# Reinforcement Learning

Jie Tang

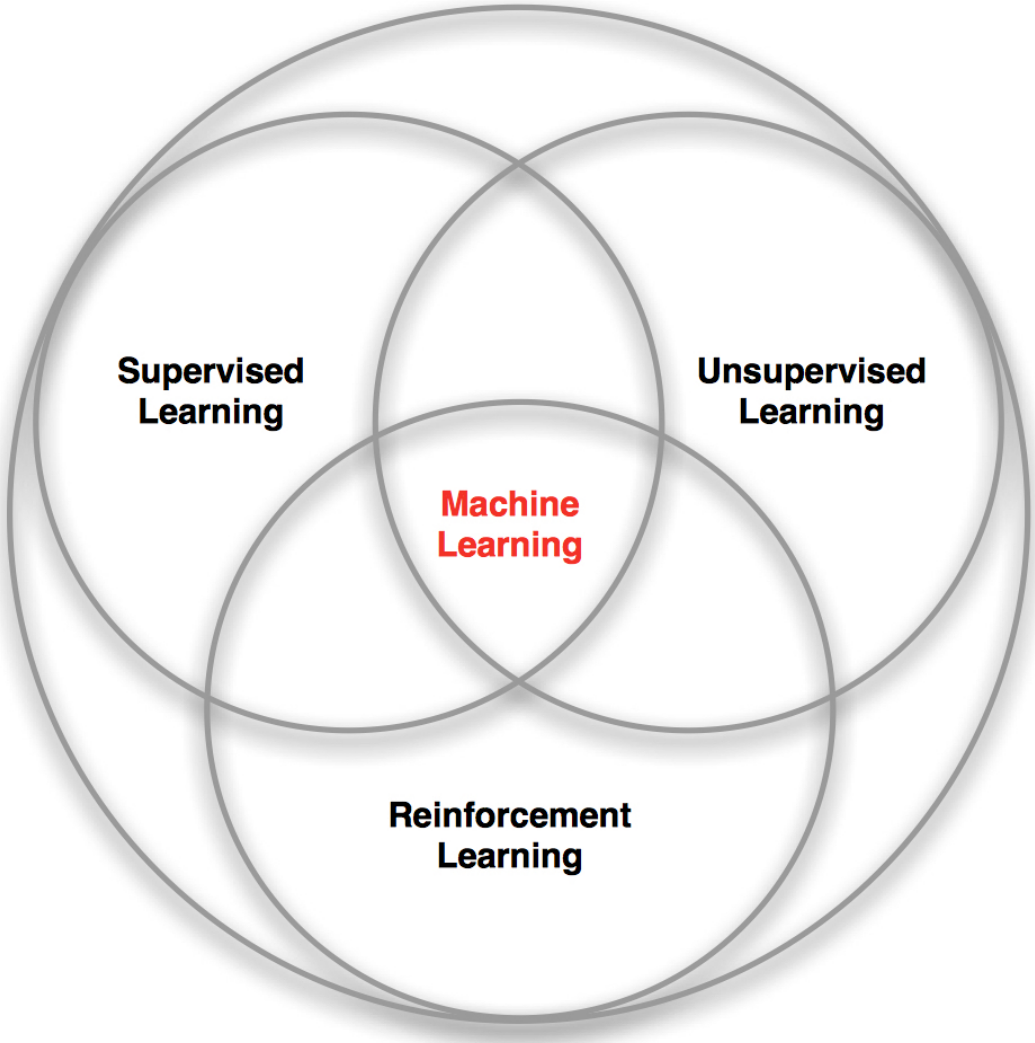
Tsinghua University

May 20, 2020

# Overview

# Outline

# Branches of Machine Learning

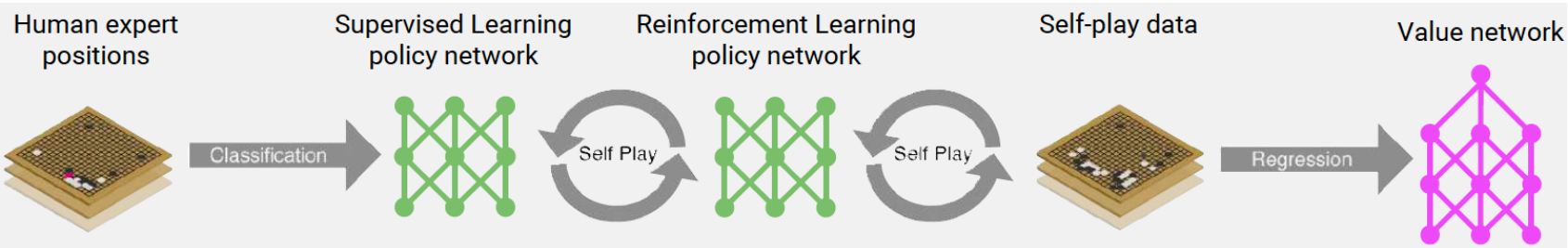


# Characteristics of Reinforcement Learning

- RL vs. supervised learning
  - without labels. There is no supervisor, only a reward signal
  - impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act.
  - trial-and-error. An agent must be able to learn from its own experience. Agent's actions affect the subsequent data it receives
  - delayed reward
- RL vs. unsupervised learning
  - maximize a reward signal instead of trying to find hidden structure
  - repeated interactions with the world

# AlphaGo/AlphaGo Zero

- AlphaGo is the first computer program to defeat a professional human Go player and a Go world champion.
- Knowledge learned from expert games and self-play
- Monte-Carlo search guided by policy and value networks

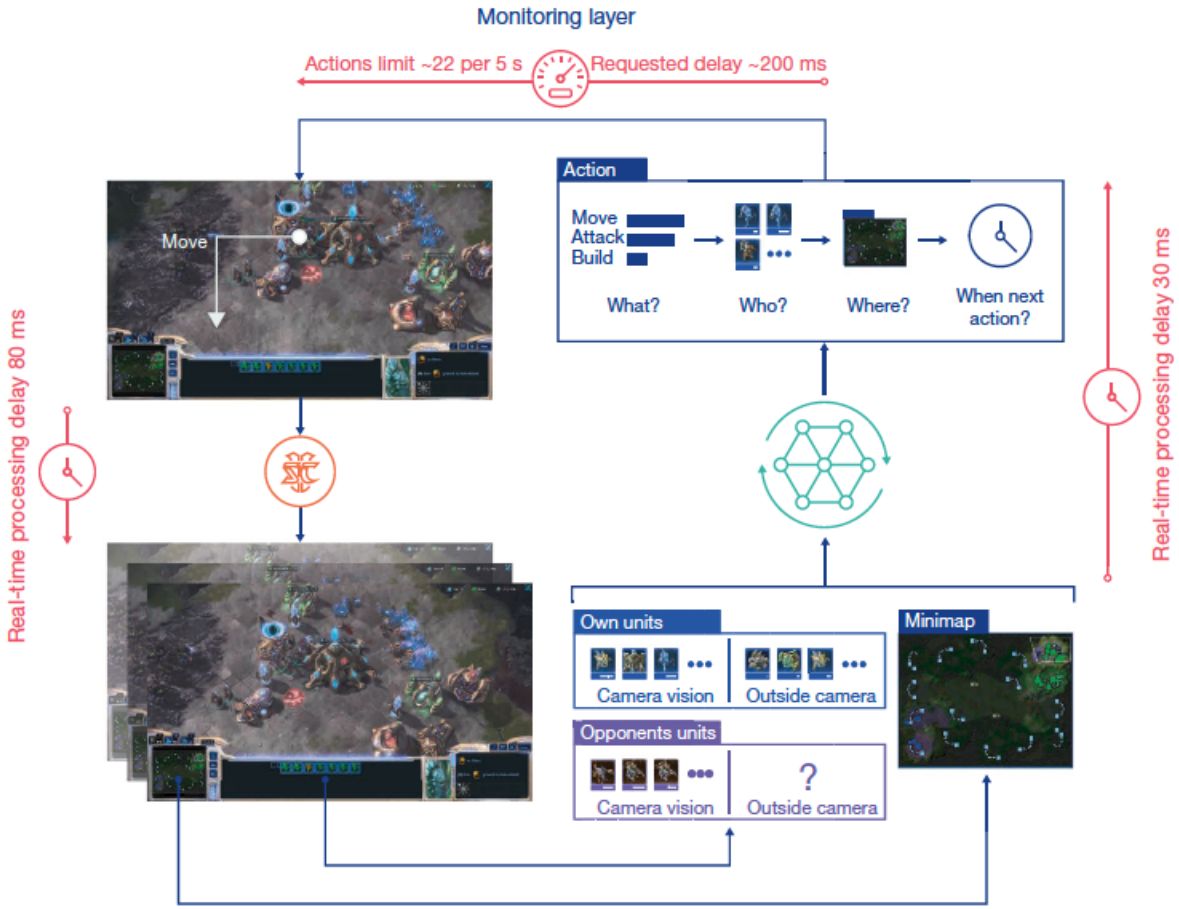



- AlphaGo Zero: without human knowledge. AlphaGo becomes its own teacher.
  - Policy is improved by AlphaGo search
  - Policy is evaluated according to outcome of AlphaGo vs AlphaGo games

[Mastering the game of Go with deep neural networks and tree search, Nature, January 2016] [Mastering the game of Go without human knowledge, Nature, October 2017]

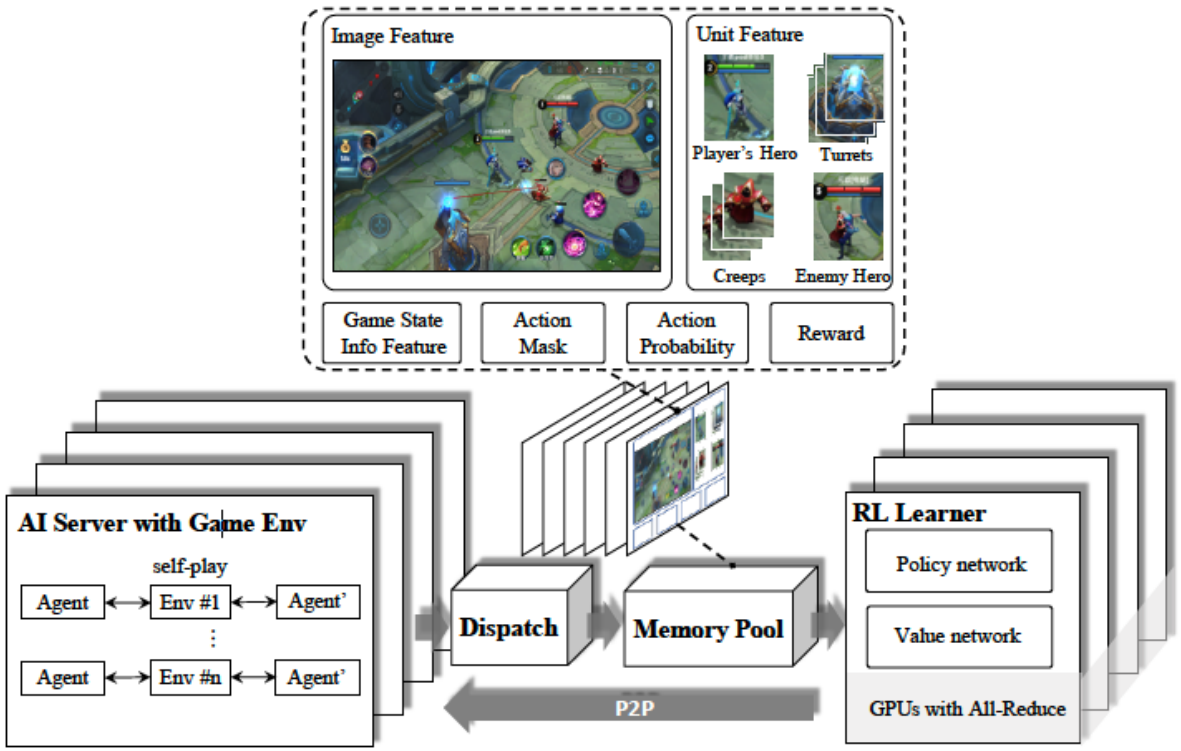
# AlphaStar

- **Challenges**. Games of incomplete information; sparse rewards; limit of action per minute (APM)
- **Solutions**. Multi-agent reinforcement learning: three distinct types of agent (main agents, main exploiters, league exploiters). Prioritized fictitious self-play (PFSP)



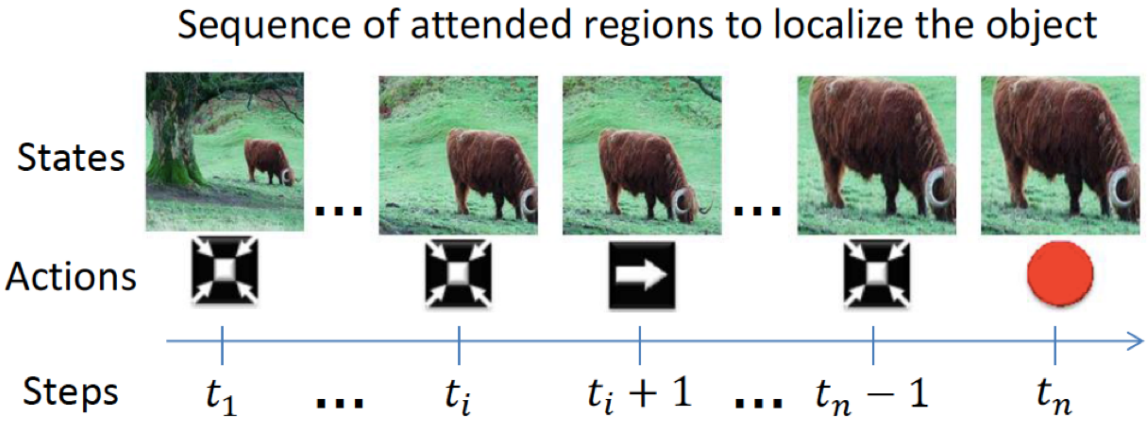
# 绝悟

- Novel strategies: control dependency decoupling, action mask, target attention, and dual clip proximal policy algorithm (PPO)
- Can defeat top professional human players in full 1v1 games of Honor of Kings （王者荣耀） .



# Wide Applications in NLP and CV...

- Active object localization with deep reinforcement learning




[J. Caicedo and S. Lazebnik, ICCV 2015]

Atari, AlphaGo/AlphaGo Zero,
AlphaZero,
AlphaFold,
AlphaStar



OpenAI Five

Volodymyr Mnih
DeepMind

David Silver
DeepMind

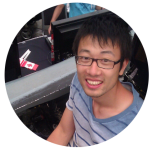
Double DQN

deterministic policy gradients (DPG)

DQN

prioritized replay

deep DPG (DDPG)



Marc G. Bellemare
DeepMind

Duelling DQN

**A lot ...**

asynchronous advantage actor-critic (A3C)

Marc G. Bellemare
DeepMind

Categorical DQN

A2C

noise DQN

Rainbow

Ape-X

Meire Fortunato
DeepMind

JANE X. WANG
DeepMind

# Problem Statement


- Trajectory/episode: $s_1, a_1, r_1, s_2, a_2, r_2, \ldots$
- Policy: $\pi(a \mid s) : S \to A$
- Objective: choose $a_t$ to maximize return

$$G_t := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

- Compare with Online Learning
  - No contexts $\Rightarrow$ multi-arm bandits
  - contexts do not depend on past actions $\Rightarrow$ contextual bandits
  - contexts/state depends on past actions $\Rightarrow$ reinforcement learning

# Major Components of an RL Agent

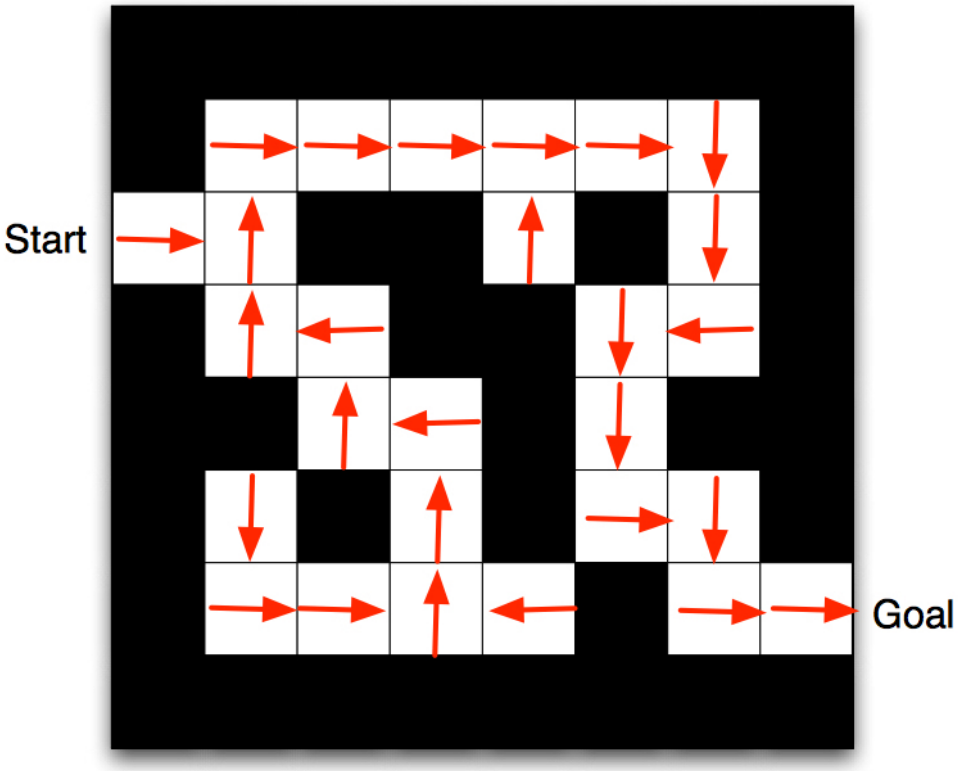
- An RL agent may include one or more of these components:
  - Policy: agent's behavior function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment
- Policy $\pi : S \rightarrow A$, mapping from states to actions
  - Deterministic policy:

$$\pi(s) = a$$

  - Stochastic policy:

$$\pi(a|s) = \Pr(a_t = a|s_t = s)$$

# Maze Example: Policy




- Arrows represent policy $\pi(s)$ for each state $s$

# Value Function

- Return $G_t$: the total discounted reward from time-step $t$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

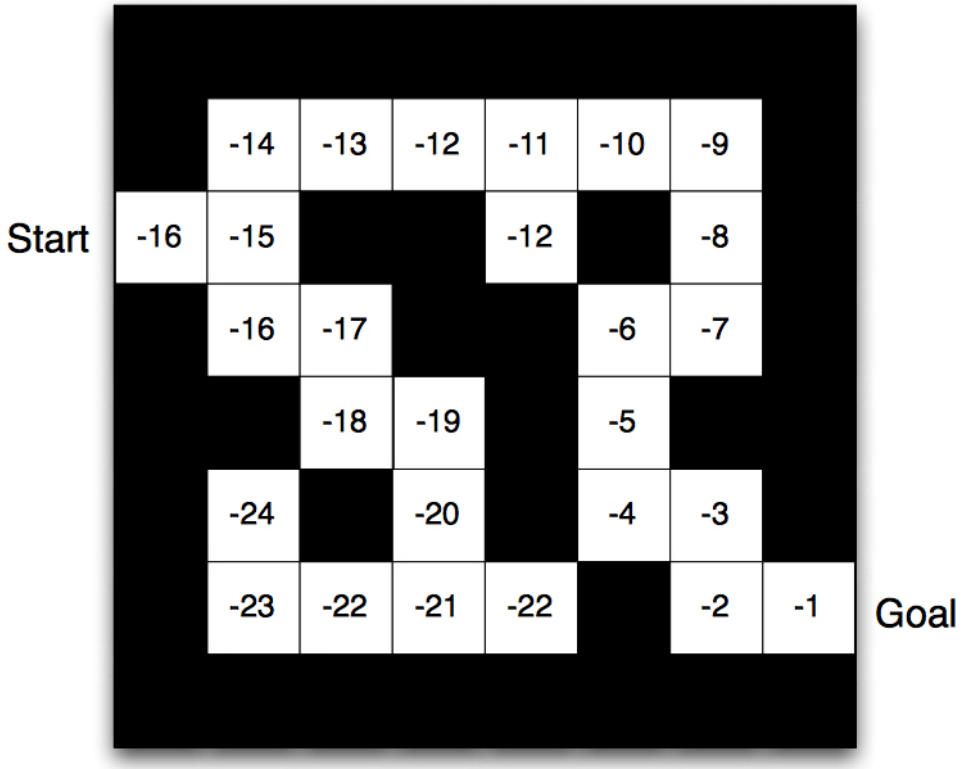
- Value function $v_\pi$: expected discounted sum of future rewards under a particular policy $\pi$

$$v_\pi(s_t = s) = \mathbb{E}_\pi \left[ G_t | s_t = s \right]$$

$$v_\pi \left( s_t = s \right) = \mathbb{E}_\pi \left[ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots | s_t = s \right]$$

- Discount factor $\gamma$ weighs immediate vs future rewards
- Used to evaluate the goodness/badness of states and actions
- And decide how to act by comparing policies

# Maze Example: Value




- Numbers represent value $v_\pi(s)$ of each state $s$

# Model

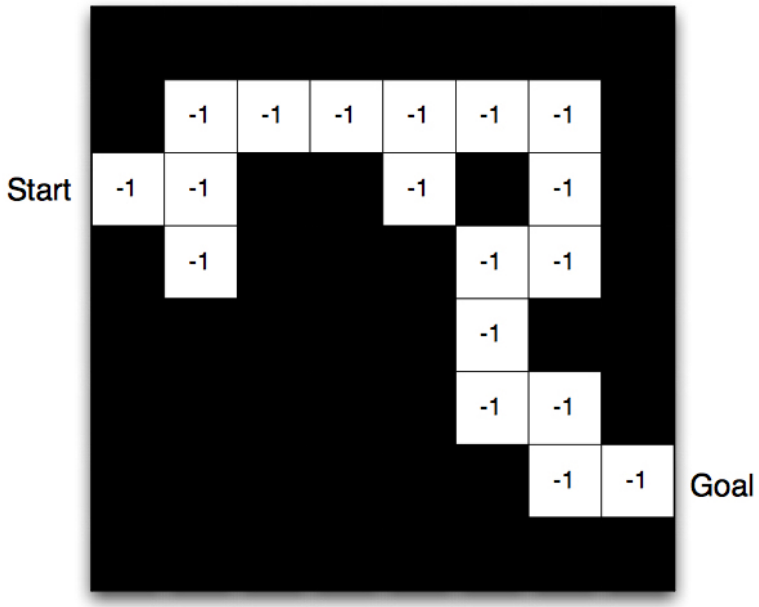
- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state

$$\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right]$$

- $\mathcal{R}$ predicts the next (immediate) reward

$$\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$$

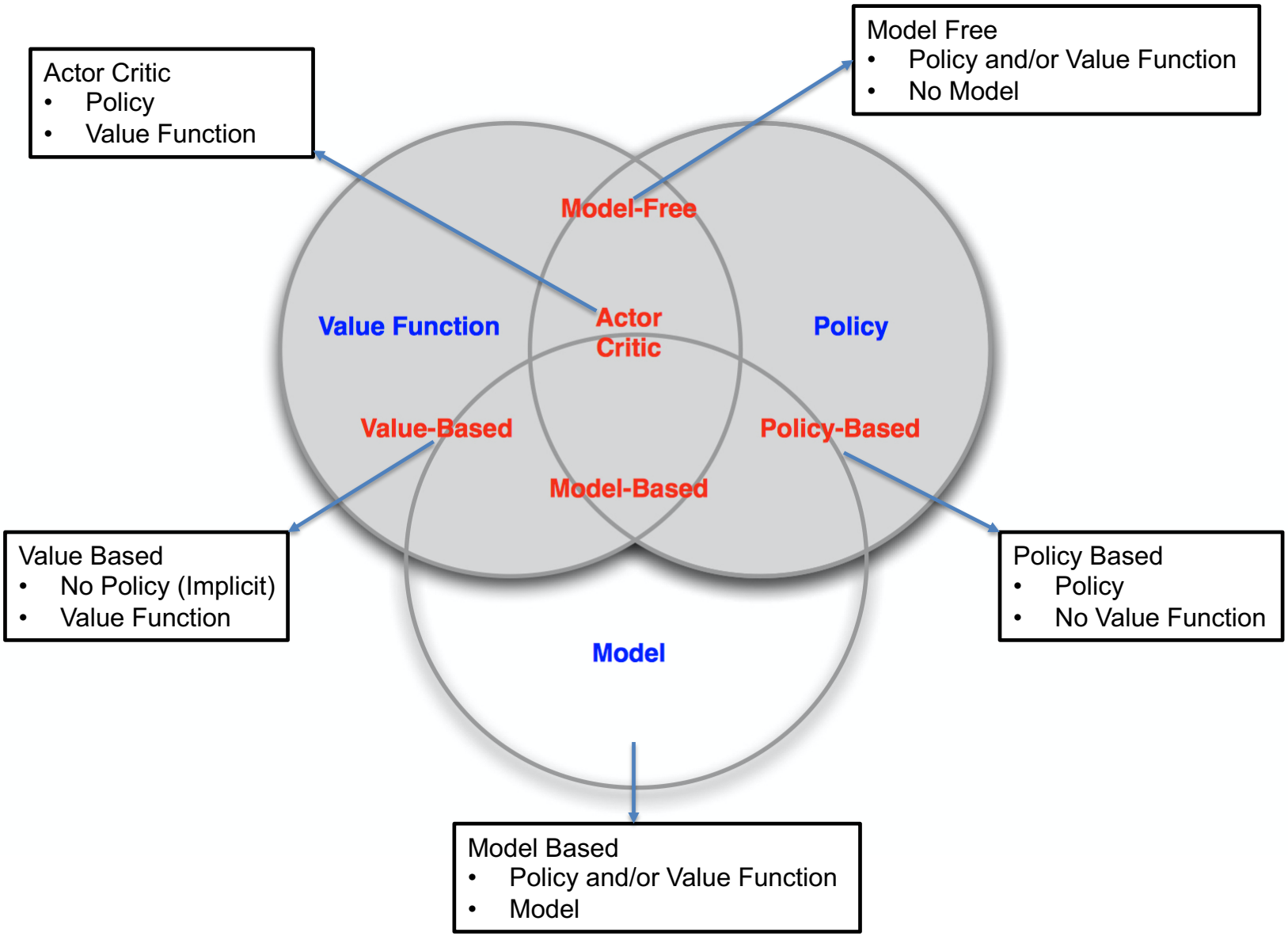
# Maze Example: Model




- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model $\mathcal{P}^a_{ss'}$
- Numbers represent immediate reward $\mathcal{R}^a_s$ from each state $s$ (same for all $a$)

# RL Taxonomy


Actor Critic
• Policy
• Value Function
Model Free
• Policy and/or Value Function
• No Model
Model-Free
Value Function
Actor Critic
Policy
Value-Based
Policy-Based
Model-Based
Model
Value Based
• No Policy (Implicit)
• Value Function
Policy Based
• Policy
• No Value Function
Model Based
• Policy and/or Value Function
• Model

# RL Framework

# Two important tasks in RL: Evaluation and Control

- Evaluation/Prediction
  - Estimate values/the expected rewards given a policy
- Control
  - Optimization: find the best policy
- Agent should also consider exploration and exploitation for the actions it tries.
  - Exploration: trying new things that might enable the agent to make better decisions in the future
  - Exploitation: choosing actions that are expected to yield good reward given past experience
- Recall bandit problems

# Outline

# Introduction to MDPs

- For MDPs, the current state completely characterises the process.
- Almost all RL problems can be formalised as MDPs
- A Markov process is a sequence of random states $S_1, S_2, \ldots$ with the Markov property.
- Markov property: future is independent of past given present

$$\mathbb{P}\left[S_{t+1}|S_t\right] = \mathbb{P}\left[S_{t+1}|S_1, \ldots, S_t\right]$$

**Definition**

A Markov Process (or Markov Chain) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $P$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s'|S_t = s\right]$

# Markov Reward Process

A Markov reward process is a Markov chain with values.

**Definition**

A Markov Reward Process is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- $\gamma$ is a discount factor, $\gamma \in [0, 1]$

# Markov Decision Process

- A Markov decision process (MDP) is a Markov reward process with decisions. It is an environment in which all states are Markov.

**Definition**

A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right]$
- $R$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$
- $\gamma$ is a discount factor, $\gamma \in [0, 1]$.

# Policy

**Definition**

A policy $\pi$ is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are stationary (time-independent),

$$A_t \sim \pi\left(\cdot|S_t\right), \forall t > 0$$

# Policy

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy $\pi$
- The state sequence $S_1, S_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
- where

$$\mathcal{P}^\pi_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{ss'}$$

$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s$$

# Value Function

**Definition**

The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state $s$, and then following policy $\pi$

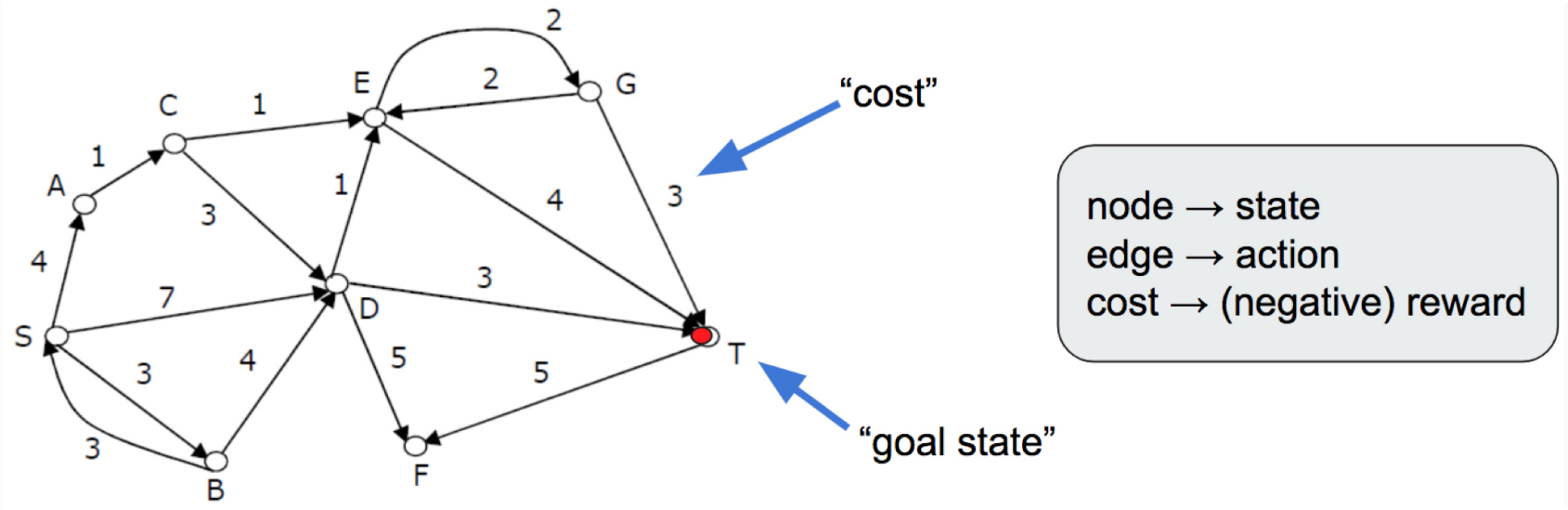
$$v_\pi(s) = \mathbb{E}_\pi \left[ G_t | S_t = s \right]$$

**Definition**

The action-value function $q_\pi(s, a)$ is the expected return starting from state $s$, taking action $a$, and then following policy $\pi$

$$q_\pi(s, a) = \mathbb{E}_\pi \left[ G_t | S_t = s, A_t = a \right]$$

# Simple MDP: Shortest Path Problem



# Bellman Expectation Equation

Calculating the long-term return $G_t$ of value function is difficult and time-consuming.
The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma v_\pi \left( S_{t+1} \right) | S_t = s \right]$$

The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma q_\pi \left( S_{t+1}, A_{t+1} \right) | S_t = s, A_t = a \right]$$

# Bellman Expectation Equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi\left(s'\right)$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi\left(s'\right) \right)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi\left(a'|s'\right) q_\pi\left(s', a'\right)$$

# Solving the Bellman Equation

- The Bellman equation is a linear equation
- Computational complexity is $O(n^3)$ for $n$ states in MRPs
- Computational complexity is $O((nm)^3)$ for $n$ states and $m$ actions in MDPs
- Direct solution only possible for small MRPs/MDPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

# Optimal Value Function

- The optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

- The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

- An MDP is "solved" when we know the optimal value function.

# Optimal Policy

For any Markov Decision Process

- There exists an optimal policy $\pi_*$ that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
- All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$

# Finding an Optimal Policy

- An optimal policy can be found by maximizing over $q_*(s, a)$

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

# Bellman Optimality Equation

$$v_*(s) = \max_a q_*(s,a)$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*\left(s'\right)$$

$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*\left(s'\right)$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*\left(s', a'\right)$$

# Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

# Outline

# Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- (i.e., reward function, transition probabilities)
- For evaluation(prediction):
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a fixed policy $\pi$
  - or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
  - Output: value function $v_\pi$
- For control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function $v_*$
  - or: optimal policy $\pi_*$

# Policy Evaluation

Given an exact model (i.e., reward function, transition probabilities), and a fixed policy $\pi$, how to evaluate the policy?

- Recall: Bellman Expectation Equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi \left( s' \right) \right)$$

- Update rule: For all $s \in \mathcal{S}$,

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k \left( s' \right) \right)$$

- Arbitrary initialization: $v_0$
- The sequence $\{v_k\}$ converges to $v_\pi$ as $k \to \infty$
- $v_k = v_\pi$ is a fixed point

# Policy Evaluation, Iterative Algorithm

Iterative Policy Evaluation

Initialize $V(s)$ for $s \in S$ arbitrarily

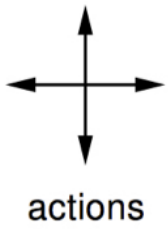
For $k = 1, 2, \ldots,$ do

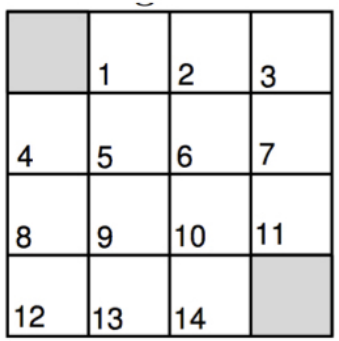
$$\forall s \in S, \quad V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k \left( s' \right) \right)$$

Stopping criterion: $\| V_{k+1} - V_k \|_\infty < \epsilon$

- two arrays, old $V$ and new $V$
- one array, in place, converges faster, In-Place Dynamic Programming

# Evaluating a Random Policy in the Small Gridworld



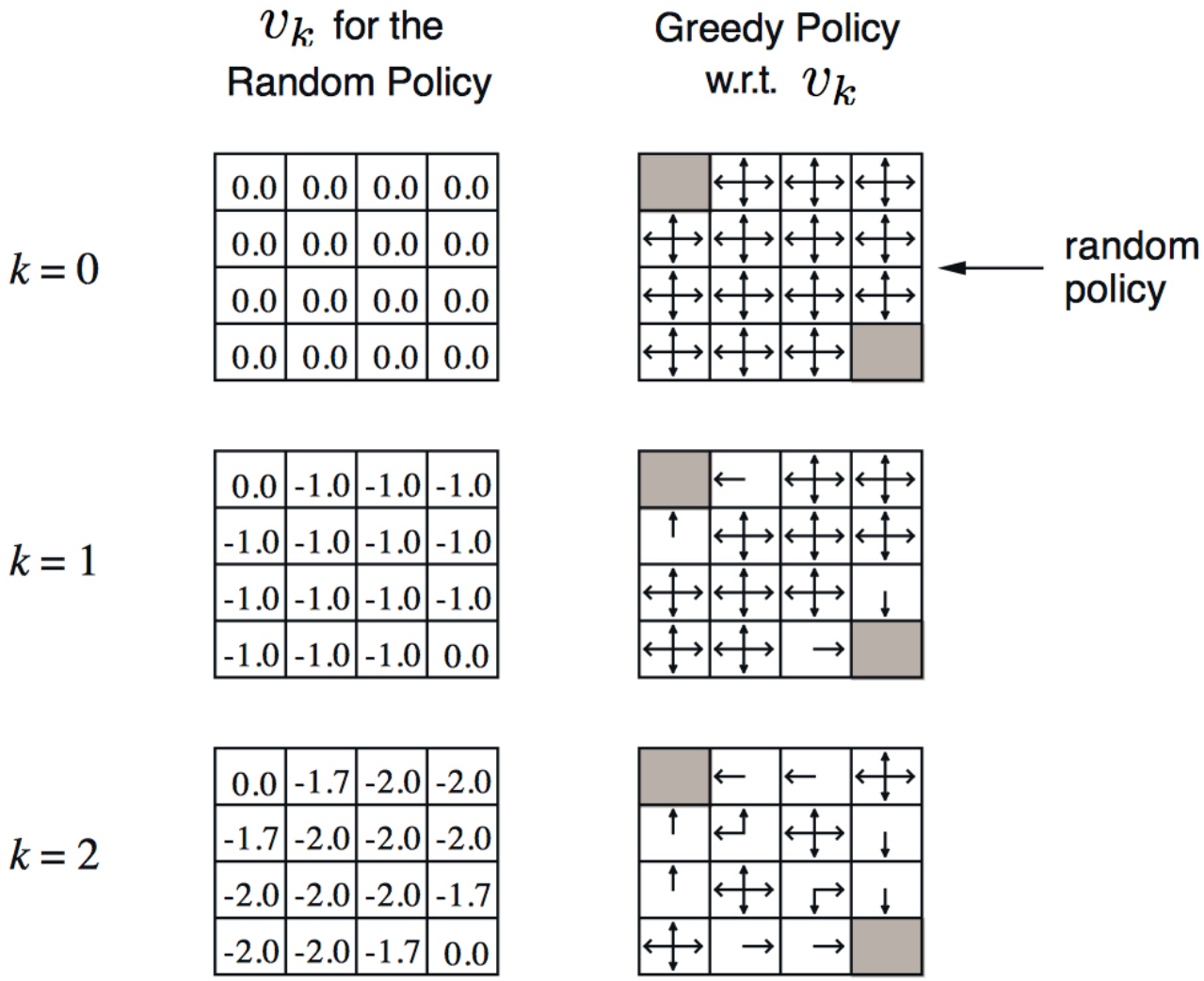




$r = -1$
on all transitions

- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, ..., 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is $-1$ until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

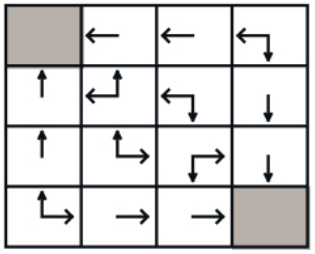
# Evaluating a Random Policy in the Small Gridworld



# Evaluating a Random Policy in the Small Gridworld

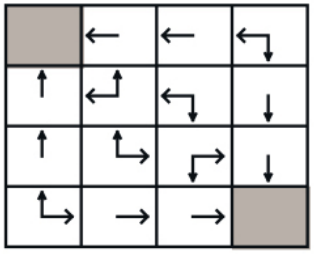
$k = 3$

| | | | |
|---|---|---|---|
| 0.0 | -2.4 | -2.9 | -3.0 |
| -2.4 | -2.9 | -3.0 | -2.9 |
| -2.9 | -3.0 | -2.9 | -2.4 |
| -3.0 | -2.9 | -2.4 | 0.0 |



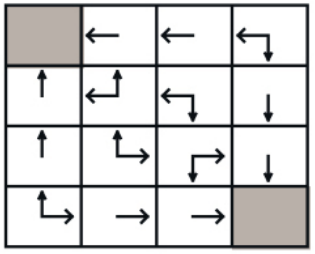
$k = 10$

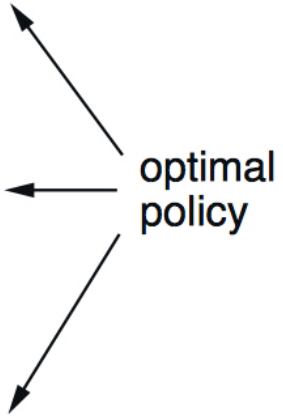
| | | | |
|---|---|---|---|
| 0.0 | -6.1 | -8.4 | -9.0 |
| -6.1 | -7.7 | -8.4 | -8.4 |
| -8.4 | -8.4 | -7.7 | -6.1 |
| -9.0 | -8.4 | -6.1 | 0.0 |



$k = \infty$

| | | | |
|---|---|---|---|
| 0.0 | -14. | -20. | -22. |
| -14. | -18. | -20. | -20. |
| -20. | -20. | -18. | -14. |
| -22. | -20. | -14. | 0.0 |





# Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can improve the policy by acting greedily

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}} q_\pi(s, a)$$

- This improves the value from any state s over one step,

$$q_\pi\left(s, \pi'(s)\right) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi\left(s, \pi'(s)\right) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) | S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, \pi'\left(S_{t+1}\right)\right) | S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi\left(S_{t+2}, \pi'\left(S_{t+2}\right)\right) | S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s\right] = v_{\pi'}(s) \end{aligned}$$

# Policy Improvement

- If improvements stop,

$$q_\pi\left(s, \pi'(s)\right) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- Then the Bellman optimality equation has been satisfied

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

- $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- $\pi$ is an optimal policy

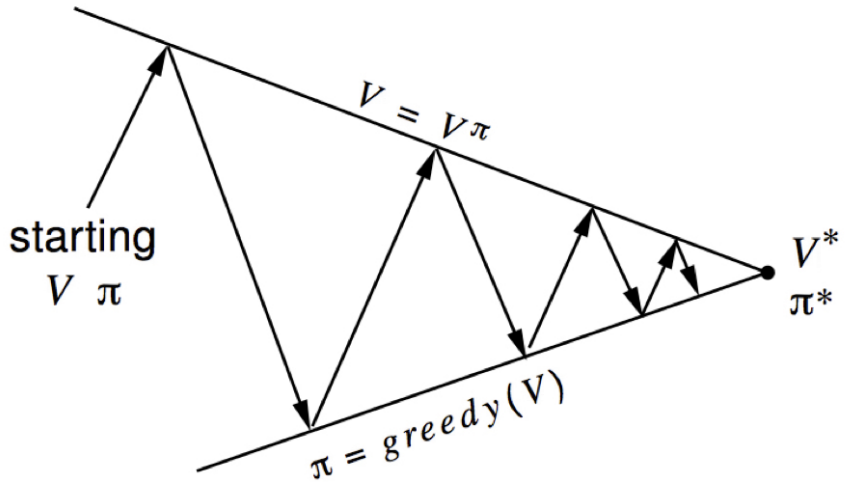
# Policy Iteration

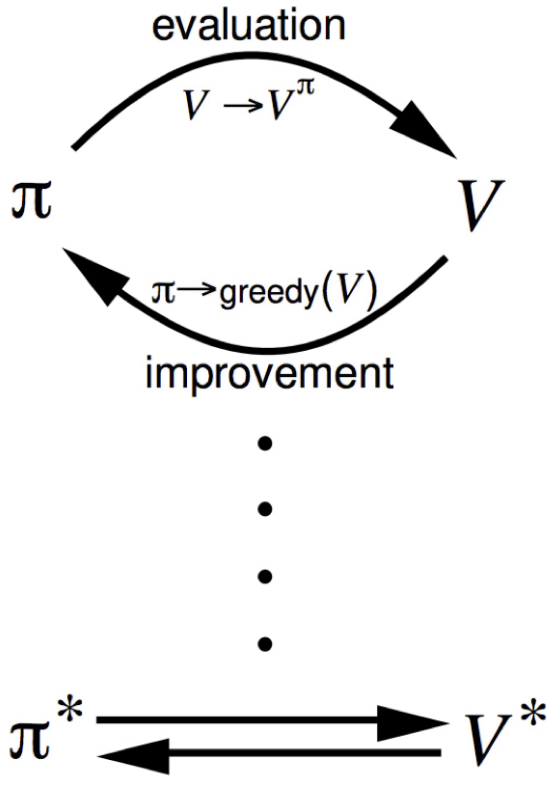
- Once a policy, $\pi$, has been improved using $v_\pi$ to yield a better policy, $\pi'$, we can then compute $v_{\pi'}$ and improve it again to yield an even better $\pi''$.
- Policy Iteration: a sequence of monotonically improving policies and value functions:

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

- $\xrightarrow{\mathrm{E}}$: policy evaluation
- $\xrightarrow{\mathrm{I}}$: policy improvement

# Policy Iteration




Policy evaluation Estimate $v_\pi$

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

# The Drawback of Policy Iteration

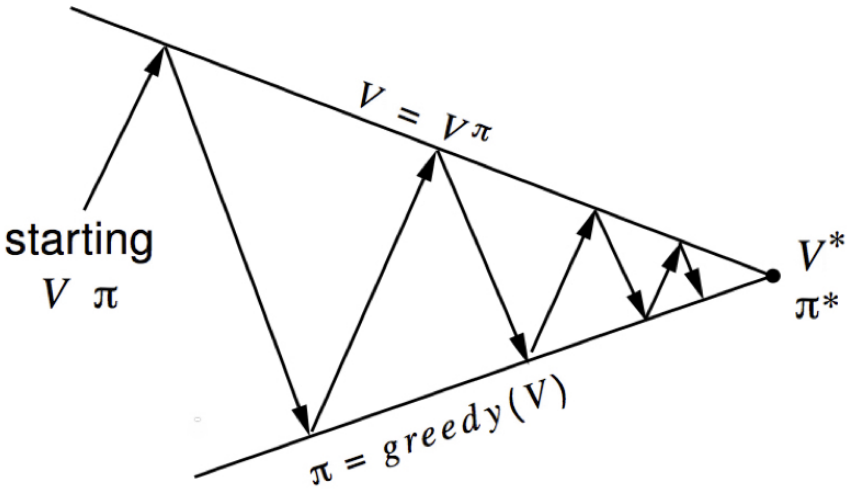
- Drawback: each of its iterations involves policy evaluation, which itself is a protracted computation requiring multiple sweeps through the state set.

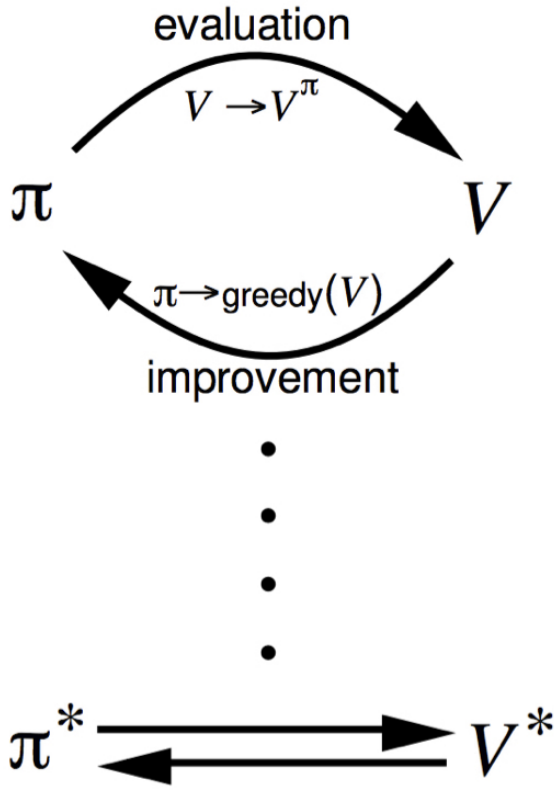
$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

# Modified Policy Iteration

- Does policy evaluation need to converge to $v_\pi$?
- Or should we introduce a stopping condition
  - e.g. $\epsilon-$convergence of value function
- Or simply stop after $k$ iterations of iterative policy evaluation?
- Why not update policy every iteration? i.e. stop after $k = 1$ (Value Iteration)

# Generalized Policy Iteration




Policy evaluation Estimate $v_\pi$

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

# Value Iteration

## Value Iteration

Initialize $V(s)$ for $s \in S$ arbitrarily

For $k = 1, 2, \ldots,$ do

$$\forall s \in S, \quad V_{k+1}(s) = \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k\left(s'\right) \right)$$

Stopping criterion: $\| V_{k+1} - V_k \|_\infty < \epsilon$

- $q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi\left(s'\right)$
- two arrays, old $V$ and new $V$
- one array, in place, converges faster

# Understanding Value Iteration

- A special case of policy iteration
  - In each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.
- Bellman optimality equation

$$v_*(s) = \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_* \left(s'\right)$$

  - The idea of value iteration is to apply the updates iteratively
  - The value iteration update is identical to the policy evaluation update except that it requires the maximum to be taken over all actions

# Understanding Value Iteration

- Faster convergence is often achieved by interposing multiple policy evaluation updates between each policy improvement update.
- In general, the entire class of truncated policy iteration algorithms can be thought of as sequences of updates, some of which use policy evaluation updates and some of which use value iteration updates
- The max operation is added to some updates of policy evaluation

# Dynamic Programming Algorithms for MDPs

| Problem | Bellman Equation | Algorithm |
|---|---|---|
| Prediction | Bellman Expectation Equation | Iterative Policy Evaluation |
| Control | Bellman Expectation Equation + Greedy Policy Improvement | Policy Iteration |
| Control | Bellman Optimality Equation | Value Iteration |

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O\left(mn^2\right)$ per iteration, for $m$ actions and $n$ states
- Could also apply to action-value function $q_\pi(s,a)$ and $q_*(s,a)$
- Complexity $O\left(m^2n^2\right)$ per iteration

# Bellman Operator

- Bellman Operator $\mathcal{B}$ transforms $v$ to another function $\mathcal{B}v$ on $\mathcal{S}$:

$$\mathcal{B}v(s) := \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v\left(s'\right) \right)$$

  Similarly for $q(s, a)$
- Bellman Optimality Equation re-expressed as: $v_* = \mathcal{B}v_*$ ($v_*$ is called fixed point of $\mathcal{B}$)
- it is a contraction
- fixed point exists and is unique

# Contraction of $\mathcal{B}$

**Theorem**

*For any $v_1$ and $v_2$, $\|\mathcal{B}v_1 - \mathcal{B}v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$*

**Proof:** For any $s \in \mathcal{S}$:

$$
\begin{aligned}
&\mathcal{B}v_1(s) - \mathcal{B}v_2(s) \\
\leq &\max_{a\in\mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_1(s') \right) - \max_{a\in\mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_2(s') \right) \\
\leq &\max_{a\in\mathcal{A}} \left[ \left( \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_1(s') \right) - \left( \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_2(s') \right) \right] \\
\leq &\gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^{a1}(v_1(s') - v_2(s')) \leq \gamma \max_{s'\in\mathcal{S}}(v_1(s') - v_2(s')) \\
= &\gamma \|v_1 - v_2\|_\infty
\end{aligned}
$$

Symmetrically, $\mathcal{B}v_2(s) - \mathcal{B}v_1(s) \leq \gamma \|v_1 - v_2\|_\infty$ Therefore, $\|\mathcal{B}v_1 - \mathcal{B}v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$. □

# Existence and Uniqueness of Fixed Point

**Existence**

- For any $v_1$ and $v_2$ , contraction implies

$$\left\|\mathcal{B}^k v_1 - \mathcal{B}^k v_2\right\|_\infty \leq \gamma \left\|\mathcal{B}^{k-1} v_1 - \mathcal{B}^{k-1} v_2\right\|_\infty$$
$$\leq \cdots \leq \gamma^k \left\|v_1 - v_2\right\|_\infty \overset{k\to\infty}{\longrightarrow} 0$$

- So both $\mathcal{B}^k v_1$ and $\mathcal{B}^k v_2$ converge to some fixed point $\overline{v}$
- Rigorous proof requires contraction mapping theorem □

**Uniqueness**

- Assume $\mathcal{B}$ has two fixed points $\overline{v}_1 \neq \overline{v}_2$, then

$$\left\|\overline{v}_1 - \overline{v}_2\right\|_\infty = \left\|\mathcal{B}\overline{v}_1 - \mathcal{B}\overline{v}_2\right\|_\infty \leq \gamma \left\|\overline{v}_1 - \overline{v}_2\right\|_\infty$$

- Hence $\left\|\overline{v}_1 - \overline{v}_2\right\|_\infty = 0$, $\overline{v}_1 = \overline{v}_2$ □

# Convergence of Value Iteration

- Contraction of $\mathcal{B}$ implies

$$\|v_k - v_*\|_\infty = \left\|\mathcal{B}^k v_0 - \mathcal{B}^k v_*\right\|_\infty \leq \gamma^k \|v_0 - v_*\|_\infty$$

- Value Iteration converges to $v_*$ exponentially fast
- Can be used to decide termination condition for Value Iteration

# Outline

# Monte Carlo (MC) Methods

- Monte Carlo methods are learning methods
  - Experience $\rightarrow$ values, policy
- MC is model-free: no knowledge of MDP transitions /rewards
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate

# Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy $\pi$

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

- Remember that the return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

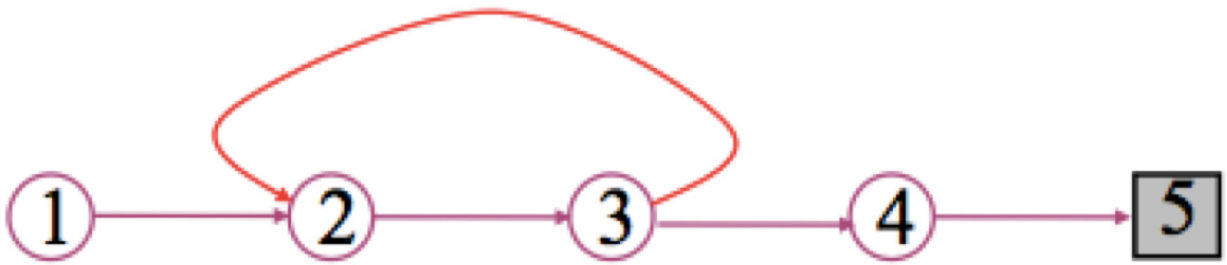
- Remember that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi \left[ G_t | S_t = s \right]$$

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

# Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy $\pi$




- Every-Visit MC: average returns for every time s is visited in an episode
- First-Visit MC: average returns only for first time s is visited in an episode
- Both converge asymptotically

# First-Visit MC Policy Evaluation

- To evaluate state $s$
- The first time-step $t$ that state $s$ is visited in an episode
- Increment counter: $N(s) \leftarrow N(s) + 1$
- Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

# Every-Visit MC Policy Evaluation

- To evaluate state $s$
- Every time-step $t$ that state $s$ is visited in an episode
- Increment counter: $N(s) \leftarrow N(s) + 1$
- Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

# Incremental Mean

- The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally:

$$\begin{aligned}
\mu_k &= \frac{1}{k} \sum_{j=1}^{k} x_j \\
&= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\
&= \frac{1}{k} \left( x_k + (k-1)\mu_{k-1} \right) \\
&= \mu_{k-1} + \frac{1}{k} \left( x_k - \mu_{k-1} \right)
\end{aligned}$$

# Incremental Monte Carlo Updates

- Update $V(s)$ incrementally after episode
- For each state $S_t$ with return $G_t$

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

# MC Estimation of Action Values (Q)

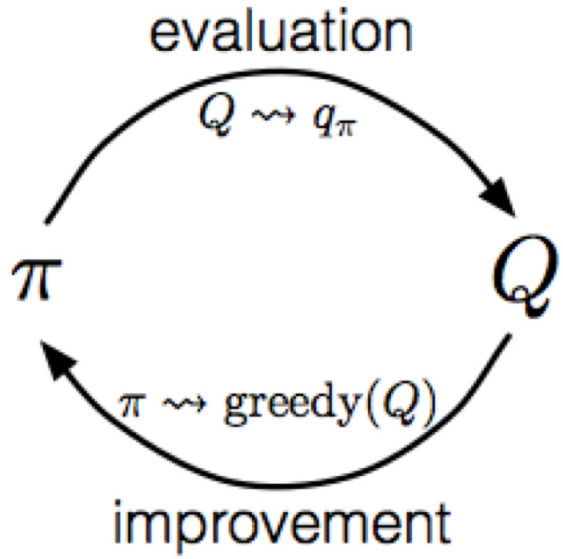
- Monte Carlo (MC) is most useful when a model is not available
  - We want to learn $q^*(s, a)$
- $\mathrm{q}_\pi(\mathrm{s}, \mathrm{a})$ - average return starting from state s and action a following $\pi$

$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi \left[ R_{t+1} + \gamma v_\pi \left( S_{t+1} \right) | S_t = s, A_t = a \right] \\
&= \sum_{s', r} p \left( s', r | s, a \right) \left[ r + \gamma v_\pi \left( s' \right) \right]
\end{aligned}
$$

- Converges asymptotically if every state-action pair is visited
- Exploring starts: Every state-action pair has a non-zero probability of being the starting pair

# Monte-Carlo Control

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$




- MC policy iteration step: Policy evaluation using MC methods
- Policy improvement step: greedily with respect to value (or action-value) function

$$\pi(s) \doteq \arg\max_a q(s, a)$$

# On-policy Monte Carlo Control

- On-policy: learn about policy currently executing
- How do we get rid of exploring starts?
  - The policy must be eternally soft: $\pi(a|s) > 0$ for all $s$ and $a$.
- For example, for $\epsilon-$soft policy, probability of an action,

$$\pi(\mathrm{a}|\mathrm{s}) = \begin{cases} \dfrac{\epsilon}{|\mathcal{A}(s)|}, & non-max \\ 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|}, & max(greedy) \end{cases}$$

- Similar to GPI (Generalized Policy Iteration): move policy towards greedy policy
- Converges to the best $\epsilon-$soft policy.

# Off-policy methods

- All learning control methods face a dilemma:
  - learn action values conditional on subsequent optimal behavior
  - behave non-optimally in order to explore all actions (to find the optimal actions)
- Off-policy methods: use two policies!
  - target policy: is learned about and becomes the optimal policy
  - behavior policy: more exploratory, used to generate behavior
  - learning is from data "off" the target policy, and the overall process is termed off-policy learning

# Off-policy methods

- For example, $\pi$ is the greedy policy (and ultimately the optimal policy) while $\mu$ is exploratory (e.g., $\epsilon$-soft) policy
- Off-policy vs. On-policy
  - often of greater variance
  - slower to converge
  - more powerful and general
- Idea: Importance Sampling
  - Weight each return by the ratio of the probabilities of the trajectory under the two policies.
  - SB (Sutton and Barton) Chapter 5.5

# Outline

# Temporal-Difference Learning

- "If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning."

  — Sutton and Barto 2017
- Combination of Monte Carlo & dynamic programming
- Model-free: no knowledge of MDP transitions / rewards
- Bootstraps and samples. TD updates a guess towards a guess
- Can be used in episodic or infinite-horizon non-episodic settings
- Immediately updates estimate of $v$ after each $(s, a, r, s')$ tuple.

# MC and TD Learning

- Goal: learn $v_\pi(s)$ from episodes of experience under policy $\pi$
- Incremental every-visit Monte-Carlo:
  - Update value $\mathrm{V}(\mathrm{S_t})$ toward actual return $G_t$

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

- Simplest Temporal-Difference learning algorithm: TD(0)
  - Update value $\mathrm{V}(\mathrm{S_t})$ toward estimated returns $R_{t+1} + \gamma V(S_{t+1})$

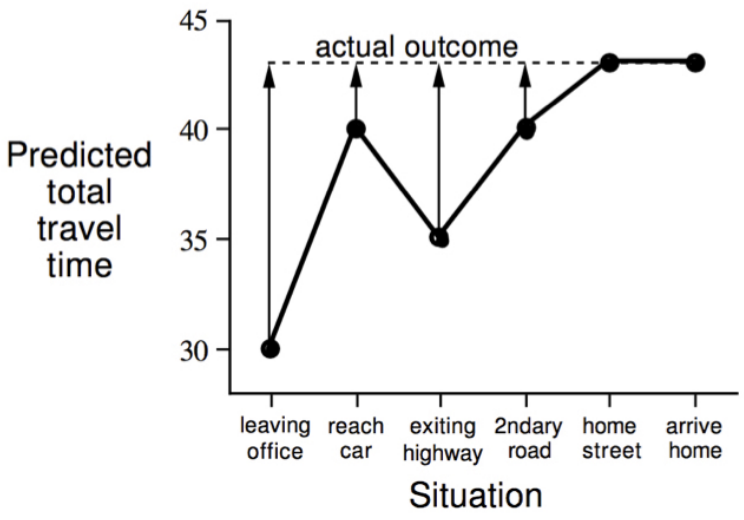
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

NewEstimate $\leftarrow$ OldEstimate $+$ StepSize [ Target $-$ OldEstimate]

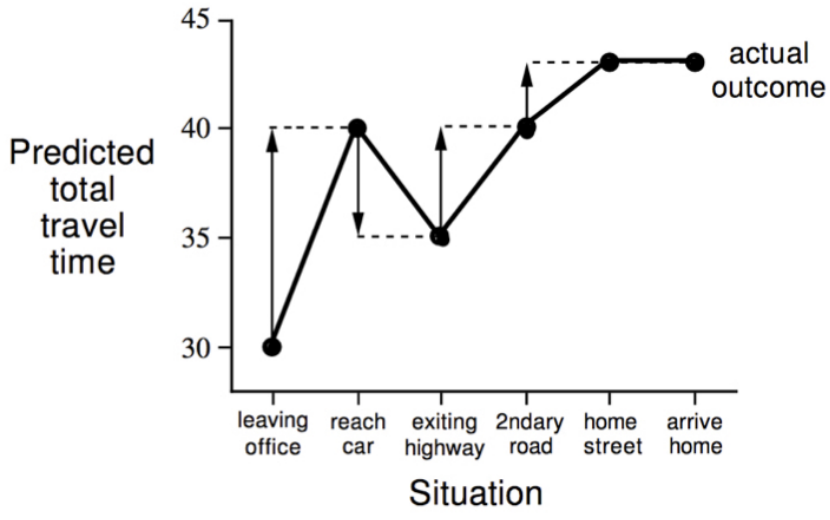
- $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error

# Driving Home Example: MC vs. TD

Changes recommended by Monte Carlo methods ($\alpha$=1)




Changes recommended by TD methods ($\alpha$=1)



# DP vs. MC vs. TD Learning

Remember:

MC: sample average return approximates expectation

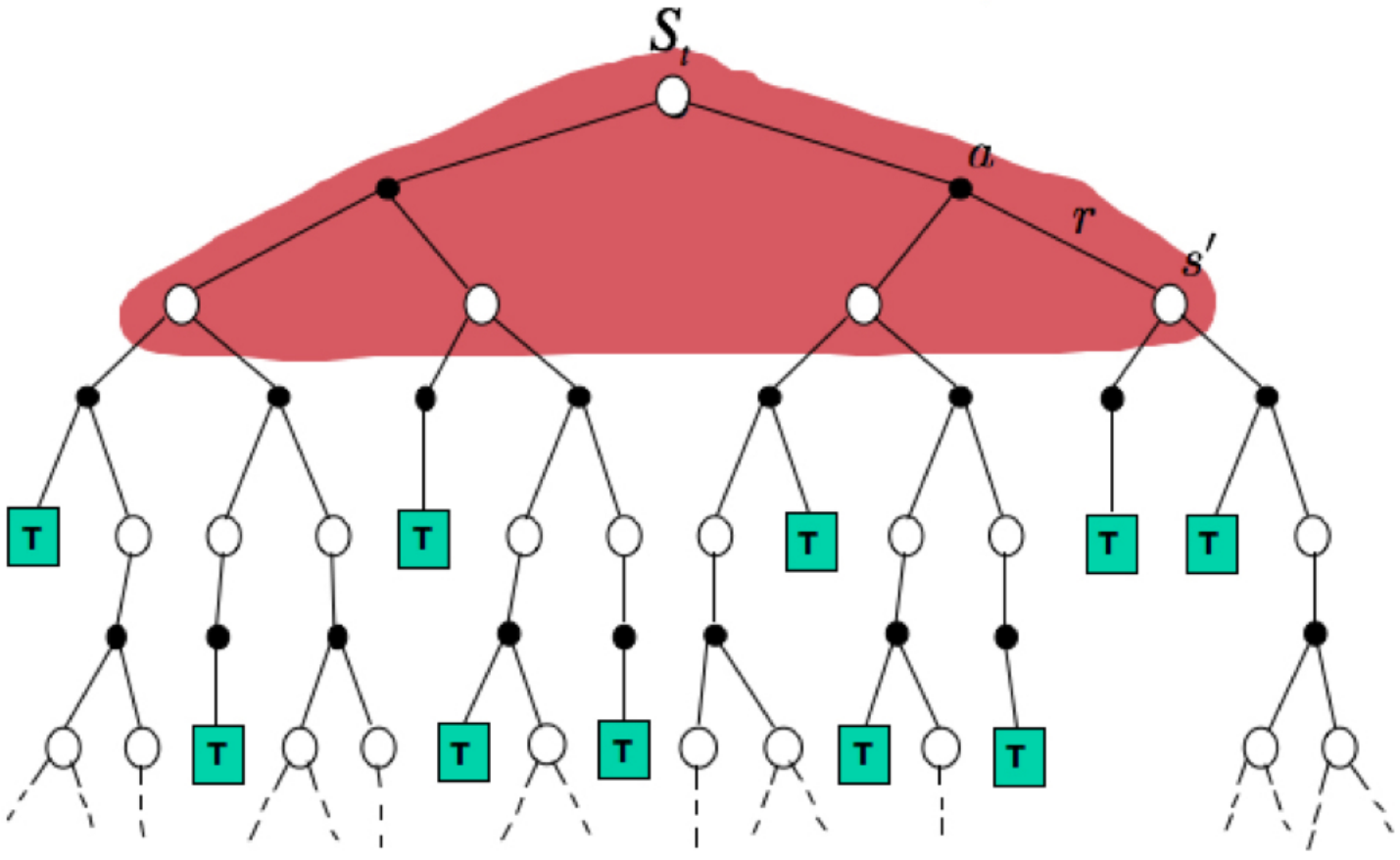
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \,.
\end{aligned}
$$

TD: combine both: Sample expected values and use a current estimate $V(S_{t+1})$ of the true $v_\pi(S_{t+1})$

DP: the expected values are provided by a model. But we use a current estimate $V(S_{t+1})$ of the true $v_\pi(S_{t+1})$

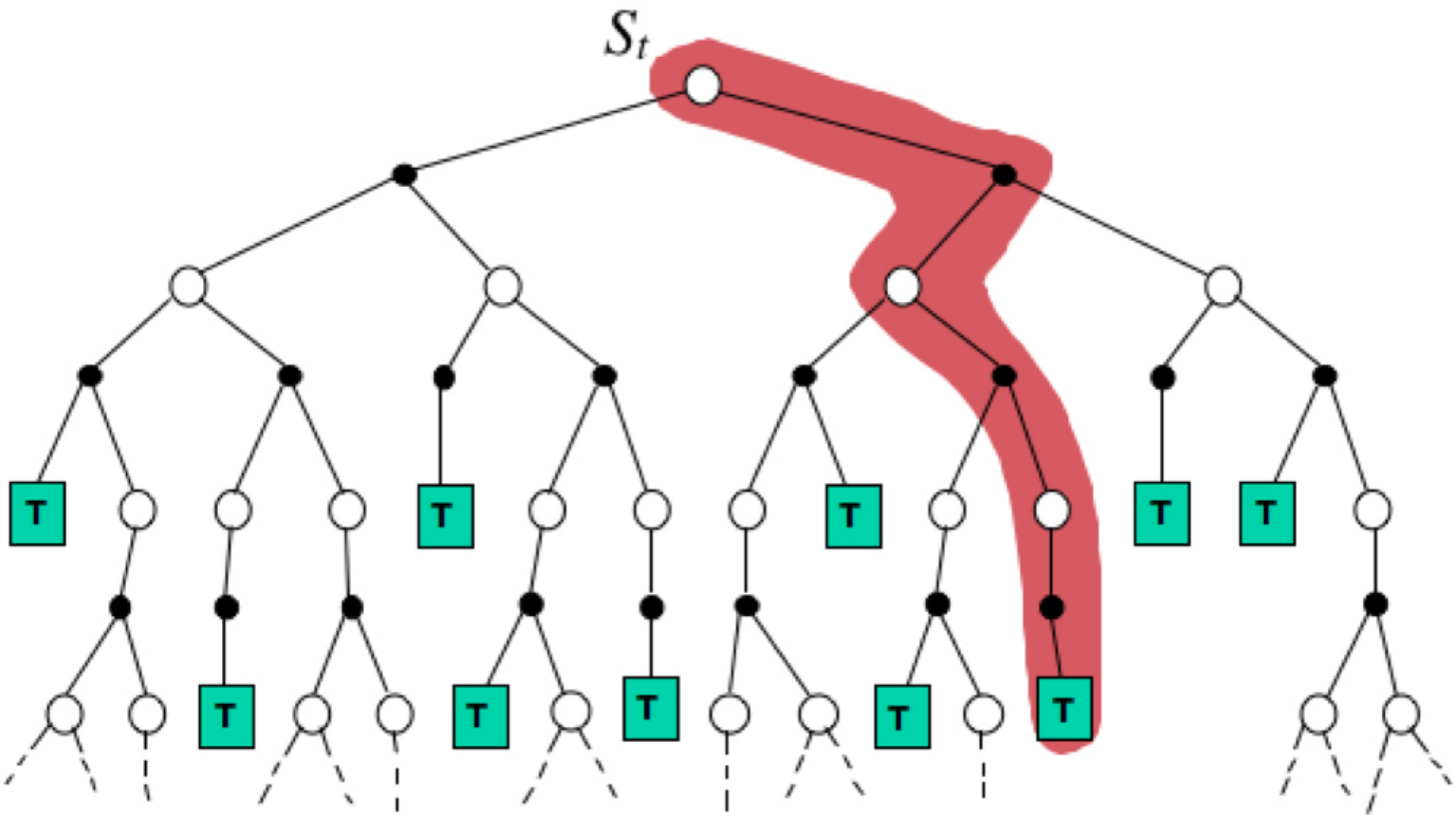
# Dynamic Programming

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s', r|S_t, a)[r + \gamma V(s')]$$



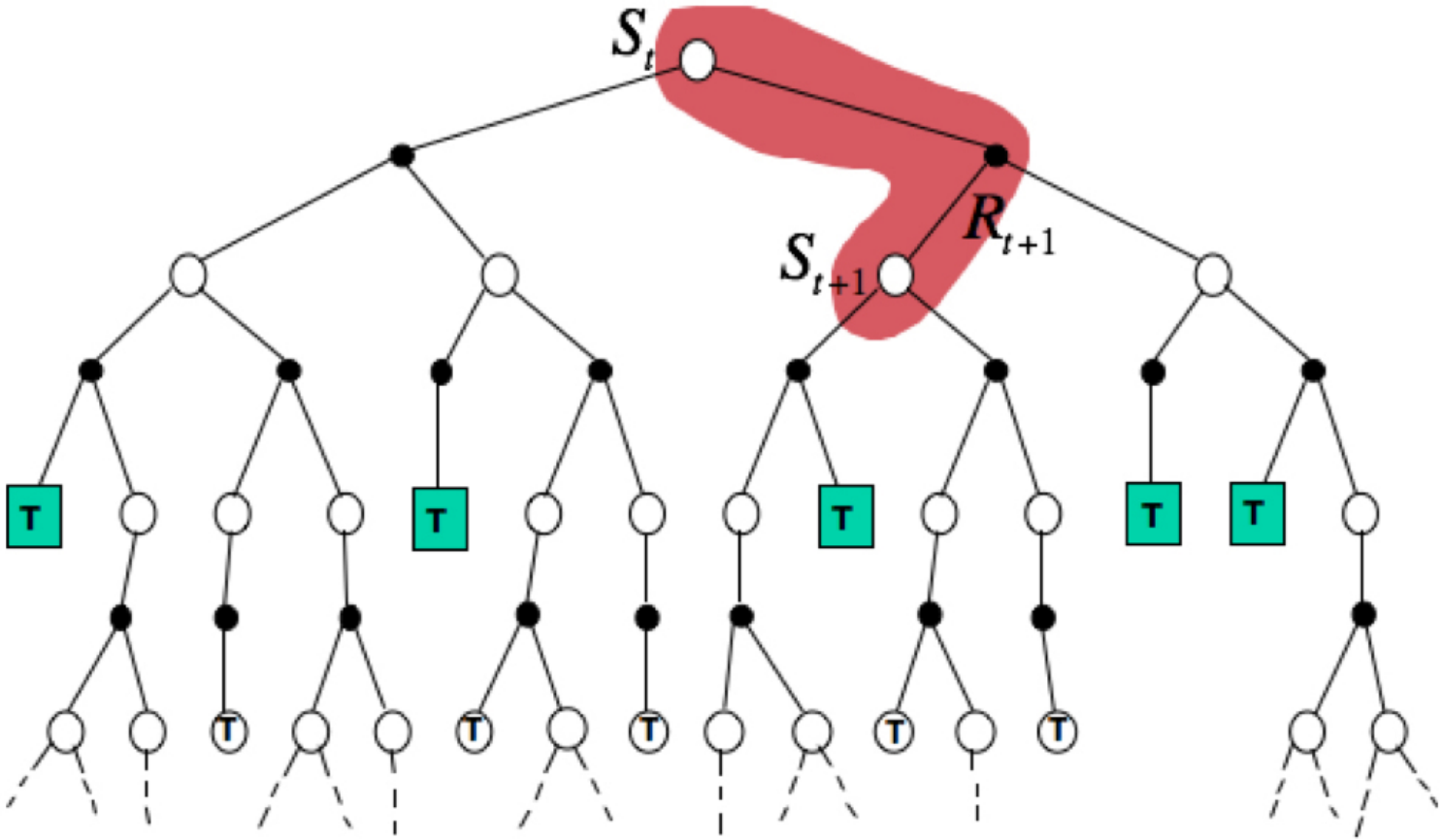
# Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$

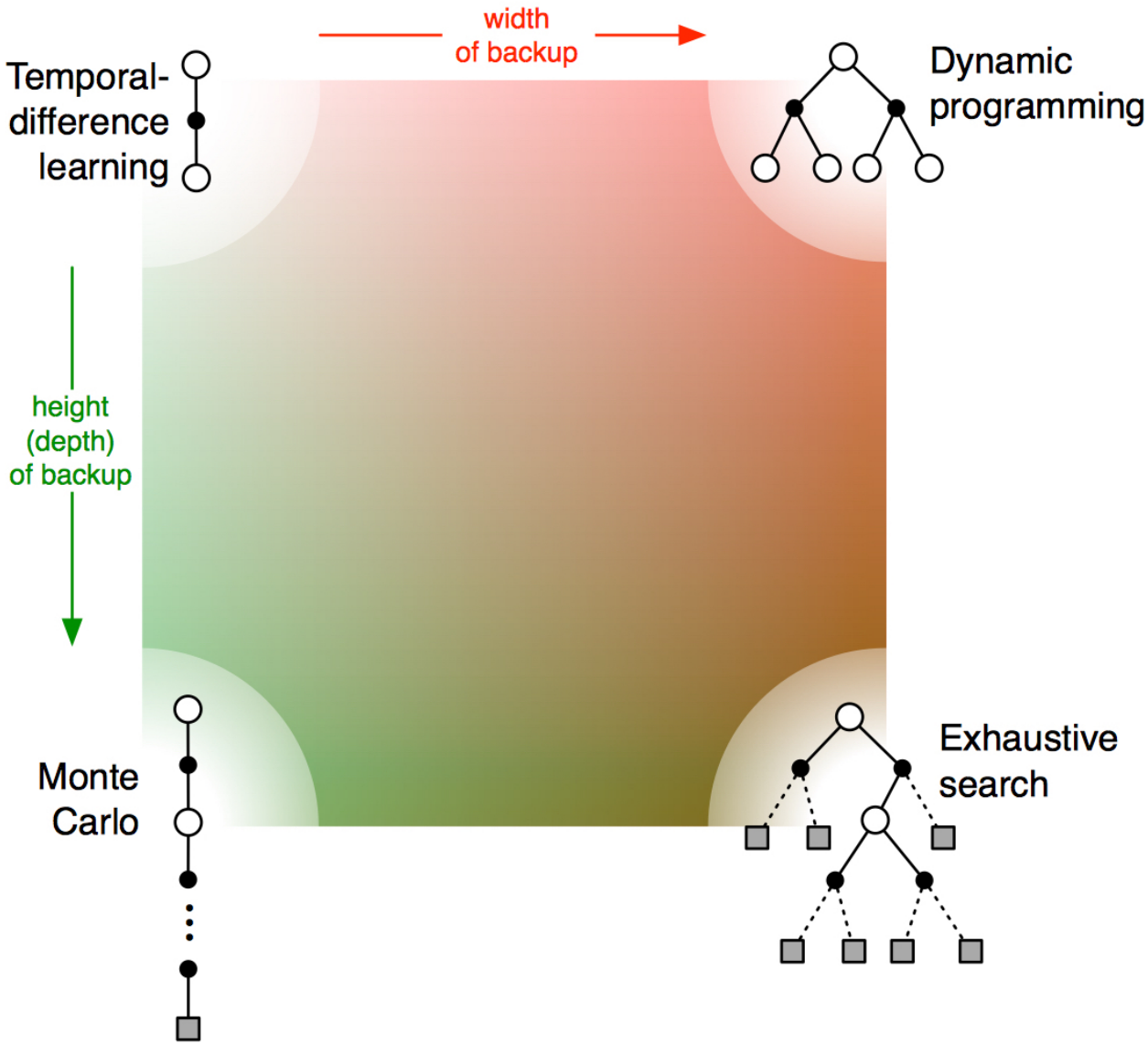


# Simplest TD(0) Method

$$V(S_t) \leftarrow V(S_t) + \alpha \left( R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



# Unified View of Reinforcement Learning

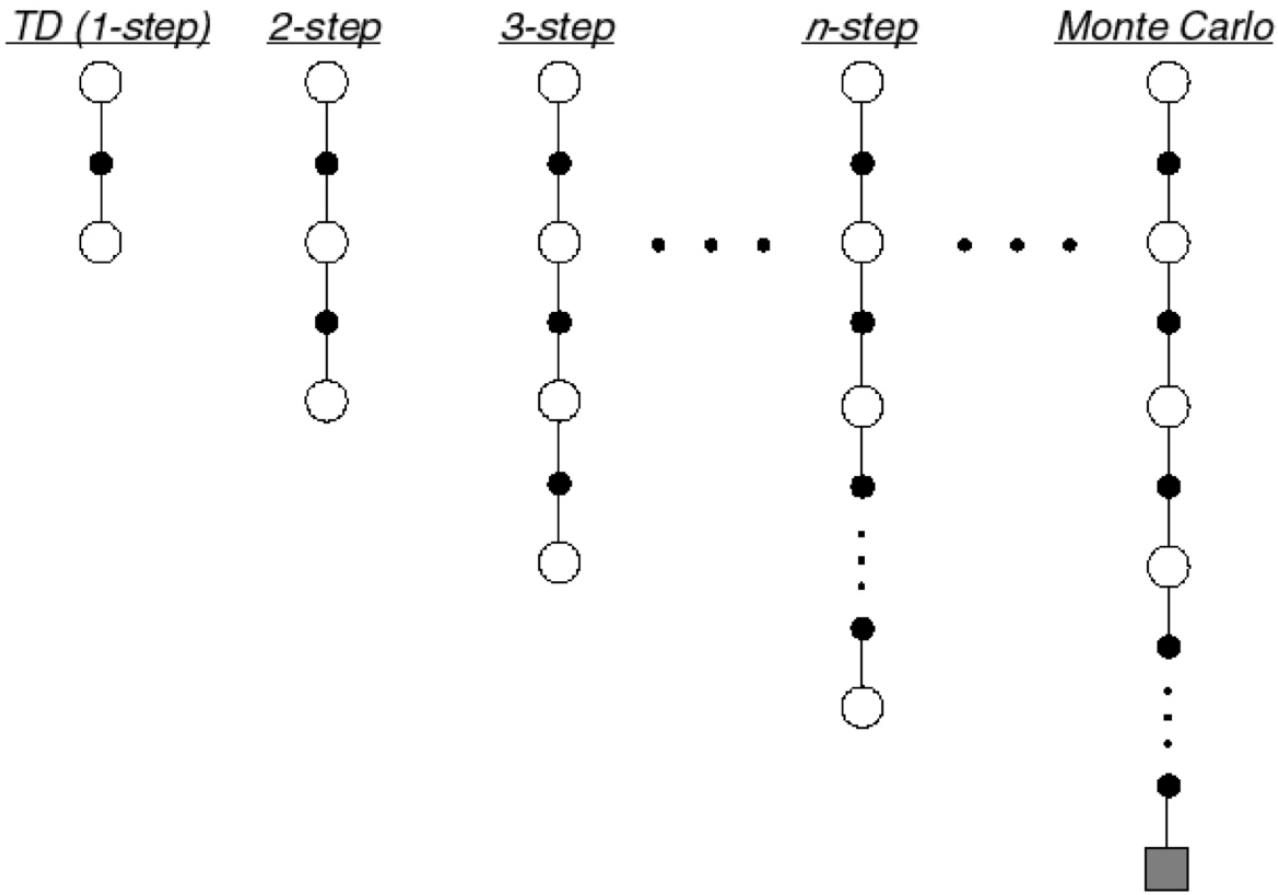


# TD Methods Bootstrap and Sample

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update does not involve an expected value
  - MC samples
  - DP does not sample
  - TD samples

# n-Step Prediction

- Let TD target look n steps into the future



# n-Step Return

- Consider the following n-step returns for $n = 1, 2, \infty$:

$$\begin{array}{lll} n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\ n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\ \vdots & & \vdots \\ n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T \end{array}$$

- Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V\left(S_{t+n}\right)$$

- n-step temporal-difference learning

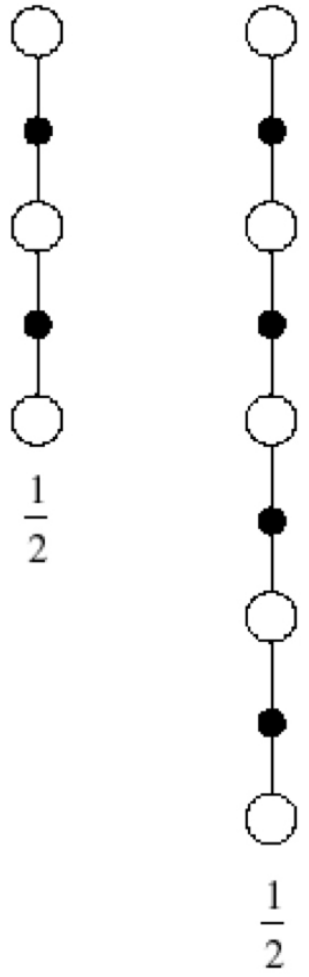
$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha \left( G_t^{(n)} - V\left(S_t\right) \right)$$

# Averaging n-Step Returns

- Average n-step returns over different n
- e.g. average the 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

- Combines information from two different time-steps
- Unify and generalize TD and Monte Carlo methods

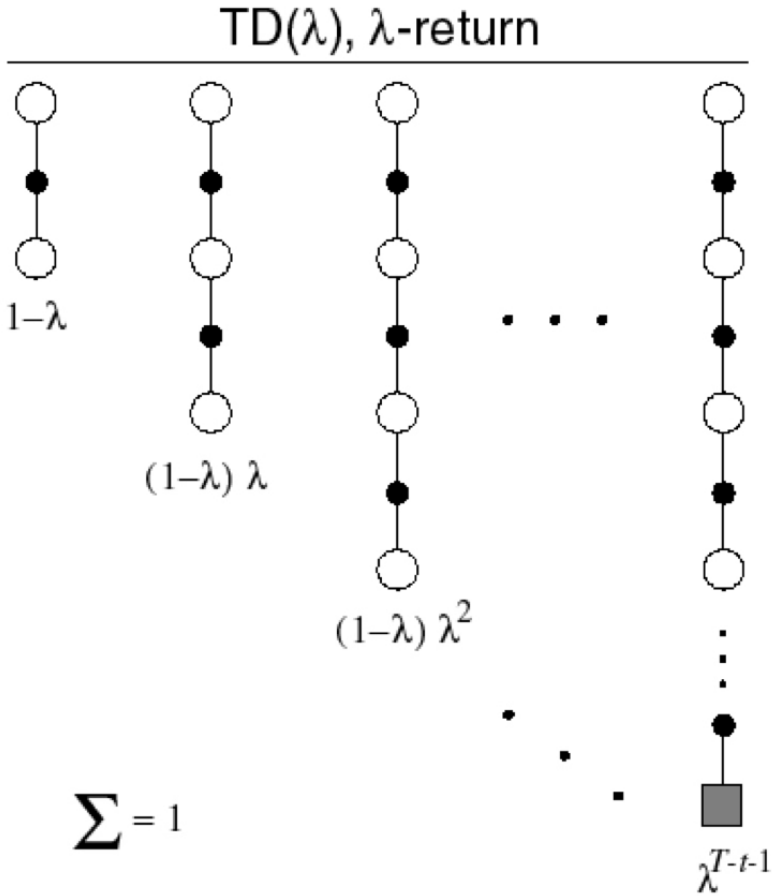


# TD($\lambda$) and $\lambda-$return

- The $\lambda-$return $G_t^\lambda$ combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

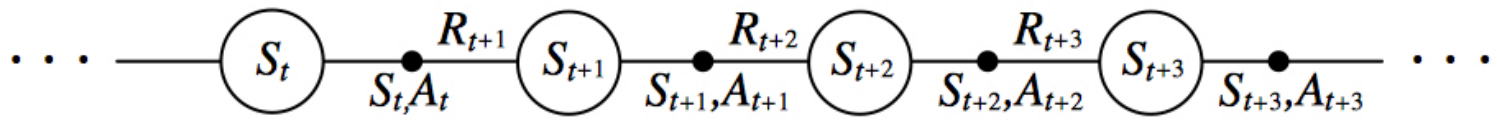
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

- Unify and generalize TD and Monte Carlo methods
  - $\lambda = 0$, TD(0)
  - $\lambda = 1$, MC
- SB (Sutton and Barton) Chapter 12



# TD: Learning An Action-Value Function

- Estimate $q_\pi$ for the current policy $\pi$




After every transition from a nonterminal state, $S_t$, do:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

If $S_{t+1}$ is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$

# Sarsa: On-Policy TD Control

- Sarsa: state-action-reward-state-action
- Turn this into a control method by always updating the policy to be current estimate:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

# Q-Learning: Off-Policy TD Control

- We now consider off-policy learning of action-values $Q(s, a)$
- Next action is chosen using behavior policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left( R_{t+1} + \gamma Q\left(S_{t+1}, A'\right) - Q(S_t, A_t) \right)$$

# Q-Learning: Off-Policy TD Control

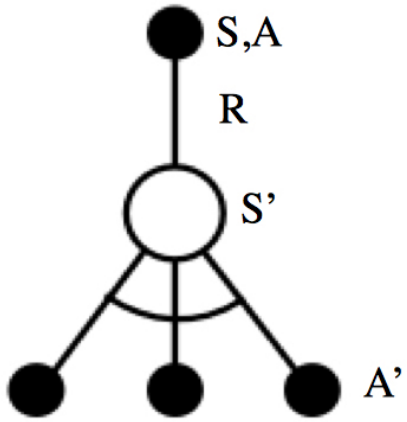
- We now allow both behavior and target policies to improve
- The target policy $\pi$ is greedy w.r.t. $Q(s, a)$

$$\pi\left(S_{t+1}\right) = \underset{a'}{\text{argmax}} Q\left(S_{t+1}, a'\right)$$

- The behavior policy $\mu$ is e.g. $\epsilon$-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q\left(S_{t+1}, A'\right) \\ = & R_{t+1} + \gamma Q\left(S_{t+1}, \text{argmax}_{a'}\, Q\left(S_{t+1}, a'\right)\right) \\ = & R_{t+1} + \max_{a'} \gamma Q\left(S_{t+1}, a'\right) \end{aligned}$$

# Q-Learning: Off-Policy TD Control




$$Q(S,A) \leftarrow Q(S,A) + \alpha \left( R + \gamma \max_{a'} Q\left(S', a'\right) - Q(S,A) \right)$$

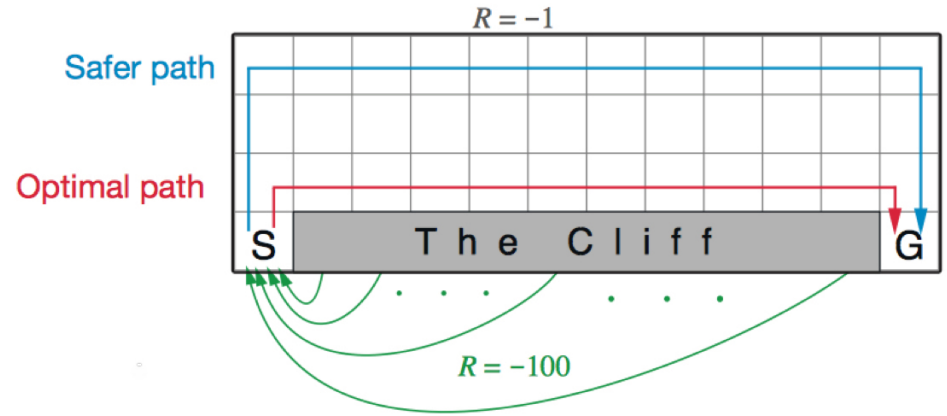
**Theorem**

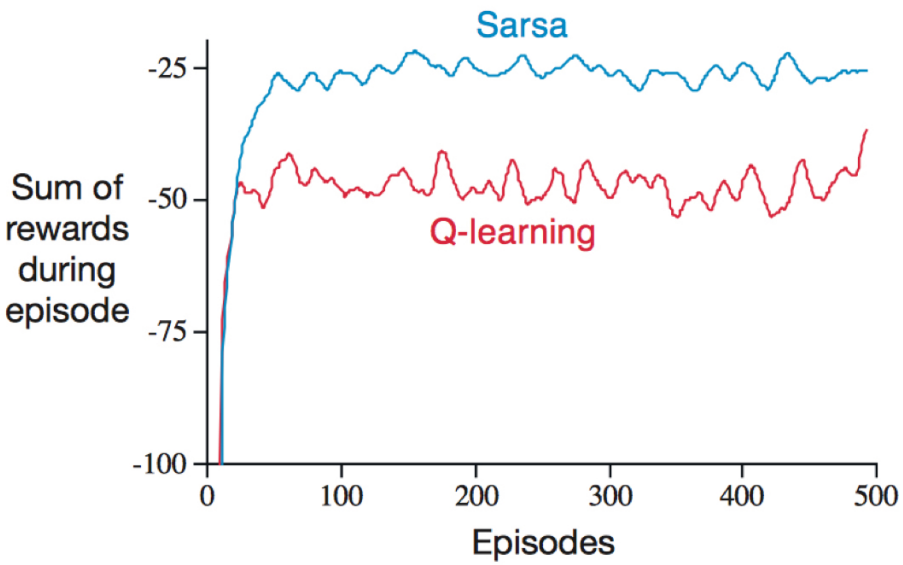
*Q-learning control converges to the optimal action-value function,* $Q(s,a) \rightarrow q_*(s,a)$

# Q-Learning: Off-Policy TD Control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$;
  until $S$ is terminal

# Cliff Walking




- $\epsilon = 0.1$.
- Q-learning learns the optimal policy. However, this results in its occasionally falling off the cliff because of the $\epsilon$-greedy action.
- Sarsa learns the longer but safer path.
- Here the online performance of Q-learning is worse than that of Sarsa.

# Outline

# Large-Scale Reinforcement Learning

- Tabular Solution Methods: we have represented value function $V(s)$ or $Q(s, a)$ by a lookup table
  - All calculations (such as Bellman operator) in finite state/action MDPs can be done exactly.
- Computationally, how to handle large MDPs?
  - Backgammon: $10^{20}$ states
  - Computer Go: $10^{170}$ states
  - Helicopter: continuous state space
- Statistically, how to generalize information to unseen states to reduce sample complexity?
- That is why we need good representation in RL

# Function Approximation (FA) in RL

- Tabular/exact representations (covered so far)
- Linearly parametric approximation

$$Q(s, a; \theta) = \theta^\top \phi(s, a), \qquad \text{where } \theta, \phi(s, a) \in \Re^d$$

- Nonlinear parametric approximation, e.g., neural nets
- Non-parametric representations: kernel, decision trees, ...

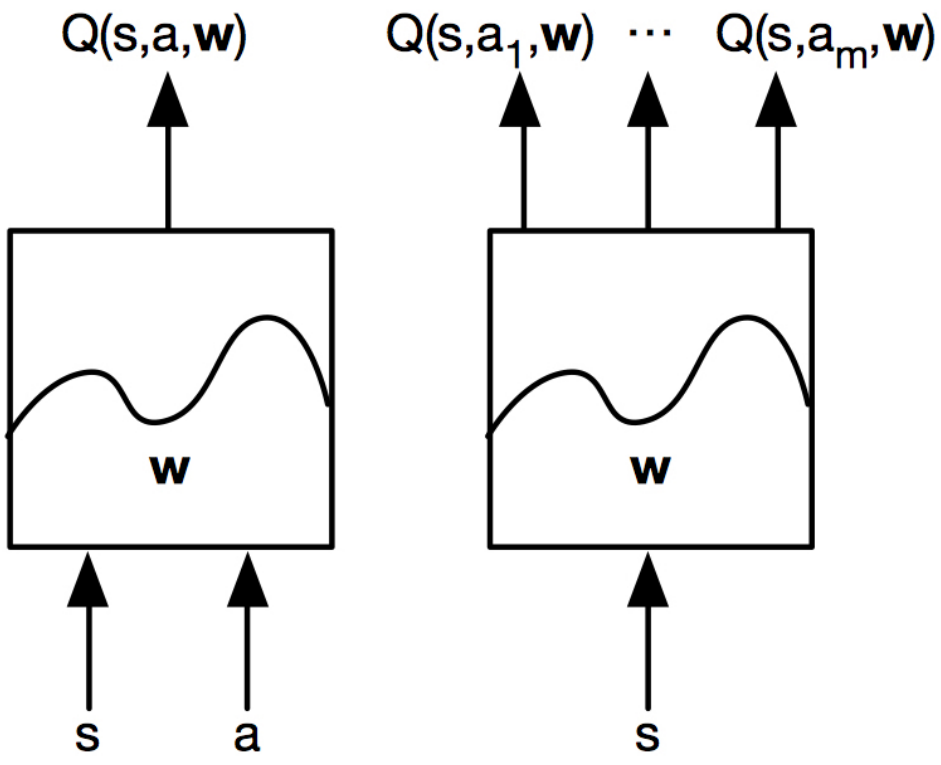
# Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by stochastic gradient descent (SGD)

# Q-Networks

- Represent value function by Q-network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



# Q-learning with Value Function Approximation

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q\left(s', a'\right)^* | s, a\right]$$

- Treat right-hand $r + \gamma \max_{a'} Q\left(s', a', w\right)$ as a target
- Minimize MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q\left(s', a', \mathbf{w}\right) - Q(s, a, \mathbf{w})\right)^2$$

# Q-learning with Value Function Approximation

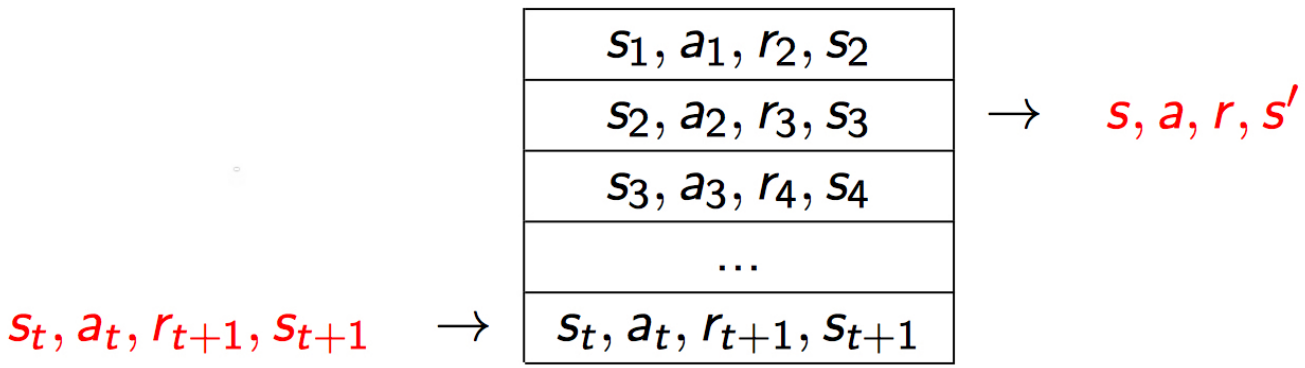
- Minimize MSE loss by stochastic gradient descent

$$I = \left( r + \gamma \max_{a} Q\left(s', a', \mathbf{w}\right) - Q(s, a, \mathbf{w}) \right)^2$$

- Converges to the optimal $Q^*$ using table lookup representation
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses both of these challenges by
  - Experience replay
  - Fixed Q-targets

# DQNs: Experience Replay & Fixed Q-targets

- To remove correlations, store dataset (called a **replay buffer**) $\mathcal{D}$ from agent's own experience




- Sample experiences from dataset and apply update

$$I = \left( r + \gamma \max_{a'} Q\left(s', a', \mathbf{w}^{-}\right) - Q(s, a, \mathbf{w}) \right)^2$$

- To deal with non-stationarity, target parameters $w-$ are held fixed for multiple updates

# DQNs

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay buffer $\mathcal{D}$
- $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the buffer $\mathcal{D}$
- Compute the target value $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-)$ w.r.t. old fixed parameters $\mathbf{w}^-$
- Use stochastic gradient descent to update the network weights

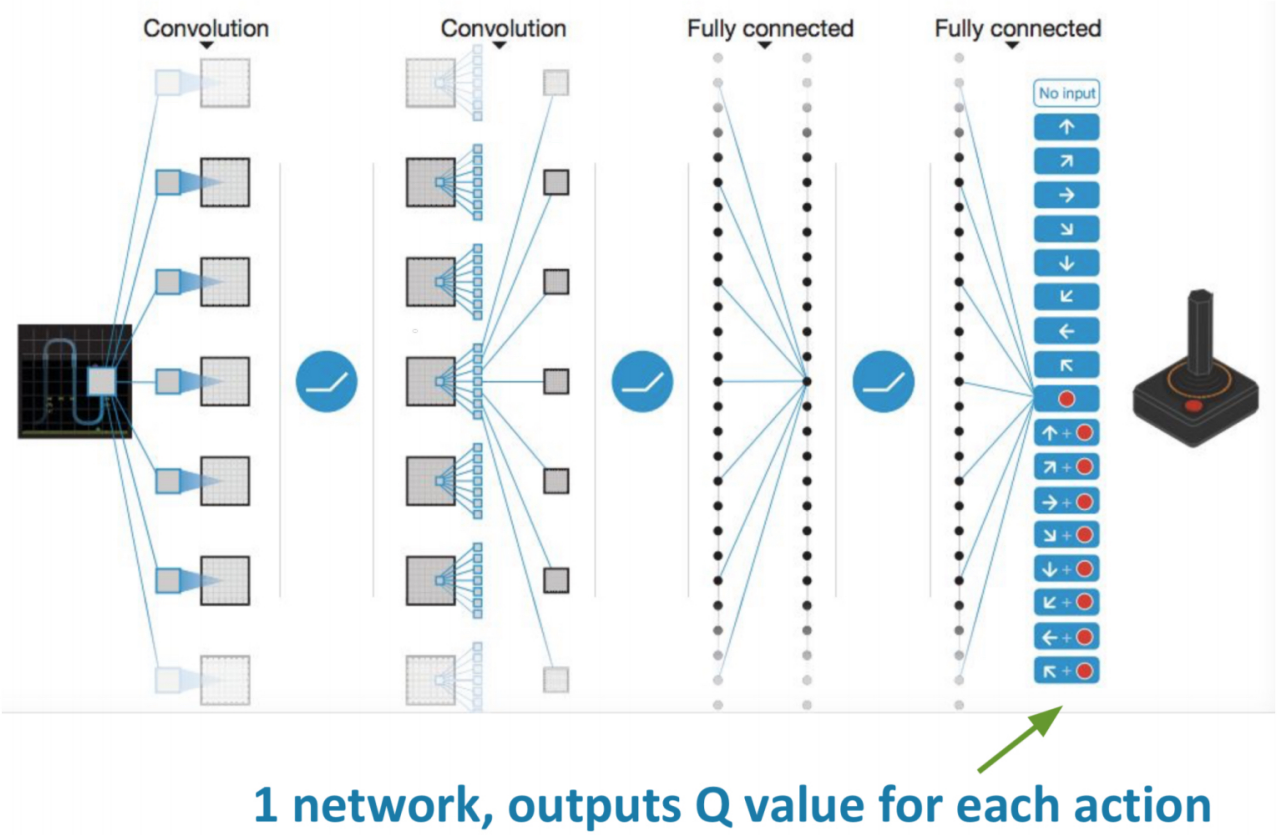
$$\Delta \mathbf{w} \leftarrow \alpha \left[ r_t + \gamma \max_{a'} \underbrace{Q\left(s', a', \mathbf{w}^-\right)}_{\text{target network}} - \underbrace{Q(s, a, \mathbf{w})}_{\text{q-network}} \right] \nabla_w Q(s, a, \mathbf{w})$$

- $\mathbf{w}^- \leftarrow \mathbf{w}$

# DQNs

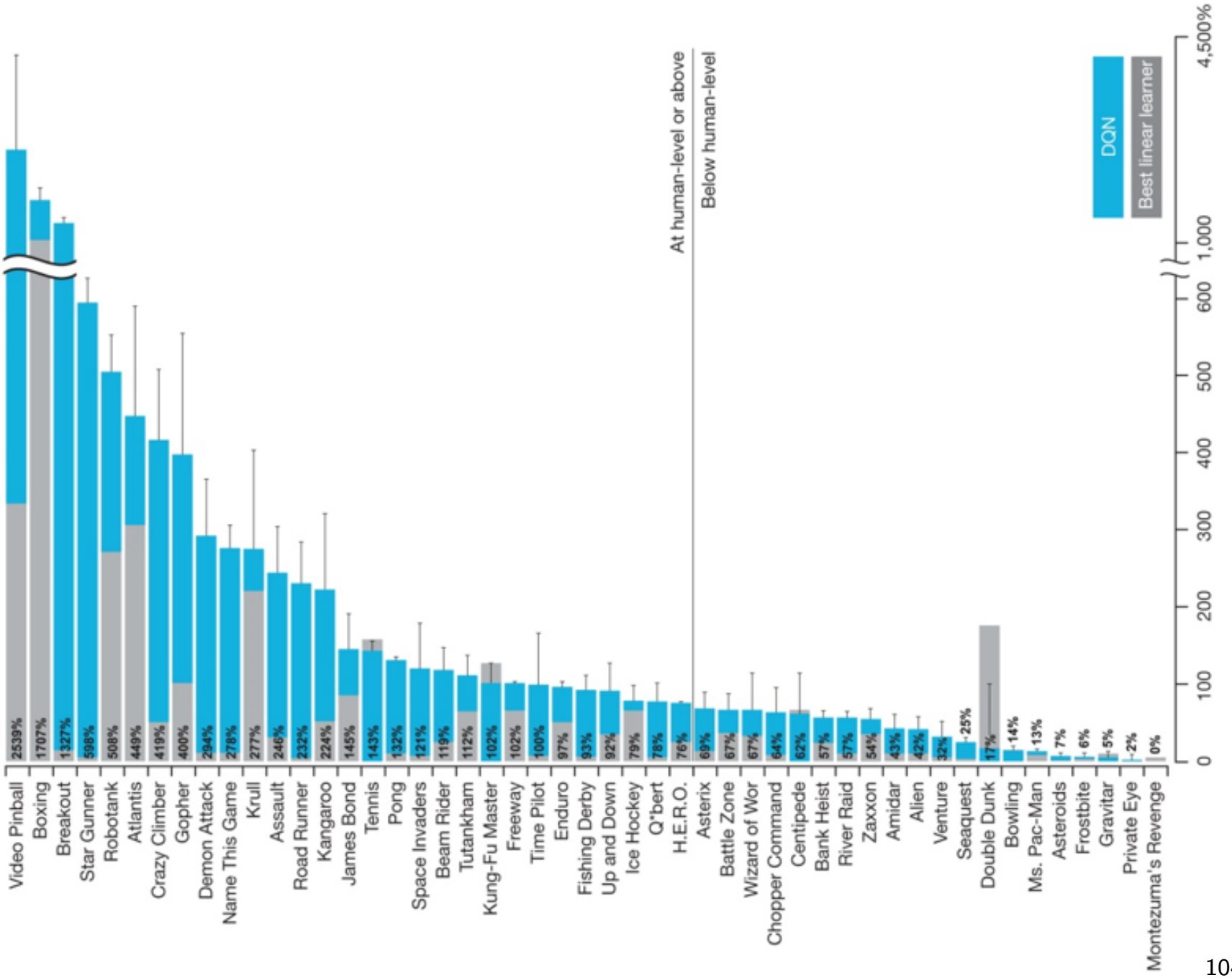
1: Initialize weights $w$ and $\overline{w}$ at random
2: Observe current state $s$
3: **loop**
4: Select action $a$ and execute it
5: Receive immediate reward $r$
6: Observe new state $s'$
7: Add $(s, a, r, s')$ to the replay buffer $\mathcal{D}$
8: Sample mini-batch of experiences from buffer $\mathcal{D}$
9: **for** each experience $(s, a, r, s')$ in mini-batch **do**
10: stochastic gradient descent to update weights **w**
11: **end for**
12: Update state: $s \leftarrow s'$
13: Every $c$ steps, update target: $\mathbf{w}^{-} \leftarrow \mathbf{w}$
14: **end loop**

# DQNs in Atari




- Human-level control through deep reinforcement learning, V Mnih et al, Nature 2015

# DQN Results in Atari



# Which Aspects of DQN were Important for Success?

| Game | Linear | Deep Network | DQN w/ fixed Q | DQN w/ replay | DQN w/replay and fixed Q |
|---|---|---|---|---|---|
| Breakout | 3 | 3 | 10 | 241 | 317 |
| Enduro | 62 | 29 | 141 | 831 | 1006 |
| River Raid | 2345 | 1453 | 2868 | 4102 | 7447 |
| Seaquest | 656 | 275 | 1003 | 823 | 2894 |
| Space Invaders | 301 | 302 | 373 | 826 | 1089 |

- Replay is hugely important

# Value-based Deep Reinforcement Learning

- Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
  - Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
  - Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)
- Combined algorithm: $3x$ mean Atari score vs Nature DQN

# Double DQN

- Remove maximization bias caused by $\max_a Q(s, a, w)$
- Current Q-network $\mathbf{w}$ is used to select actions
- Older Q-network $\mathbf{w}^-$ is used to evaluate actions

Action evaluation: w−

$$l = \left( r + \gamma Q(s', \underset{a'}{\operatorname{argmax}}\, Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

Action selection: w

# Double DQN

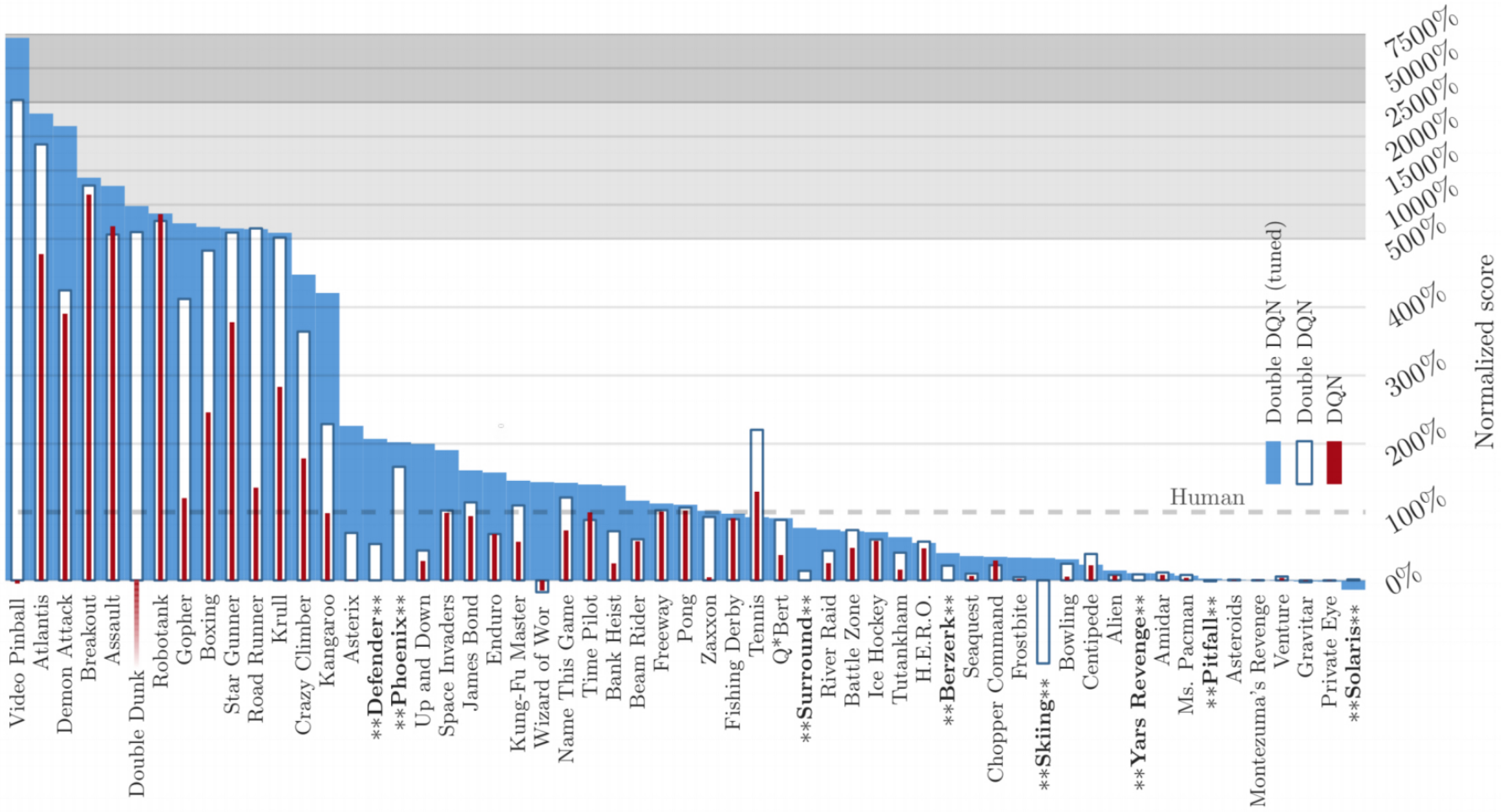



Figure: van Hasselt, Guez, Silver, 2015

# Prioritized Replay

- Weight experience according to surprise
- Priority of a tuple $i$ is proportional to DQN error

$$p_i \propto \left| r + \gamma \max_{a'} Q\left(s_{i+1}, a'; w^-\right) - Q\left(s_i, a_i; w\right) \right|$$

- Update $p_i$ every update
- Stochastic Prioritization

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

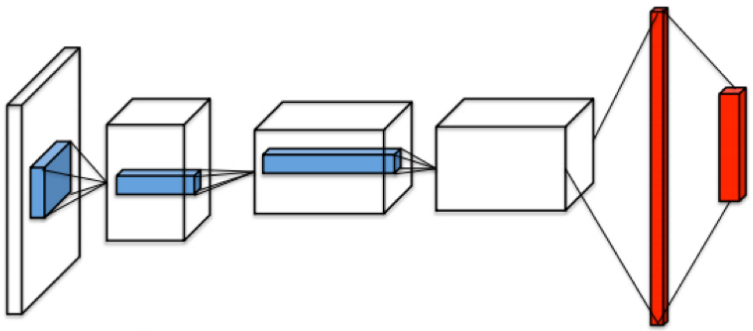
- $\alpha$ determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.

# Dueling Networks

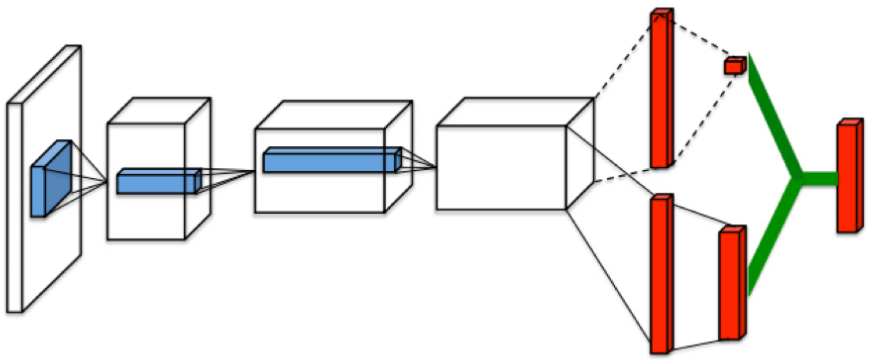
- Split Q-network into two channels
- Action-independent value function $V(s, v)$
- Action-dependent advantage function $A(s, a, w)$

$$Q(s, a) = V(s, v) + A(s, a, w)$$

# Dueling Networks vs. DQNs



DQN

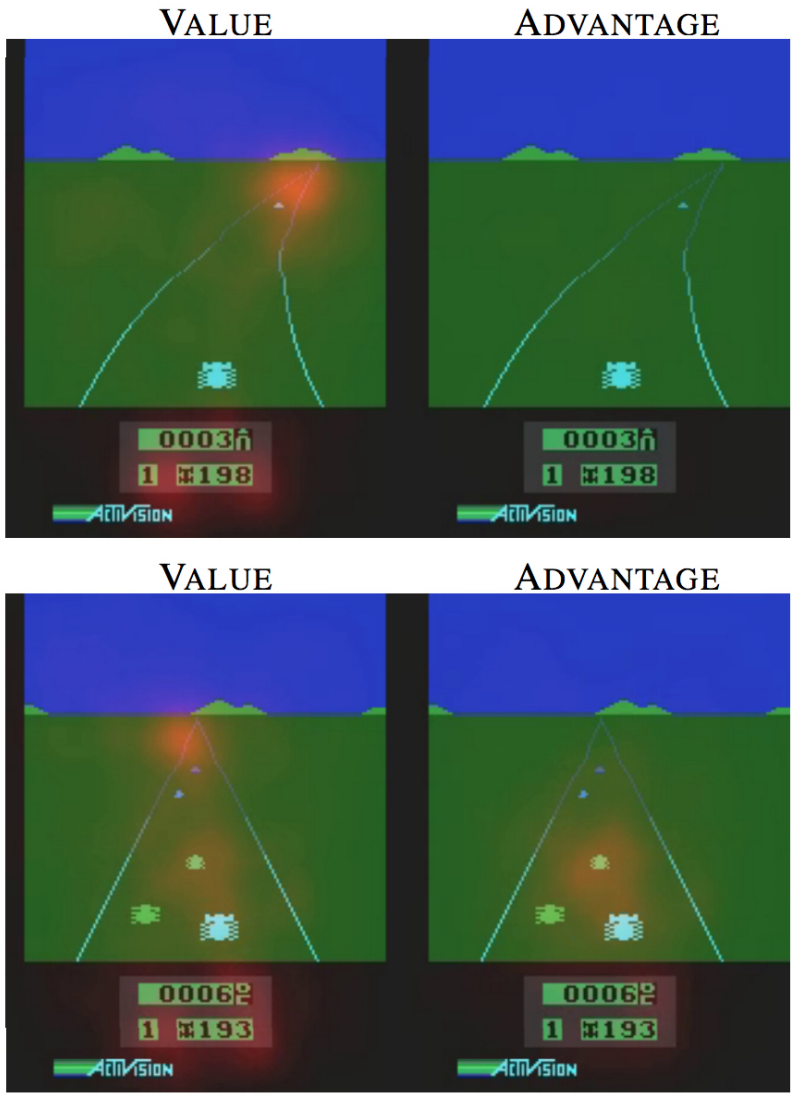


Dueling Networks

$$Q(s,a) = V(s,v) + A(s,a,\mathbf{w})$$

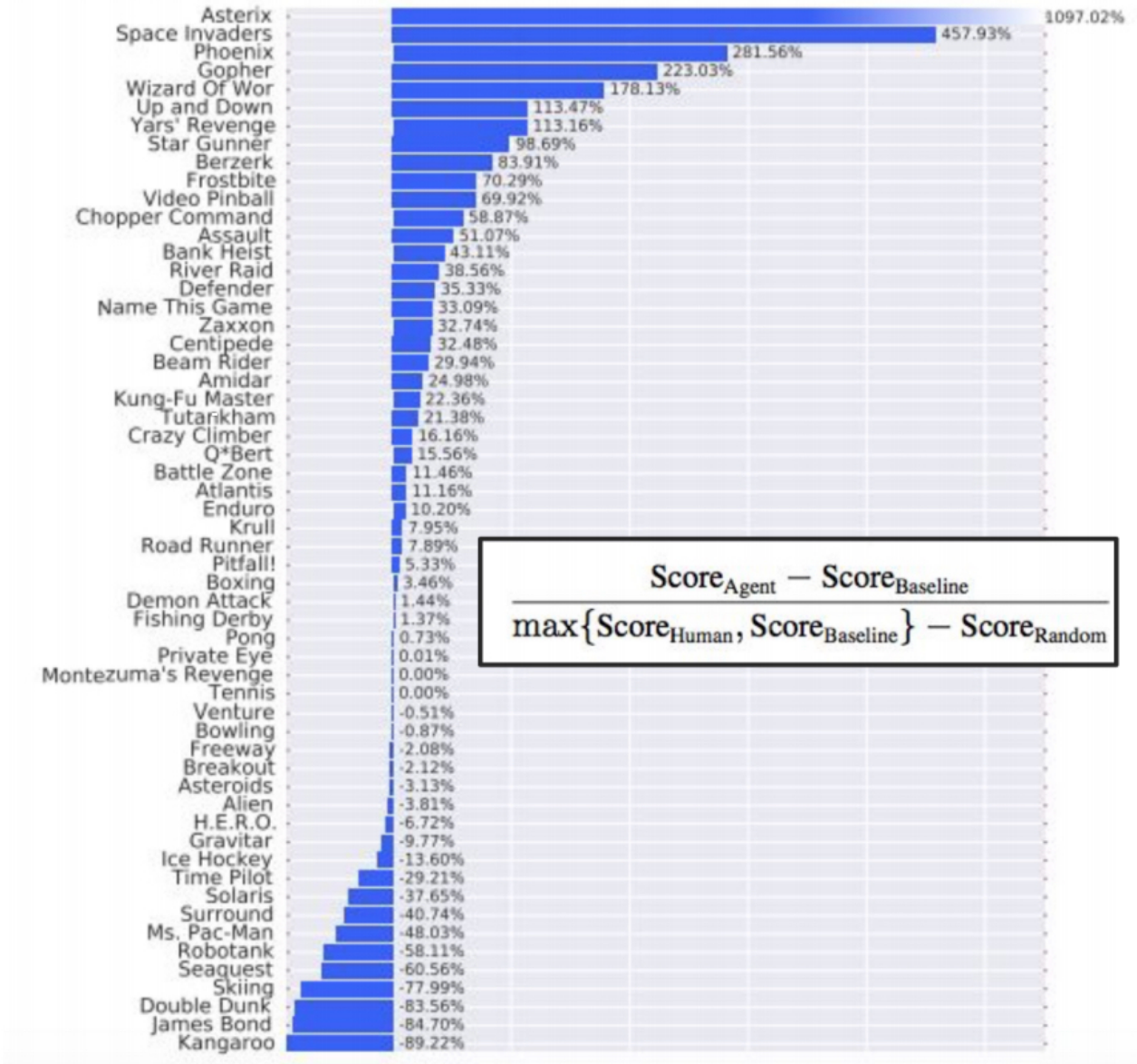
Wang et.al., ICML, 2016

# Dueling Networks




- The value stream learns to pay attention to the road
- The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions

# Dueling DQN V.S. Double DQN with Prioritized Replay



# Outline

# Policy-Based Reinforcement Learning

- So far, we approximated the value or action-value function using parameters

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- A policy was generated directly from the value function
  - e.g. using $\epsilon$-greedy
- Directly parametrize the policy

$$\pi_\theta(s,a) = \mathbb{P}[a|s,\theta]$$

- model-free reinforcement learning

# Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters $\theta$, find best $\theta$
- But how do we measure the quality of a policy $\pi_\theta$?
- In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments we can use the average value

$$J_{av}V(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

  - $d^{\pi_\theta}(s)$: stationary distribution of Markov chain for $\pi_\theta$

# Policy Gradient

- Let $J(\theta)$ be any policy objective function

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- $\alpha$ is a step-size parameter

# Score Function

- compute policy gradient of $\pi_\theta(s, a)$ compute the policy gradient analytically
- Assume policy $\pi_\theta$ is differentiable whenever it is non-zero

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

- The score function is $\nabla_\theta \log \pi_\theta(s, a)$

# One-Step MDPs

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- Compute the policy gradient

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \mathcal{R}_{s,a} \end{aligned}$$

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) \mathcal{R}_{s,a} \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) r\right] \end{aligned}$$

# Policy Gradient Theorem

- generalize to multi-step MDPs
- Replaces instantaneous reward $r$ with long-term value $Q^{\pi}(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

**Theorem**

*For any differentiable policy* $\pi_\theta(s, a)$,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a) \right]$$

# Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return $V_t$ as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

**function REINFORCE**
  Initialise $\theta$ arbitrarily
  **for** each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    **for** $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    **end for**
  **end for**
  **return** $\theta$
**end function**

# Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function,

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain two sets of parameters
  - Critic updates action-value function parameters $w$
  - Actor updates policy parameters $\theta$, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a) \right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$$

# Action-Value Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s, a) = \phi(s, a)^\top w$

  Critic Updates $w$ by linear TD(0)

  Actor Updates $\theta$ by policy gradient

**function** QAC

  Initialise $s$, $\theta$

  Sample $a \sim \pi_\theta$

  **for** each step **do**

    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$

    Sample action $a' \sim \pi_\theta(s', a')$

    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

    $w \leftarrow w + \beta \delta \phi(s, a)$

    $a \leftarrow a', s \leftarrow s'$

  **end for**

**end function**

# Inverse Reinforcement Learning

- Given state space, action space, transition model $P(s^{'}|s,a)$
- **No reward function** $R$
- Set of one or more teacher's demonstrations $(s_0, a_0, s_1, a_2, ...)$ (actions drawn from teacher's policy $\pi$)
- Goal: infer the reward function $R$
- With no assumptions on the optimality of the teacher's policy, what can be inferred about $R$?
- Now assume that the teacher's policy is optimal. What can be inferred about $R$?

# Max Entropy Inverse RL

- Maximizing the entropy of the distribution over the paths subject to the feature constraints from observed data implies we maximize the likelihood of the observed data under the maximum entropy (exponential family) distribution

$$P_\theta(\tau) = \frac{1}{Z} \exp(-c_\theta(\tau))$$

$$Z = \sum_\tau \exp(-c_\theta(\tau))$$

- Strong preference for low cost paths, equal cost paths are equally probable.

# Maximum Entropy Inverse RL

Notation

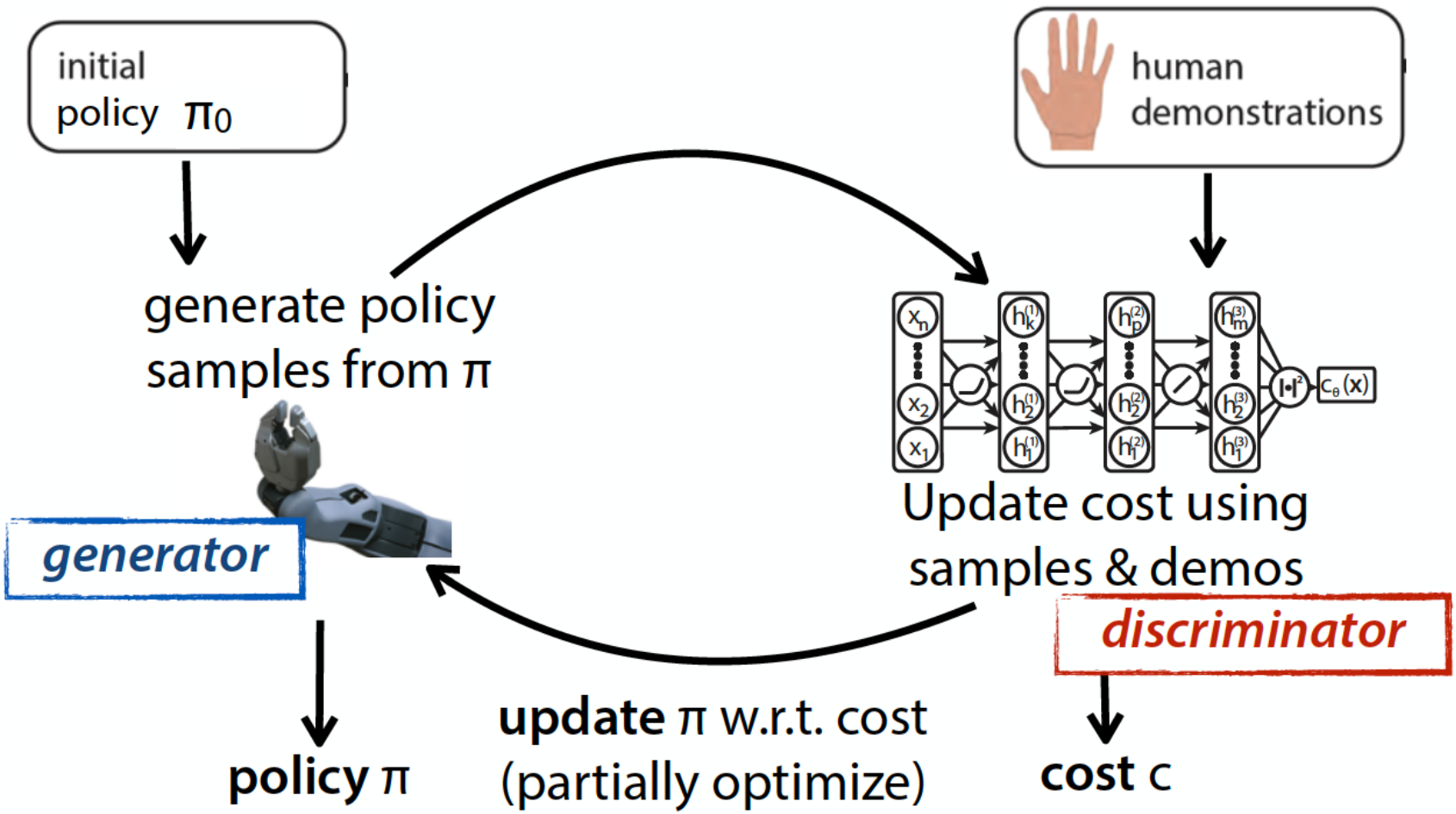
- $\tau = \{s_0, a_0, s_1, a_1, ...\}$
- $c_\theta$: cost with parameter $\theta$
  (linear case $c_\theta(\tau) = \theta^\mathsf{T} f_\tau = \sum_{s \in \tau} \theta^\mathsf{T} f_s$)
- $\mathcal{D}$: dataset of demonstrations $M = |\mathcal{D}|$
- $T$: transition dynamics

# Maximum Entropy Inverse RL

- Initialize $\theta$, gather demonstrations $\mathcal{D}$
- while
  - Solve for optimal policy $\pi(a|s)$ w.r.t. $c_\theta$ with value iteration
  - Solve for state visitation frequencies $p(s|\theta, T)$
  - Compute gradient $\nabla_\theta \mathcal{L} = \frac{1}{M} \sum_{\tau_d \in \mathcal{D}} f_{\tau_d} + \sum_s p(s|\theta, T) f_s$
  - Update $\theta$ with one gradient step using $\nabla_\theta \mathcal{L}$
- End until convergence

# Case Study: Guided Cost Learning

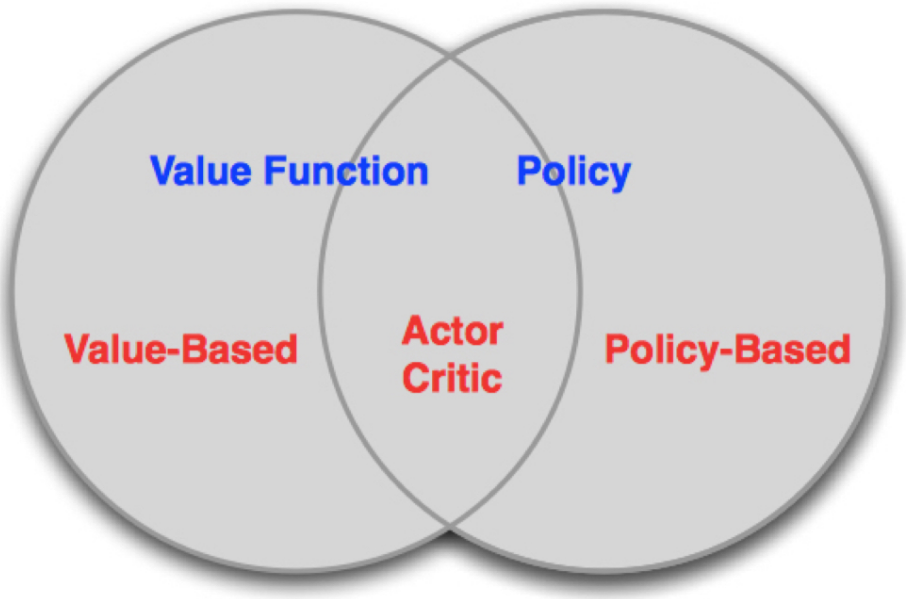
- Goals:
  - remove need to solve MDP in the inner loop
  - be able to handle unknown dynamics
  - handle continuous state & actions spaces



# Adaptive Importance Sampling

1: Initialize $q_k(\tau)$ as either a random initial controller or from demonstrations
2: **for** iteration $i = 1$ to $I$ **do**
3: Generate samples $\mathcal{D}_{\text{traj}}$ from $q_k(\tau)$
4: Append samples: $\mathcal{D}_{\text{samp}} \leftarrow \mathcal{D}_{\text{samp}} \cup \mathcal{D}_{\text{traj}}$
5: Use $\mathcal{D}_{\text{samp}}$ to update cost $c_\theta$ using Algorithm 2
6: Update $q_k(\tau)$ using $\mathcal{D}_{\text{traj}}$ and the method from (Levine & Abbeel, 2014) to obtain $q_{k+1}(\tau)$
7: **end for**
8: **return** optimized cost parameters $\theta$ and trajectory distribution $q(\tau)$

# Value-Based and Policy-Based RL




- Value Based
  - Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)
- Policy Based
  - No Value Function
  - Policy
- Actor Critic
  - Value Function
  - Policy

# Outline

# Multi-Agent Reinforcement Learning (MARL)

- Fundamentals of Game Theory
- MARL Algorithms
- Deep MARL Algorithms

# Fundamentals of Game Theory

- Models **strategic interactions** as games
- In **normal form games**, all players simultaneously select an action, and their joint action determines individual payoff
  - One-shot interaction
  - Can be represented as an $n$-dimensional payoff matrix, for $n$ players
- A player's **strategy** is defined as a probability distribution over his possible actions

# Example: Prisoner's Dilemma

- Two prisoners (A and B) commit a crime together
- They are questioned separately and can choose to confess or deny
  1. both confess: both prisoners will serve 3 years in jail
  2. both deny: both serve only 1 year for minor charges
  3. only one confesses: he goes free, while the other serves 5 years

| | $C$ | $D$ |
|---|---|---|
| $C$ | -3,-3 | -0,-5 |
| $D$ | -5,-0 | -1,-1 |

# Example: Prisoner's Dilemma

- What should they do?
- If both deny, their total penalty is lowest
  - But is this individually rational?
- Purely selfish: regardless of what the other player does, confess is the optimal choice
  1. If the other confesses, 3 instead of 5 years
  2. If the other denies, free instead of 1 year

| | $C$ | $D$ |
|---|---|---|
| $C$ | -3,-3 | -0,-5 |
| $D$ | -5,-0 | -1,-1 |

| | $C$ | $D$ |
|---|---|---|
| $C$ | -3,-3 | -0,-5 |
| $D$ | -5,-0 | -1,-1 |

# Strategy profiles

- **Nash equilibrium**
  - Individually optimal
  - No player can benefit in one-sidedly changing his strategy
  - Both confession
- **Pareto optimum**
  - Any other strategy change beneficial to one individual is detrimental to one or more others
  - Both denial

# Game Types

- **Competitive or zero-sum**
  - Players have opposing preferences
  - E.g. Matching Pennies, Rock paper scissors
- **Symmetric games**
  - Players are identical
  - E.g. Prioner's Dilemma, Rock paper scissors
- **Asymmetric games**
  - Players are unique
  - E.g. Battle of the Sexes

Matching Pennies

| | $C$ | $D$ |
|---|---|---|
| $C$ | +1,-1 | -1,+1 |
| $D$ | -1,+1 | +1,-1 |

Prisoner's Dilemma

| | $C$ | $D$ |
|---|---|---|
| $C$ | -3,-3 | -0,-5 |
| $D$ | -5,-0 | -1,-1 |

Battle of the Sexes

| | $C$ | $D$ |
|---|---|---|
| $C$ | 2,1 | 0,0 |
| $D$ | 0,0 | 1,2 |

# Matrix Game

- Recall: MDP is a single-agent, multiple state framework. $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
- **Matrix game** or **normal-form game** is a multi-agent, **stateless** framework $\langle n, A_{1\ldots n}, R_{1\ldots n} \rangle$
  - $n$ is the number of players,
  - $A_i$ is the set of actions available to player i
  - $A$ is the joint action space $A_1 \times \cdots \times A_n$
  - $\mathrm{R_i}$ is player i's payoff function $A_i \rightarrow \mathrm{R_i}$
- The players select actions from their available set and receive a payoff that depends on **all** the players' actions.

# 2-Player Zero-sum Matrix Game

- Policy: $\pi \in \mathrm{PD}(A)$ is a probability distribution over actions.
- Value: $V = \min_{o \in O} \sum_{a \in A} R_{o,a} \pi_a$ is Expected score of no matter which action the opponent chooses. $O$ denotes the opponent.
- Finding the $V^*$ and optimal policy $\pi^*$

$$V^* = \max_{\pi \in \mathrm{PD}(A)} \min_{o \in O} \sum_{a \in A} R_{o,a} \pi_a$$

  where $\sum_a R_{o,a} \pi_a$ expresses the expected reward to the agent for using policy $\pi$ against the opponent's action $o$
- can be solved by linear programming

# MARL Algorithms – Motivation

- Why not consider other agents as a part of the environment, and use single-agent algorithms, such as Q-learning?
  - The other learning agents cause more non-stationarity, making learning unstable
  - Cooperative behaviors of multiple agents are not easily learned by a single agent
  - ...
- Why not consider the agents as a single agent?
  - Reward functions are different for agents
  - Action space increases exponentially.
  - Centralised solutions are not useful in real applications: autonomous vehicles, traffic control

# Markov Games

- $n$ -player game: $\left\langle n, S, A^1, \ldots, A^n, \mathcal{R}^1, \ldots, \mathcal{R}^n, \mathcal{P} \right\rangle$
  - $S$ : set of states
  - $A^i$ : action set for player $i$
  - $\mathcal{R}^i$ : reward/payoff for player $i$
  - $\mathcal{P}$ : transition function
- The payoff function $\mathcal{R}^i : S \times A^1 \times \cdots \times A^n \mapsto \mathbb{R}$ maps the joint action $a = \left\langle a^1 \ldots a^n \right\rangle$ to an immediate payoff value for player $i$
- The transition function $\mathcal{P} : S \times A^1 \times \cdots \times A^n \mapsto \triangle(S)$ determines the probabilistic state change to the next state $s_{t+1}$

# Value Iteration for Zero-sum Markov Games

- Single agent MDP:

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi *}(s,a) \\ &= \max_{a \in A(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V^{\pi} \left( s' \right) \right] \end{aligned}$$

- 2-player zero-sum Markov game:

$$Q^* \left( s, \left\langle a^1, a^2 \right\rangle \right) = \mathcal{R} \left( s, \left\langle a^1, a^2 \right\rangle \right) + \gamma \sum_{s' \in S} \mathcal{P}_{s'} \left( s, \left\langle a^1, a^2 \right\rangle \right) V^* \left( s' \right)$$

$$V^*(s) = \max_{\pi \in \triangle(A^1)} \min_{a^2 \in A^2} \sum_{a^1 \in A^1} \pi_{a^1} Q^* \left( s, \left\langle a^1, a^2 \right\rangle \right)$$

# Minimax-Q for zero-sum Markov Games

- Similar to the single agent case, value iteration still requires knowledge of the environment
- Minimax-$Q$ (Littman94).
- Update rule for agent 1 with reward function $\mathcal{R}_t$ at stage $t$ :

$$Q_{t+1}\left(s_t, \left\langle a_t^1, a_t^2 \right\rangle\right) = (1-\alpha_t)\, Q_t\left(s_t, \left\langle a_t^1, a_t^2 \right\rangle\right) + \alpha_t \left[\mathcal{R}_t + \gamma V_t\left(s_{t+1}\right)\right]$$

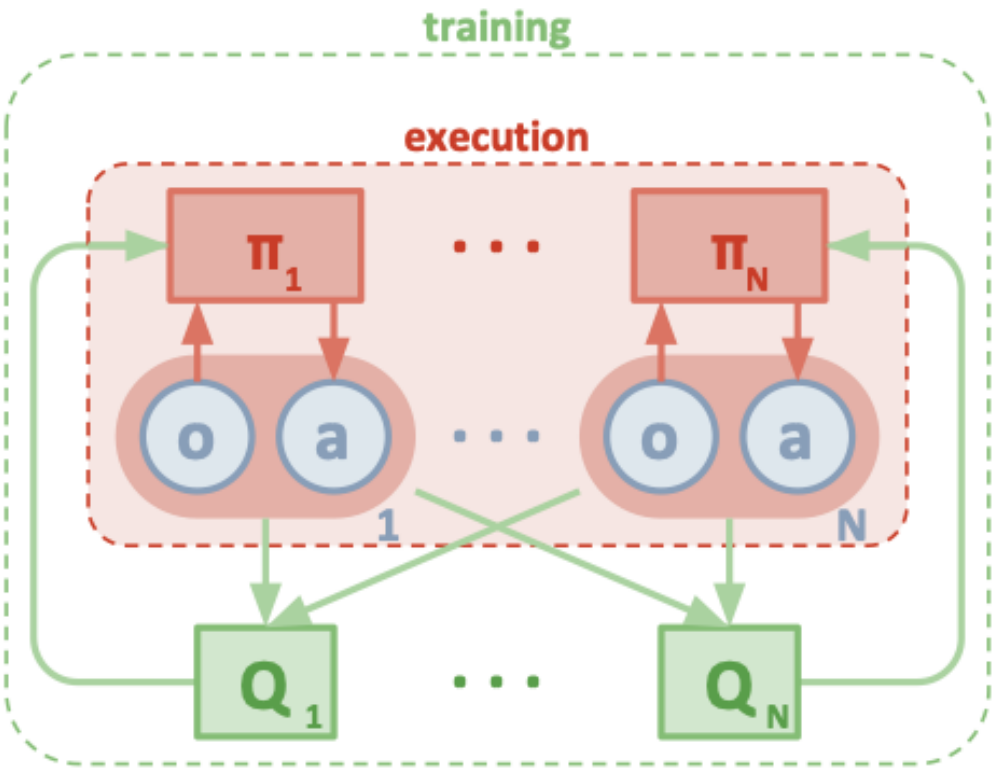
- The value of the next state $V\left(s_{t+1}\right)$ :

$$V_{t+1}(s) = \max_{\pi \in \triangle(A^1)} \min_{a^2 \in A^2} \sum_{a^1 \in A^1} \pi_{a^1} Q_t\left(s, \left\langle a^1, a^2 \right\rangle\right)$$

- Minimax-$Q$ **converges** to Nash equilibrium under the same assumptions as regular Q-learning [Littman94]

# Deep MARL – MADDPG

- A general-purpose multi-agent learning algorithm
  - Use **local information** (i.e. their own observations) at execution time. "Centralized training with decentralized execution"
  - Applicable not only to **cooperative** interaction but to **competitive** or **mixed** interaction



# Decentralized Actor Network

- Goal: find $\theta_i$ maximizing expected return for agent $i$

$$J(\theta_i) = \mathbb{E}[R_i]$$

- Gradient of $J(\theta_i)$

$$\nabla_{\theta_i} J(\pi_i) = \mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}} \left[ \nabla_{\theta_i} \pi_i(a_i | o_i) \nabla_{a_i} \left[ Q_i^{\pi}(\mathbf{x}, a_1, \ldots, a_N] |_{a_i = \pi_i(o_i)} \right] \right]$$

# Centralized Critic Network

- Centralized action-value function for agent $i$

$$Q_i^{\pi}\left(\mathbf{x}, a_1, \ldots, a_N\right)$$

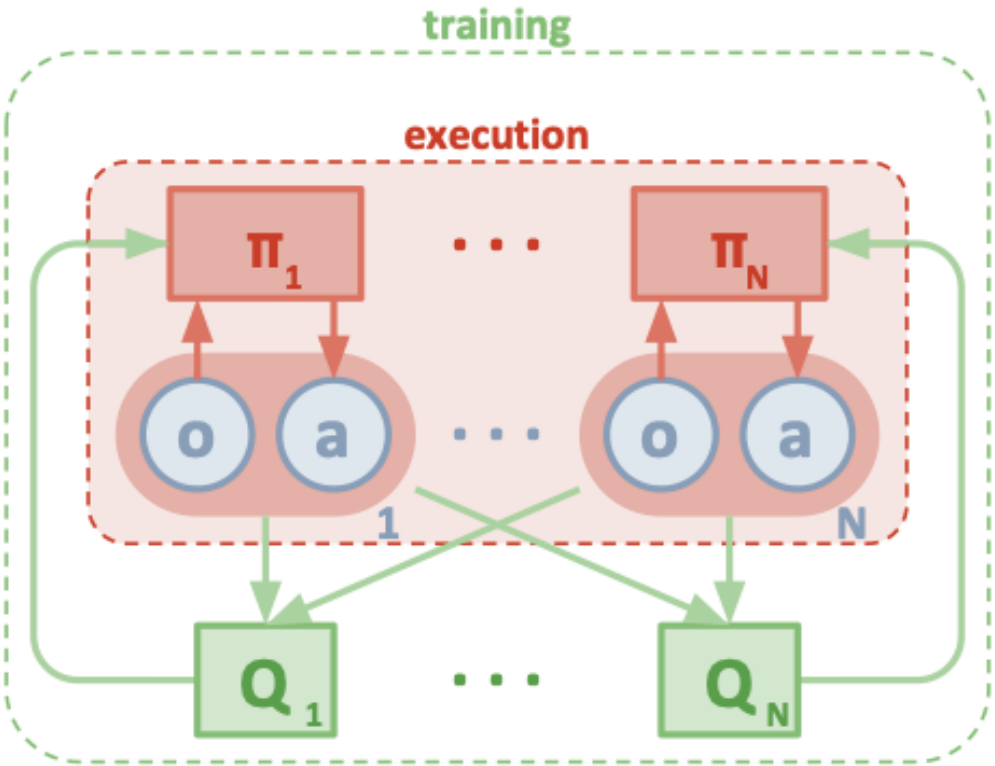
- The centralized action-value function is updated as:

$$\mathcal{L}\left(\theta_i\right) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}\left[\left(Q_i^{\pi}\left(\mathbf{x}, a_1, \ldots, a_N\right) - y\right)^2\right]$$

$$y = r_i + \gamma Q_i^{\pi'}\left(\mathbf{x}', a_1', \ldots, a_N'\right)\Big|_{a_j' = \pi_j'(o_j)}$$

# MADDPG

- Decentralized: $\pi_i\left(a_i|o_i\right)$
- Centralized: $Q_i^{\pi}\left(\mathbf{x}, a_1, \ldots, a_N\right)$



# Challenges of MARL

- Non-Unique Learning Goals
- Non-Stationarity
  - multiple agents usually learn concurrently
- Scalability Issue
  - The combinatorial nature of MARL (Hernandez-Leal et al., 2018). The dimension of joint action space increases exponentially with the number of agents
- Various Information Structures
  - Each agent has own observation and receives own rewards. These can be fully/partially/none visible to others. Direct communication between agents can improve the overall reward (such as the prisoner dilemma) and can be learned (Foerster et al. - 2016).

# Outline

# Recap GAN: Minimax Game

$$\min_G \max_D V(D,G) = \mathrm{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathrm{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]$$

- The generative model $G$ will map $\mathbf{z}$ to data space as $G(\mathbf{z};\theta_g)$, where $G$ is a differentiable function represented by a deep forward network with parameters $\theta_g$.
- Another DNN $D(\mathbf{x};\theta_d)$ outputs a single scalar and $D(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the $p_{data}(\mathbf{x})$ rather than $G(\mathbf{z};\theta_g)$.

**Goal**:Train $D$ to **maximize** the probability of assigning the correct label to both training examples and samples from $G$. simultaneously train $G$ to **minimize** that probability.

# Energy-based Models

Energy-based models associate an energy value $E_\theta(x)$ with a sample $x$, modeling the data as a Boltzmann distribution:

$$p_\theta(\mathbf{x}) = \frac{1}{Z} \exp\left(-E_\theta(\mathbf{x})\right)$$

And inverse RL is just a special case of an energy-based model, where $E_\theta(\mathbf{x}) = c_\theta(\tau)$, the cost function of a series of states and actions.

# Importance Sampling Estimation

Suppose the distribution of real data is $p(\tau)$, and we want to maximize the likelihood.

$$\begin{aligned}\mathcal{L}_{\text{cost}}(\theta) &= \mathbb{E}_{\tau\sim p}\left[-\log p_\theta(\tau)\right] = \mathbb{E}_{\tau\sim p}\left[c_\theta(\tau)\right] + \log Z \\ &= \mathbb{E}_{\tau\sim p}\left[c_\theta(\tau)\right] + \log\left(\mathbb{E}_{\tau\sim q}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{q(\tau)}\right]\right)\end{aligned}$$

The last step is an importance sampling (In the next equation, measure $\tau$ is not a probability measure, $\int_\Omega \tau = N$):

$$\begin{aligned}Z &= \sum_\tau \exp(-c_\theta(\tau)) = \int \exp(-c_\theta(\tau))d\tau = \int \frac{\exp(-c_\theta(\tau))}{q(\tau)} q(\tau)d\tau \\ &= \mathbb{E}_{\tau\sim q}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{q(\tau)}\right]\end{aligned}$$

This is an important improvement in *Guided Cost Learning*.

# Variance Reduction – Entropy Maximization

The optimal importance sampling distribution for estimating the partition function is $p_\theta$ itself, with zero variance.
During *Guided Cost Learning*, we minimize the KL divergence between $q(\tau)$ and $p_\theta(\tau)$, i.e., $KL(q(\tau)||p_\theta(\tau))$.

$$\mathcal{L}_{\text{sampler}}\left(q\right) = \mathbb{E}_{\boldsymbol{\tau}\sim q}\left[c_\theta(\boldsymbol{\tau})\right] + \mathbb{E}_{\boldsymbol{\tau}\sim q}[\log q(\boldsymbol{\tau})]$$

$q(\tau)$ is called **learned policy**, capable of generating samples from the demonstration distribution.
But if some modes in $p_\theta$ are not covered well by $q$, there will be a high variance. GCL uses a mixure of real data from $p(\tau)$ and data sampled from $q(\tau)$ as final importance sampling distribution ($\mu = \frac{1}{2}(p+q)$). Since we do not know $p(\tau)$, we need a rough estimation $\tilde{p}(\tau)$, maybe $\tilde{p}(\tau) = p_\theta(\tau)$.

$$\mathcal{L}_{\text{cost}}(\theta) = \mathbb{E}_{\tau\sim p}\left[c_\theta(\tau)\right] + \log\left(\mathbb{E}_{\tau\sim\mu}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\frac{1}{2}\tilde{p}(\tau) + \frac{1}{2}q(\tau)}\right]\right)$$

# Special GAN

An non-realistic assumption: the generated distribution $q(\tau)$ of G can be evaluated.
The discriminator is parameterized by EBM $\theta$ and outputs classified score:

$$D_\theta(\tau) = \frac{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right)}{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right) + q(\tau)}$$

Here $Z$ is another **free** parameter in D!

$$\begin{aligned}
\mathcal{L}_{\text{discriminator}}\ (D_\theta) &= \mathbb{E}_{\tau\sim p}\left[-\log D_\theta(\tau)\right] + \mathbb{E}_{\tau\sim q}\left[-\log\left(1 - D_\theta(\tau)\right)\right] \\
&= \mathbb{E}_{\tau\sim p}\left[-\log \frac{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right)}{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right) + q(\tau)}\right] \\
&+ \mathbb{E}_{\tau\sim q}\left[-\log \frac{q(\tau)}{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right) + q(\tau)}\right]
\end{aligned}$$

# Maximum Entropy IRL

If we substitute $\widetilde{p}(\tau) = p_\theta(\tau) = \frac{1}{Z}\exp\left(-c_\theta(\tau)\right)$,

$$\begin{aligned}\mathcal{L}_{\text{cost}}(\theta) &= \mathbb{E}_{\tau\sim p}\left[c_\theta(\tau)\right] + \log\left(\mathbb{E}_{\tau\sim\frac{1}{2}p+\frac{1}{2}q}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\frac{1}{2}\widetilde{p}(\tau)+\frac{1}{2}q(\tau)}\right]\right)\\ &= \mathbb{E}_{\tau\sim p}\left[c_\theta(\tau)\right] + \log\left(\mathbb{E}_{\tau\sim\mu}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\frac{1}{2Z}\exp\left(-c_\theta(\tau)\right)+\frac{1}{2}q(\tau)}\right]\right)\end{aligned}$$

Denote $\widetilde{\mu}(\tau) = \frac{1}{2Z}\exp\left(-c_\theta(\tau)\right) + \frac{1}{2}q(\tau)$ in the future.

# Connection 1

**Z estimates the partition function!**

$$\begin{aligned}
\mathcal{L}_{\mathrm{D}}\ (D_\theta) &= \mathbb{E}_{\tau\sim p}\left[-\log \frac{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right)}{\widetilde{\mu}(\tau)}\right] + \mathbb{E}_{\tau\sim q}\left[-\log \frac{q(\tau)}{\widetilde{\mu}(\tau)}\right] \\
&= \log Z + \mathbb{E}_{\tau\sim p}\left[c_\theta(\tau)\right] + \mathbb{E}_{\tau\sim p}[\log \widetilde{\mu}(\tau)] - \mathbb{E}_{\tau\sim q}[\log q(\tau)] \\
&+ \mathbb{E}_{\tau\sim q}[\log \widetilde{\mu}(\tau)] \\
&= \log Z + \mathbb{E}_{\tau\sim p}\left[c_\theta(\tau)\right] - \mathbb{E}_{\tau\sim q}[\log q(\tau)] + 2\mathbb{E}_{\tau\sim \mu}[\log \widetilde{\mu}(\tau)]
\end{aligned}$$

Optimizing by $Z$ will lead to:

$$\partial_Z \mathcal{L}_{\text{discriminator}}\ (D_\theta) = 0$$

$$\frac{1}{Z} = \mathbb{E}_{\tau\sim\mu}\left[\frac{\frac{1}{Z^2}\exp\left(-c_\theta(\tau)\right)}{\widetilde{\mu}(\tau)}\right]$$

$$Z = \mathbb{E}_{\tau\sim\mu}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\widetilde{\mu}(\tau)}\right]$$

Thus the minimizing Z is precisely the importance sampling estimate of the partition function.

# Connection 2

$c_\theta$ **optimizes the IRL objective!** For fixed Z, we optimize $\theta$:

$$\partial_\theta \mathcal{L}_{\mathsf{D}}\ (D_\theta) = \mathbb{E}_{\tau\sim p}\left[\partial_\theta c_\theta(\tau)\right] - \mathbb{E}_{\tau\sim\mu}\left[\frac{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right)\partial_\theta c_\theta(\tau)}{\tilde{\mu}(\tau)}\right]$$

On the other hand, when we differentiate the MaxEnt IRL objective, we obtain:

$$\begin{aligned}
\partial_\theta \mathcal{L}_{\text{cost}}(\theta) &= \mathbb{E}_{\tau\sim p}\left[\partial_\theta c_\theta(\tau)\right] + \partial_\theta \log\left(\mathbb{E}_{\tau\sim\mu}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\tilde{\mu}(\tau)}\right]\right) \\
&= \mathbb{E}_{\tau\sim p}\left[\partial_\theta c_\theta(\tau)\right] \\
&+ \left(\mathbb{E}_{\tau\sim\mu}\left[\frac{-\exp\left(-c_\theta(\tau)\right)\partial_\theta c_\theta(\tau)}{\widetilde{\mu}(\tau)}\right] / \mathbb{E}_{\tau\sim\mu}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\tilde{\mu}(\tau)}\right]\right) \\
&= \mathbb{E}_{\tau\sim p}\left[\partial_\theta c_\theta(\tau)\right] - \mathbb{E}_{\tau\sim\mu}\left[\frac{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right)\partial_\theta c_\theta(\tau)}{\tilde{\mu}(\tau)}\right] \\
&= \partial_\theta \mathcal{L}_{\mathsf{D}}\ (D_\theta)
\end{aligned}$$

# Connection 3

**The generator optimizes the importance sampler!**

$$
\begin{aligned}
\mathcal{L}_{\text{generator}}\left(q\right) &= \mathbb{E}_{\tau\sim q}[\log(1-D(\tau)) - \log(D(\tau))] \\
&= \mathbb{E}_{\tau\sim q}\left[\log\frac{q(\tau)}{\tilde{\mu}(\tau)} - \log\frac{\frac{1}{Z}\exp\left(-c_\theta(\tau)\right)}{\tilde{\mu}(\tau)}\right] \\
&= \mathbb{E}_{\tau\sim q}\left[\log q(\tau) + \log Z + c_\theta(\tau)\right] \\
&= \log Z + \mathbb{E}_{\tau\sim q}\left[c_\theta(\tau)\right] + \mathbb{E}_{\tau\sim q}[\log q(\tau)] \\
&= \log Z + \mathcal{L}_{\text{sampler}}\left(q\right)
\end{aligned}
$$

The term log Z is a parameter of the discriminator that is held fixed while optimizing the generator, this loss is exactly equivalent the sampler loss from MaxEnt IRL.

# Training EBM with GAN

We can derive an unbiased estimate of the partition function as

$$Z = \mathbb{E}_{\tau \sim \mu}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\widetilde{\mu}(\tau)}\right]$$

Update model parameter with SGD:

$$\mathcal{L}_{energy}(\theta) = \mathbb{E}_{\tau \sim p}\left[c_\theta(\tau)\right] + \log\left(\mathbb{E}_{\tau \sim \mu}\left[\frac{\exp\left(-c_\theta(\tau)\right)}{\frac{1}{2Z}\exp\left(-c_\theta(\tau)\right) + \frac{1}{2}q(\tau)}\right]\right)$$

Update sampling parameter with SGD:

$$\mathcal{L}_{\text{sampler}}\left(q\right) = \mathbb{E}_{\boldsymbol{\tau} \sim q}\left[c_\theta(\boldsymbol{\tau})\right] + \mathbb{E}_{\boldsymbol{\tau} \sim q}\left[\log q(\boldsymbol{\tau})\right]$$

# Bilevel Optimization

$$\phi^* = \arg\min_{\phi} F(\theta^*, \phi)$$
$$\theta^*(\phi) = \arg\min_{\theta} f(\theta, \phi)$$

- Also appears under the name "two time scale optimization"
- Naturally arises in adversarial settings in operation research
- NP-Hard even for bilevel linear programming

# Bilevel Optimization

- More than just GANs!
- Many reinforcement learning architectures have similar issues.
  - Actor-critic methods: Deep Deterministic Policy Gradients
  - Inverse reinforcement learning: Guided Cost Learning
  - Imitation learning: Generative Adversarial Imitation Learning
- All these models have close architectural similarities with GANs

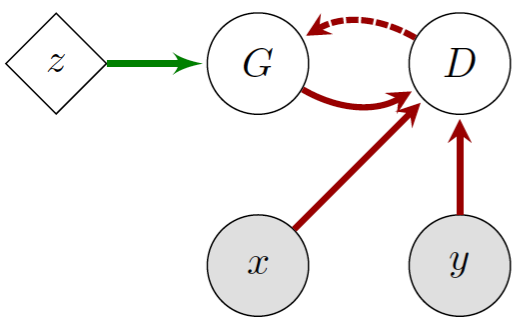
# GANs as Actor-Critic

- GAN:

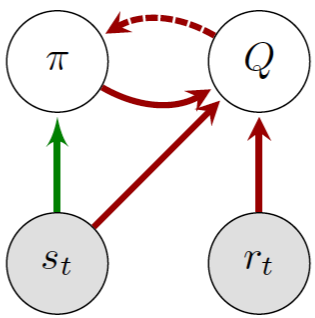
$$\mathcal{L}_D(\theta, \phi) = -\mathbb{E}_{p^*(x)} \left[\log D_\theta(x)\right] - \mathbb{E}_{q_\phi(x)} \left[\log(1 - D_\theta(x))\right]$$
$$\mathcal{L}_G(\theta, \phi) = -\mathbb{E}_{q_\phi(x)} \left[\log D_\theta(x)\right]$$

- Deep Deterministic Policy Gradients:

$$\mathcal{L}_Q(\theta, \phi) = \mathbb{E}_{\pi_\theta} \left[\|\mathbb{E}_{\pi_\theta} \left[r_t + \gamma Q_\phi(s_{t+1}, \pi_\theta(s_{t+1}))\right] - Q_\phi(s_t, \pi_\theta(s_t))\|^2\right]$$
$$\mathcal{L}_\pi(\theta, \phi) = -\mathbb{E}_{\pi_\theta} \left[Q_\phi(s_0, \pi_\theta(s_0))\right]$$

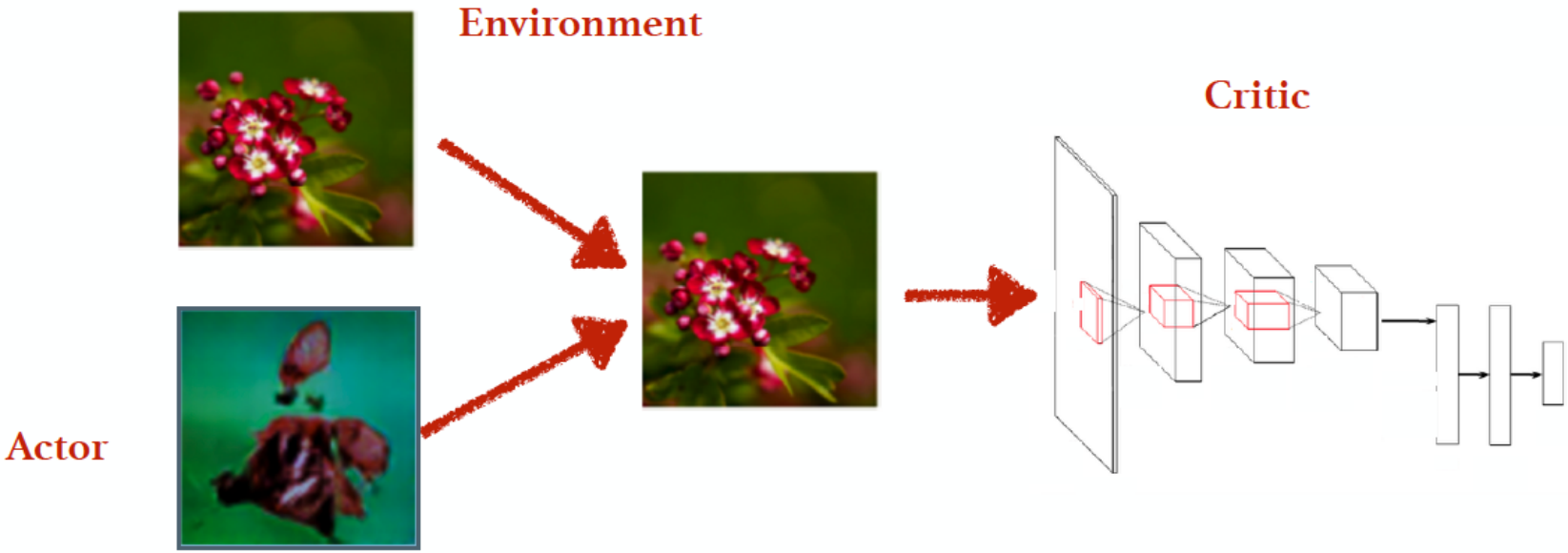



(a) Generative Adversarial Networks [4]




(b) Deterministic Policy Gradient [7] / SVG(0) [8] / Neurally-Fitted Q-learning with Continuous Actions [9]

# GANs as Actor-Critic




- Actor cannot see state - otherwise it could just copy environment
- Reward is not a function of action - actor and critic become adversarial

# Overview

# Used Materials

Disclaimer: Parts of the material and slides for this lecture were borrowed from Rich Sutton's book, David Silver's slides (UCL) and tutorial, Daan Bloembergen's ECML MARL Tutorial, Ruslan Salakhutdinov's slides (CMU) on Reinforcement Learning, Emma Brunskill's slides (Stanford) on Imitation Learning, Chelsea Finn's slides on Inverse Reinforcement Learning and Ming Ding's slides on A Connection between GAN, IRL and Energy-based Models.

# Thanks.

**HP:** http://keg.cs.tsinghua.edu.cn/jietang/
**Email:** jietang@tsinghua.edu.cn