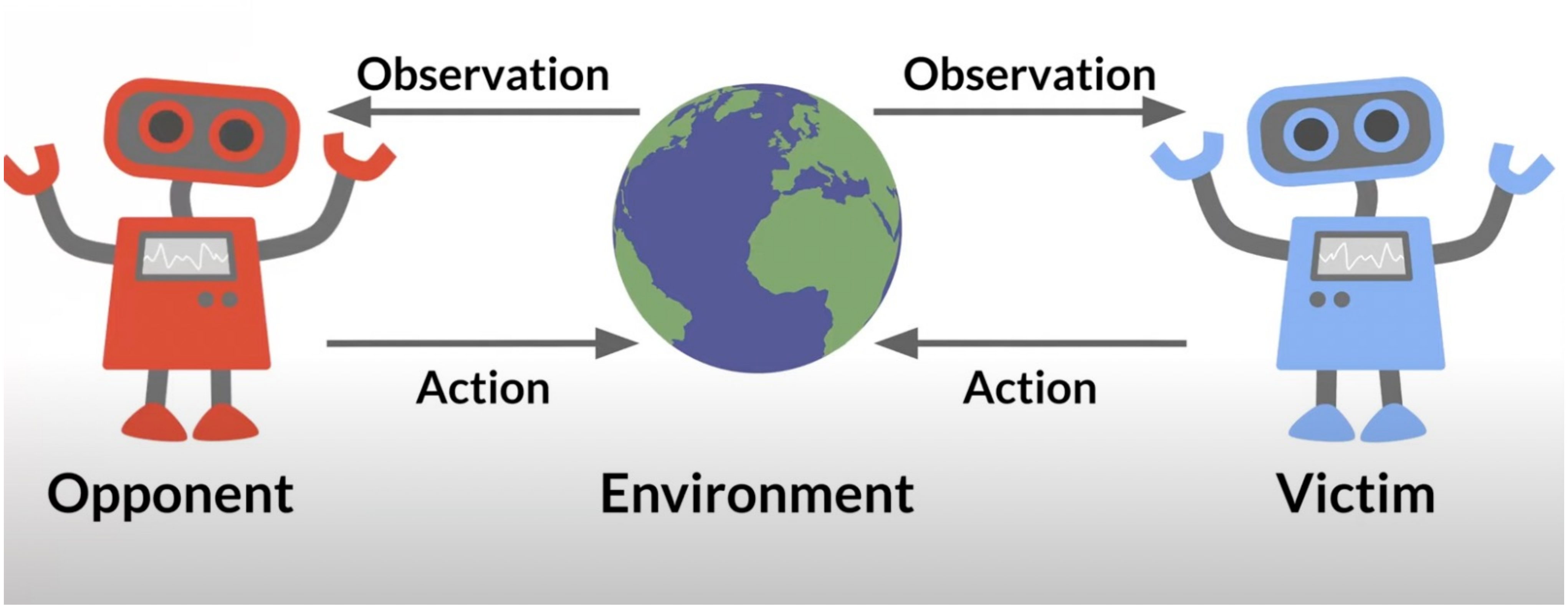


- Generate adversarial examples for Pong using Deep Q Learning
- Huang et al. Adversarial Attacks on Neural Network Policies. 2017

Threat Model in Prior Work



New Threat Model



- Opponent (adversary) can modify victim observations indirectly by performing certain actions
- Adversary takes the same set of actions as normal player (physically realizable)
- Victim policy is already trained (fixed)

Training the Adversary

Since the victim policy π_ν is held fixed, the two-player Markov game M reduces to a single-player MDP $M_\alpha = (S, A_\alpha, T_\alpha, R'_\alpha)$ that the attacker must solve. The state and action space of the adversary are the same as in M, while the transition and reward function have the victim policy π_ν embedded:

$$T_\alpha(s, a_\alpha) = T(s, a_\alpha, a_\nu) \qquad \text{and} \qquad R'_\alpha(s, a_\alpha, s') = R_\alpha(s, a_\alpha, a_\nu, s'),$$

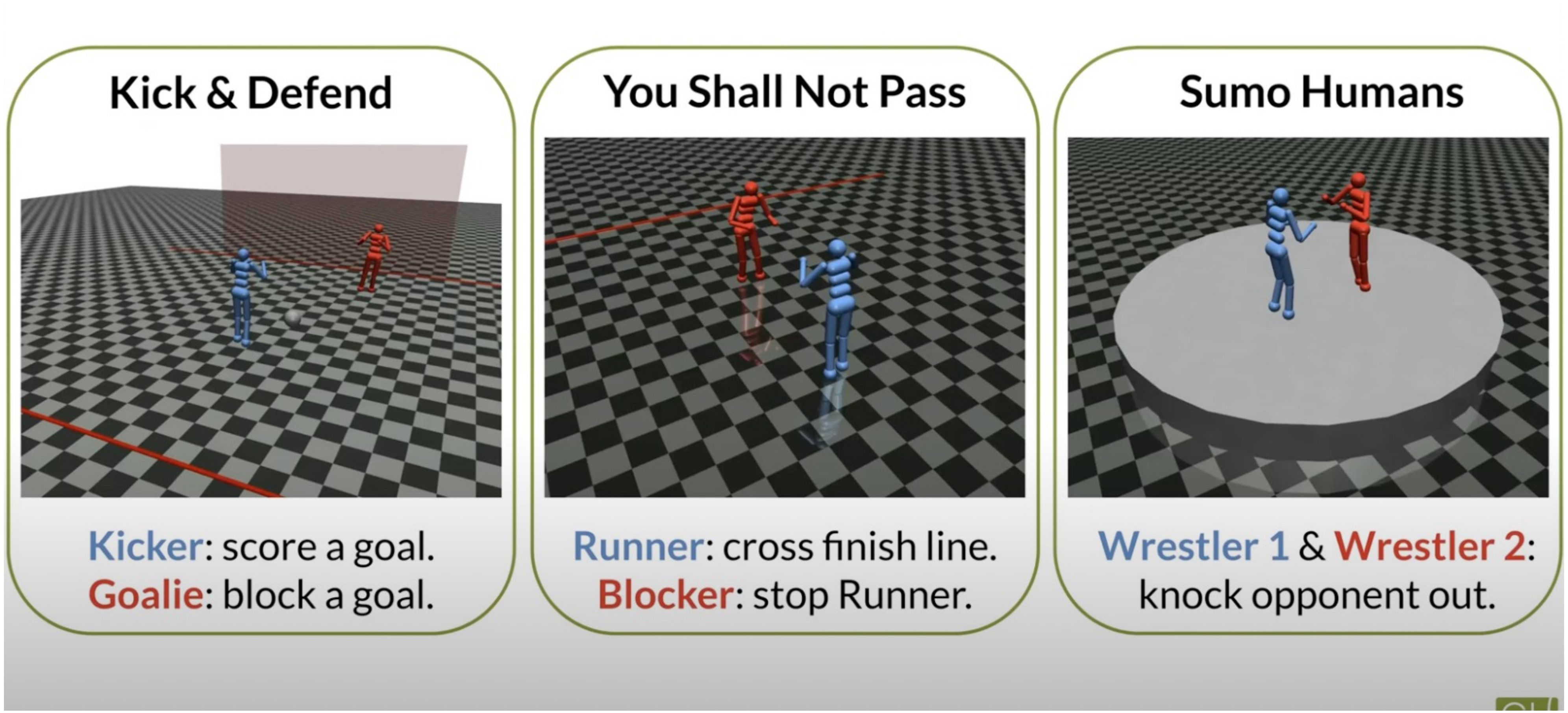
where the victim's action is sampled from the stochastic policy $a_\nu \sim \pi_\nu(\cdot \mid s)$. The goal of the attacker is to find an adversarial policy π_α maximizing the sum of discounted rewards:

$$\sum_{t=0}^{\infty} \gamma^t R_\alpha(s^{(t)}, a_\alpha^{(t)}, s^{(t+1)}), \quad \text{where } s^{(t+1)} \sim T_\alpha(s^{(t)}, a_\alpha^{(t)}) \text{ and } a_\alpha \sim \pi_\alpha(\cdot \mid s^{(t)}). \tag{1}$$

Note the MDP's dynamics T_α will be unknown even if the Markov game's dynamics T are known since the victim policy π_ν is a black-box. Consequently, the attacker must solve an RL problem.

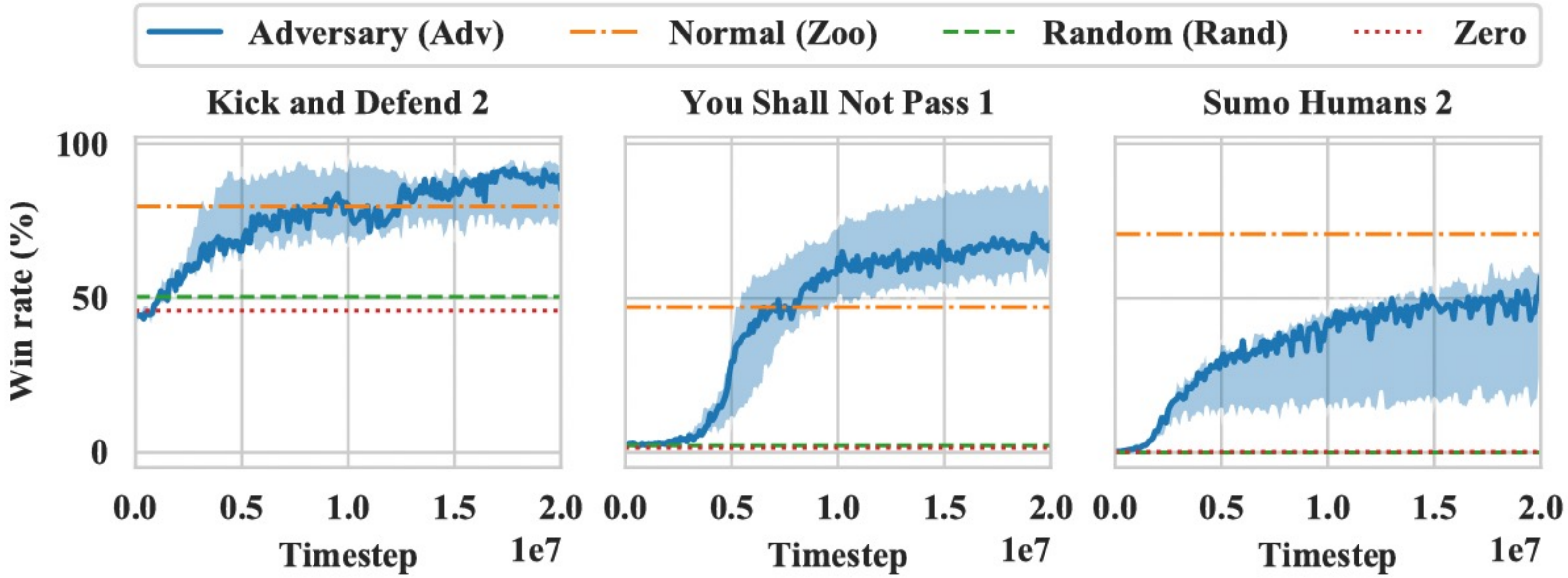
- Adversarial policy maximizes cumulative discounted reward
- Victim policy is embedded in environment
- Adversary only has black-box access (observes actions taken by victim)

Environments



- All are zero-sum games

Results against Fixed Victim



- Adversary trained for 3% of epochs relative to victim
- Adversary exploits weaknesses in victim policies (adversary does not stand up, kneel in the center)

Adversarial Moves

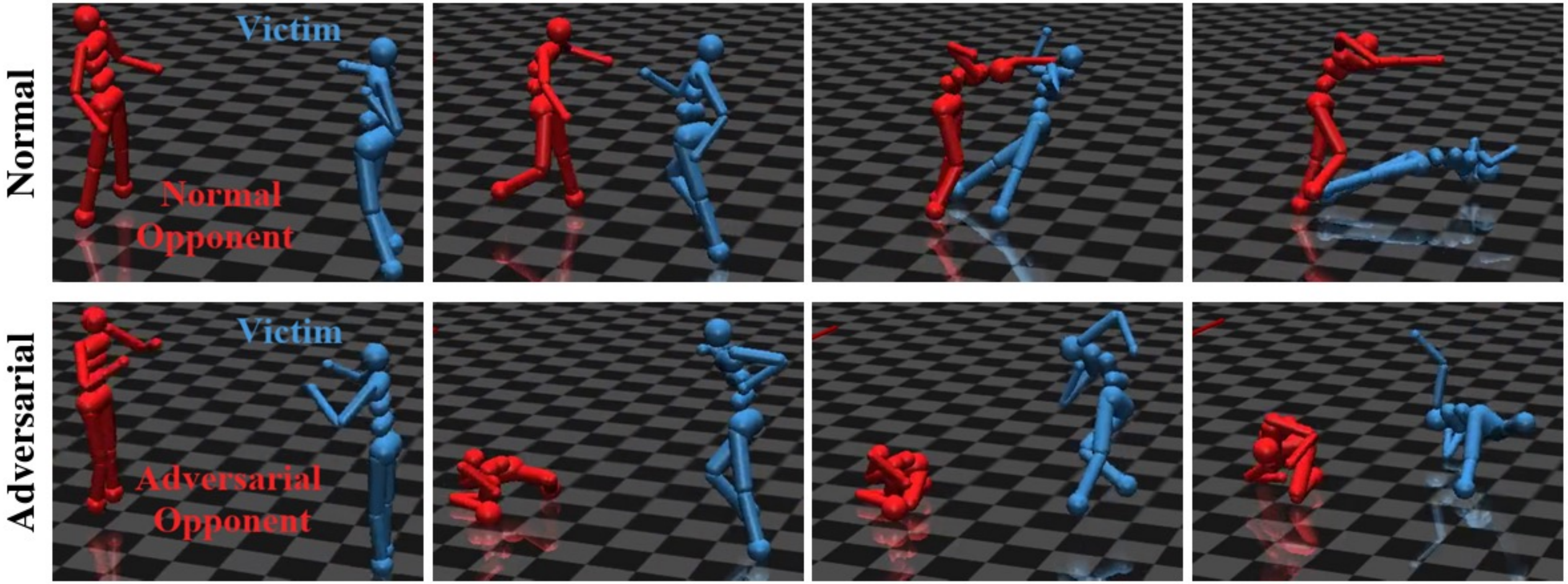
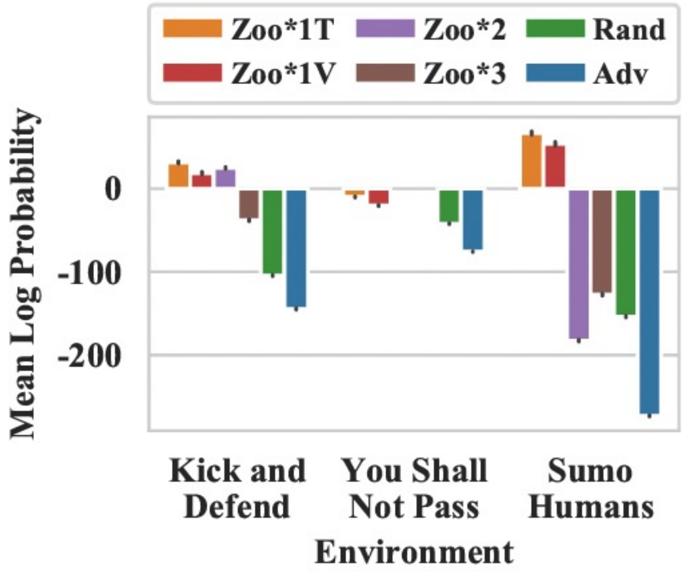
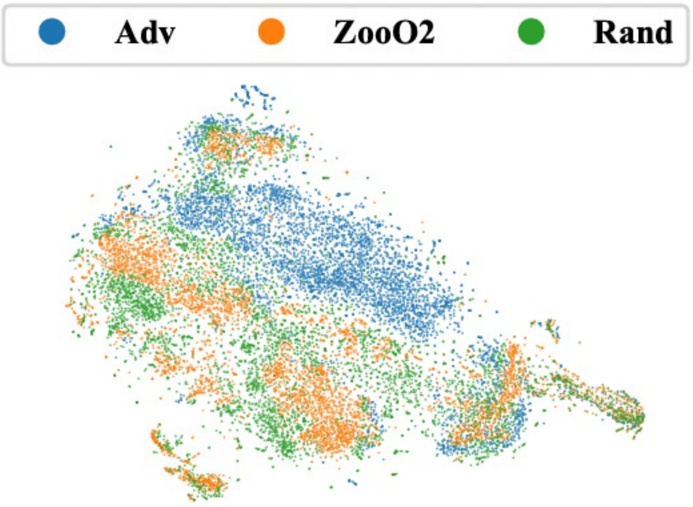


Figure 1: Illustrative snapshots of a victim (in blue) against normal and adversarial opponents (in red). The victim wins if it crosses the finish line; otherwise, the opponent wins. Despite never standing up, the adversarial opponent wins 86% of episodes, far above the normal opponent's 47% win rate.

Density Modeling

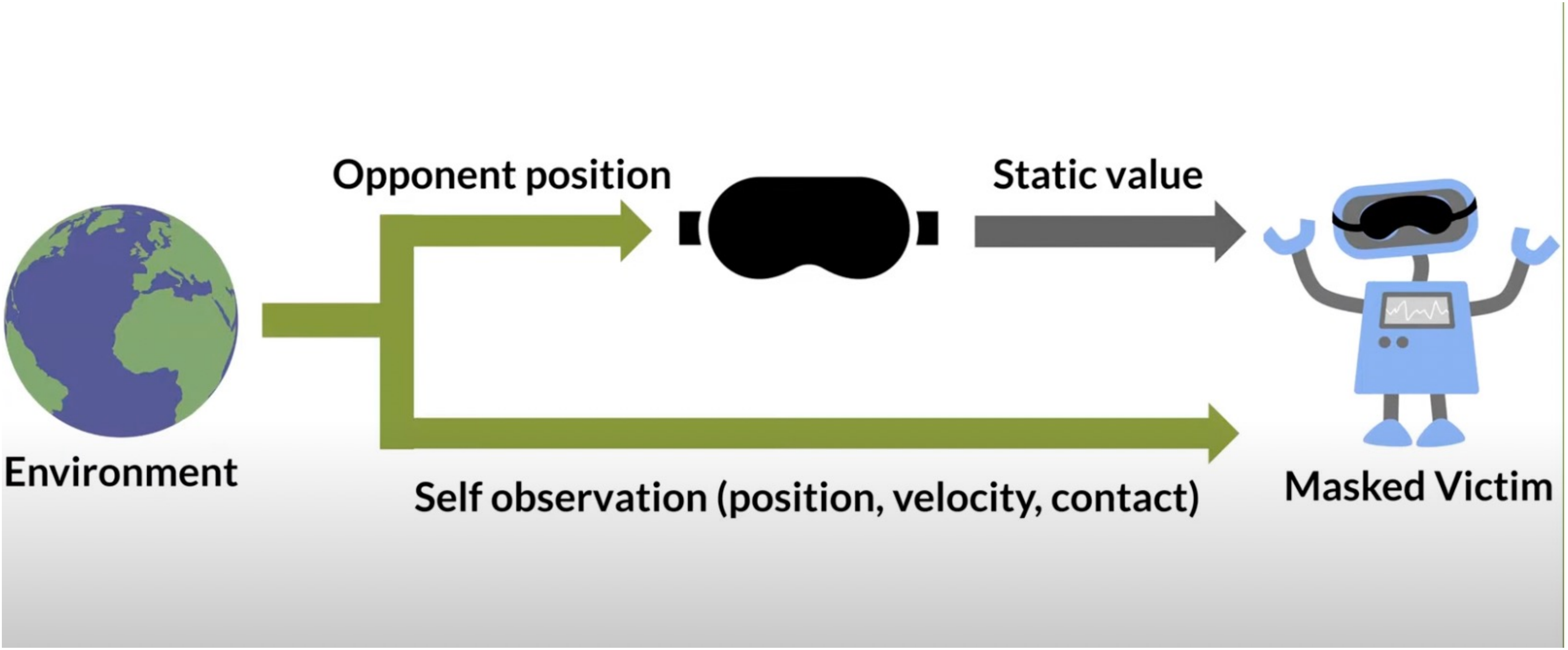




- Fit GMM model on activations and compute log likelihoods
- Adversarial activations at all layers are very unlikely
- But random activations are also unlikely

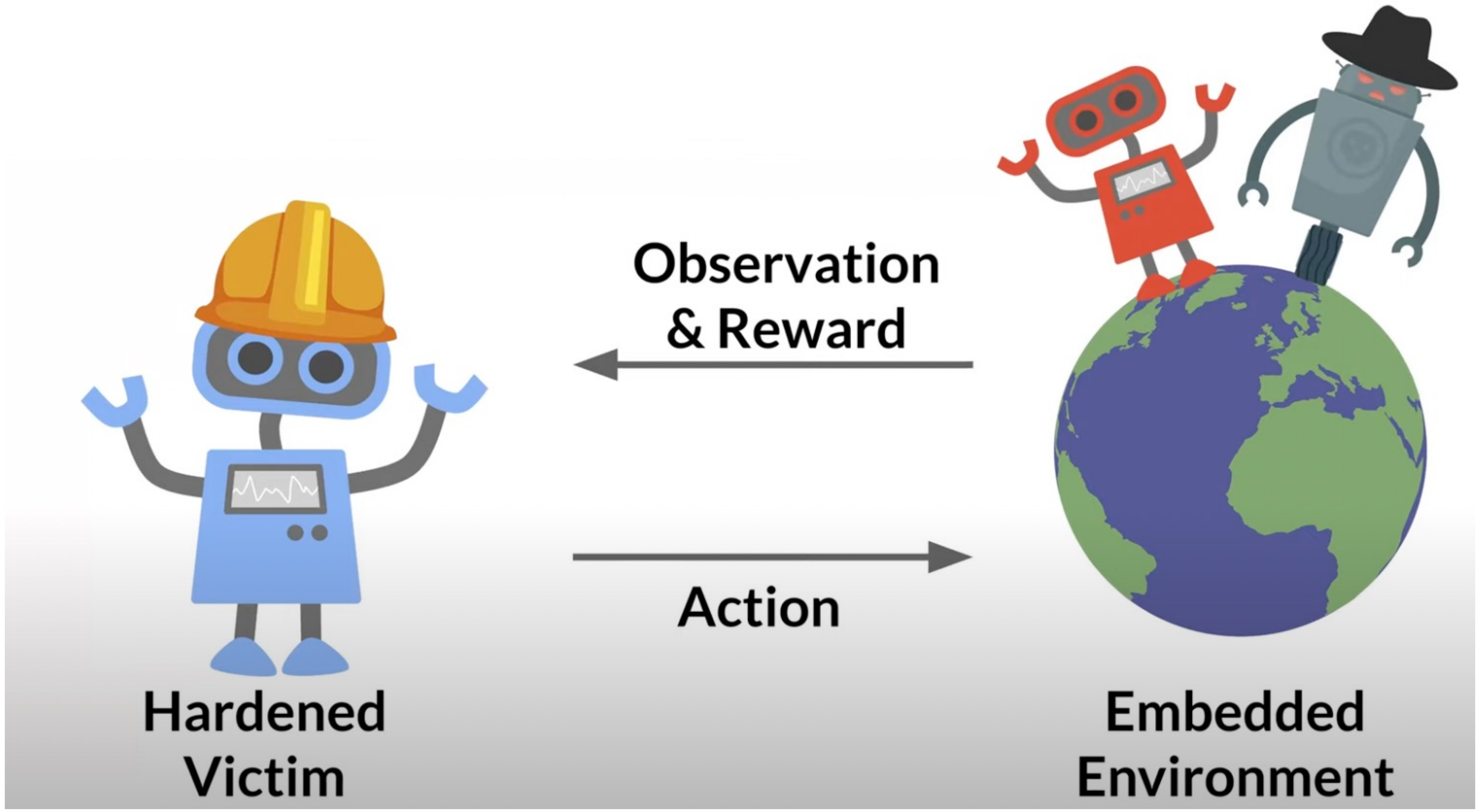
- Adversarial activations cluster together, while random are more diverse

Defense 1: Masking



- Hide the victim position to the adversary
- It works, but it hurts performance against normal opponent

Defense 2: Adversarial Training



- This works and achieves good performance against normal players
- But it fails against adaptive attack!

Strengths

- One of the first attacks against RL
  - Using RL against RL
- Realistic threat model for RL
  - Adversarial player takes moves in the game
  - Does not require directly manipulating state observations

Limitations

- Initial attack assumes victim policy is static
- The adversarial policies are not meaningful
  - Could use a term in the reward to optimize against normal opponent and victim
- None of the defenses is satisfactory
- Game environments are all similar

Acknowledgements

- Slides made using resources from
  - Eric Eaton, Adam Gleave

- Thanks!