

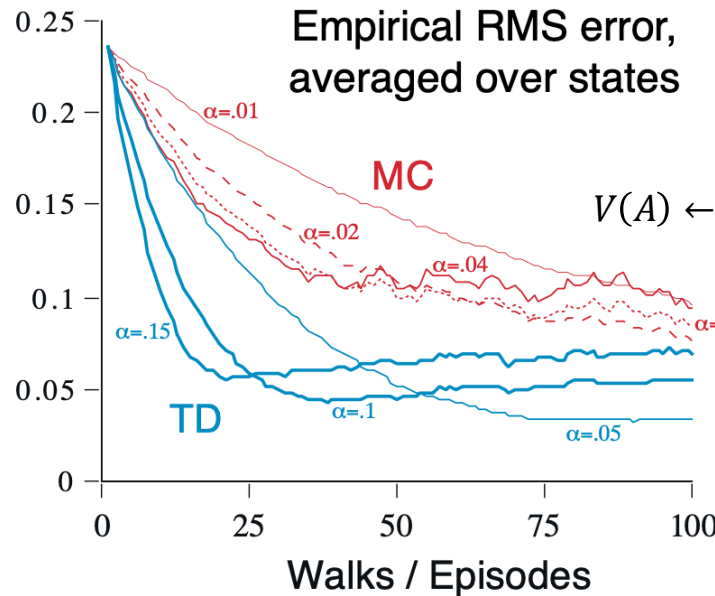
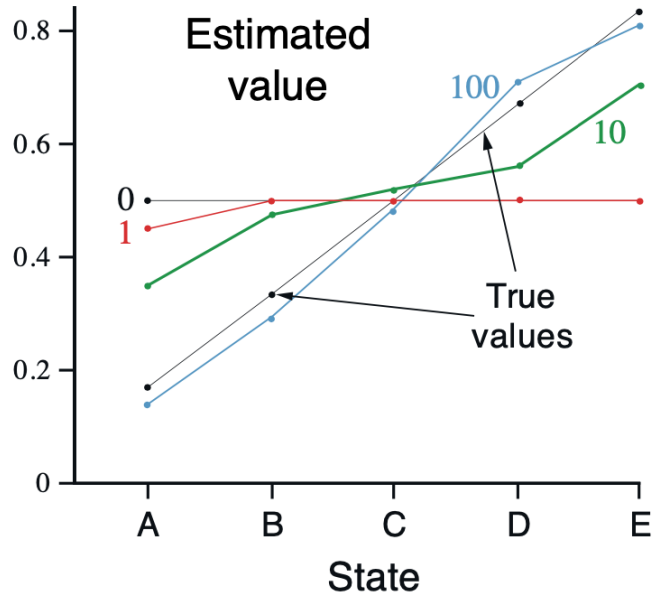
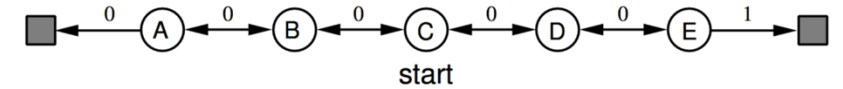
# Reinforcement Learning

---

## Lecture 4: Model-free Control

Christopher Mutschler

# Recap



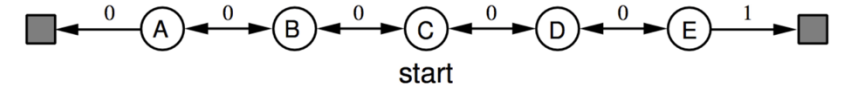
$$\gamma = 1 \quad \alpha = 0.1$$

$$V(s_t) \leftarrow V(s_t) + 0.1(r_{t+1} + V(s_{t+1}) - V(s_t))$$

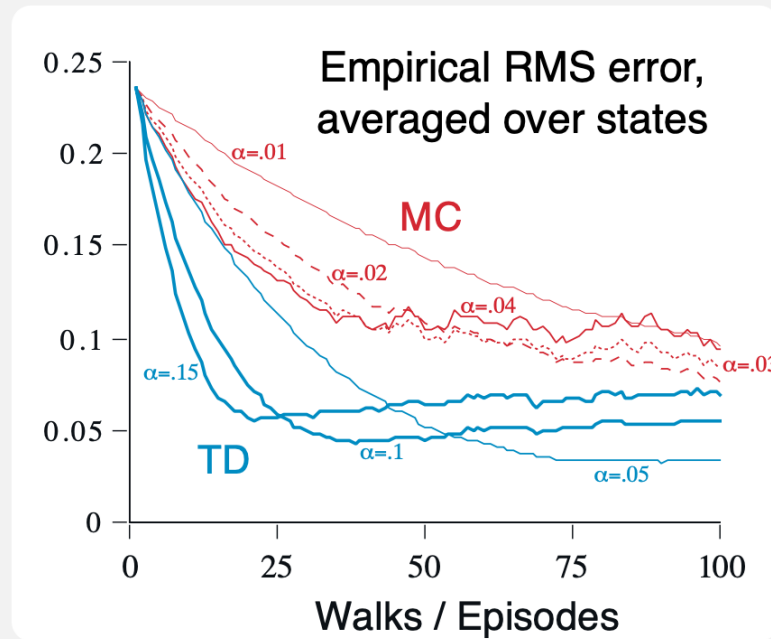
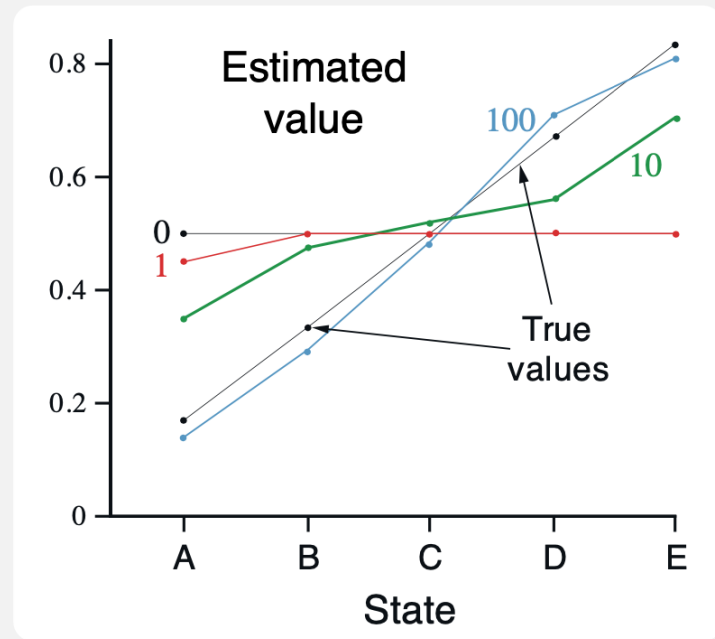
$$V(A) \leftarrow V(A) + 0.1(0 + 0 - V(A)) = 0.5 + 0.1(-0.5) = 0.45$$

*Exercise 6.3* From the results shown in the left graph of the random walk example it appears that the first episode results in a change in only  $V(A)$ . What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed?  $\square$

# Recap



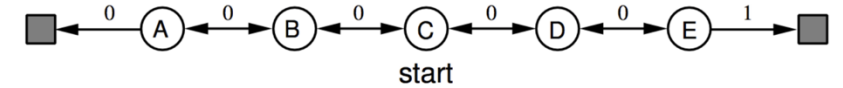
$$\gamma = 1 \quad \alpha = 0.1$$



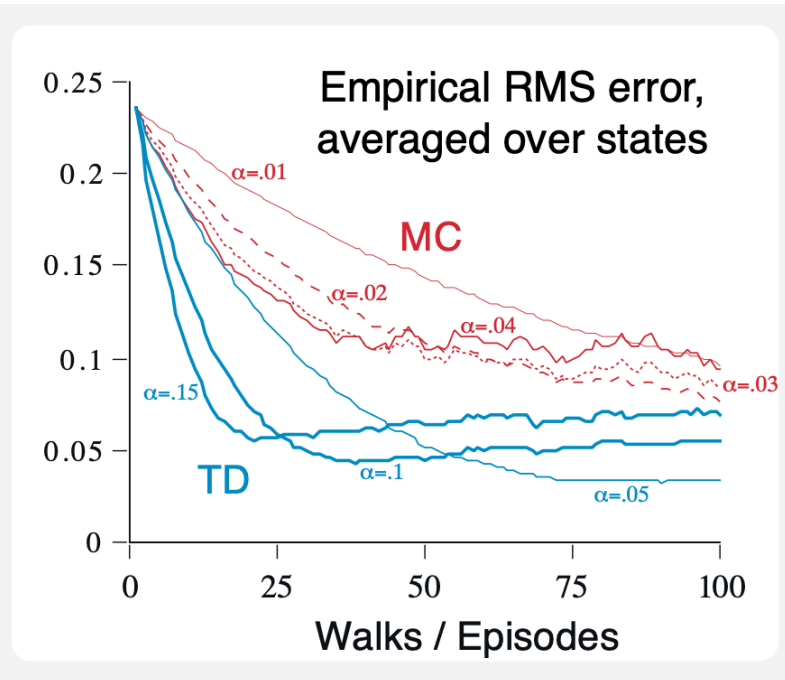
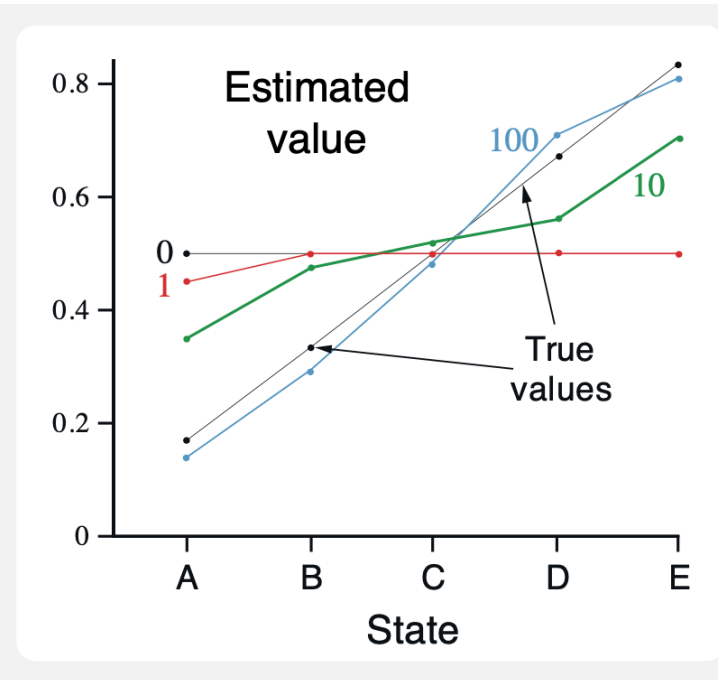
What about  
 $\alpha = \frac{1}{N(s)}$ ?

*Exercise 6.4* The specific results shown in the right graph of the random walk example are dependent on the value of the step-size parameter,  $\alpha$ . Do you think the conclusions about which algorithm is better would be affected if a wider range of  $\alpha$  values were used? Is there a different, fixed value of  $\alpha$  at which either algorithm would have performed significantly better than shown? Why or why not?  $\square$

# Recap

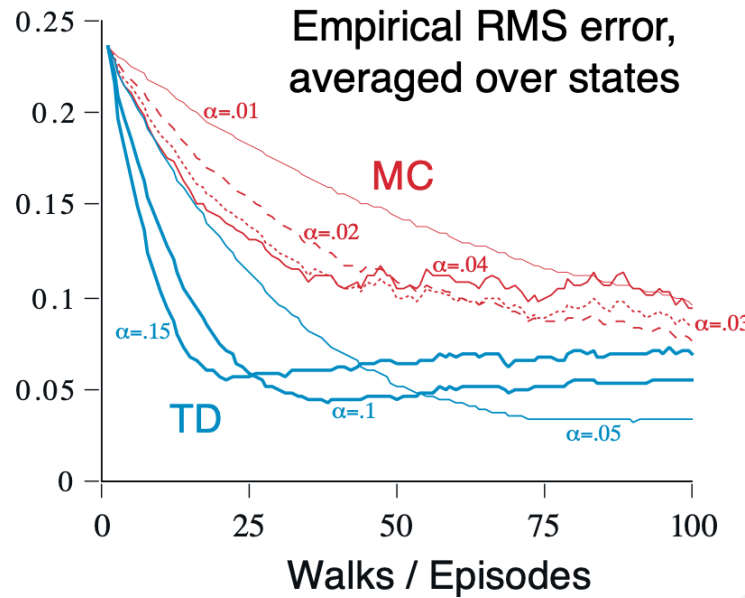
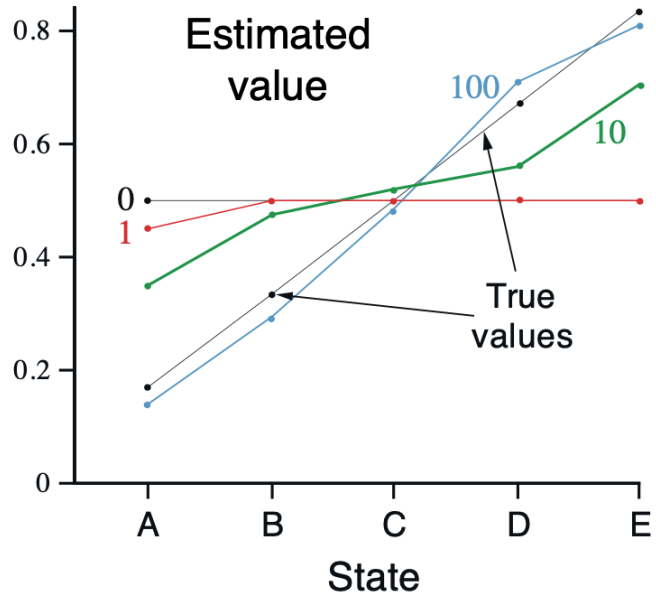
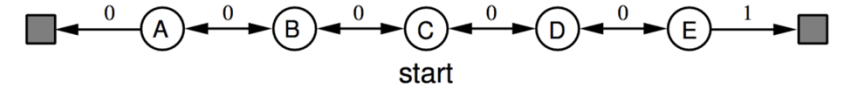


$$\gamma = 1 \quad \alpha = 0.1$$



*\*Exercise 6.5* In the right graph of the random walk example, the RMS error of the TD method seems to go down and then up again, particularly at high  $\alpha$ 's. What could have caused this? Do you think this always occurs, or might it be a function of how the approximate value function was initialized?

# Recap



$$\gamma = 1 \quad \alpha = 0.1$$

1. Value Iteration

2. e.g.:  $V(D) = \frac{1}{2} \cdot V(C) + \frac{1}{2} \cdot V(E)$

5 equations, 5 unknowns  $\rightarrow$  solve:

$$V(E) = \frac{V(D) + V(1)}{2}$$

$$V(D) = \frac{V(C) + V(E)}{2}$$

$$V(C) = \frac{V(B) + V(D)}{2}$$

$$V(B) = \frac{V(A) + V(C)}{2}$$

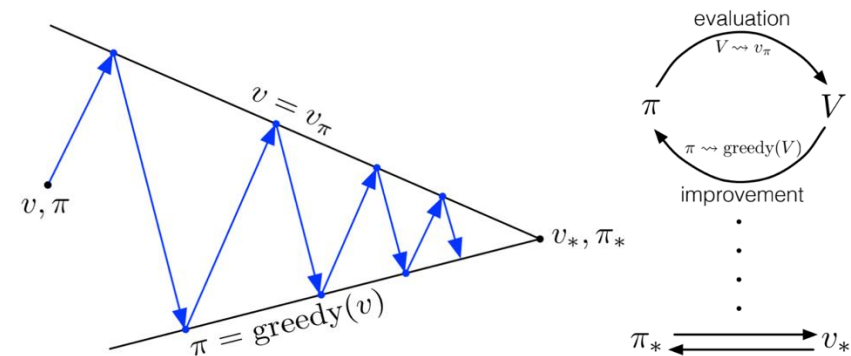
$$V(A) = \frac{0 + V(B)}{2}$$

*Exercise 6.6* In Example 6.2 we stated that the true values for the random walk example are  $\frac{1}{6}$ ,  $\frac{2}{6}$ ,  $\frac{3}{6}$ ,  $\frac{4}{6}$ , and  $\frac{5}{6}$ , for states A through E. Describe at least two different ways that these could have been computed. Which would you guess we actually used? Why?  $\square$

# Recap

## Dynamic Programming

- Dynamic Programming (DP) methods to find optimal controllers
  - DP methods are guaranteed to find optimal solutions for Q and V in polynomial time (in number of states and actions) and are exponentially faster than direct search
  - Policy Iteration computes the value function under a given policy to improve the policy while value iteration directly works on the states
    - Perform sweeps through the state set
    - Implement the Bellman equation update
    - Use bootstrapping
  - Require complete and accurate model of the environment
  - Have limited applicability in practice...
    - as they need to know the dynamics of the environment!



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

# Recap

## Monte Carlo and TD Methods

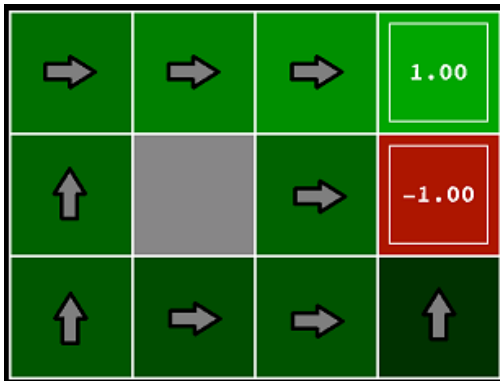
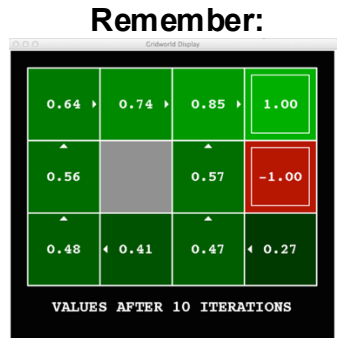
---

- So far: We know our MDP model  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ .
  - Planning by using dynamic programming
  - Solve a known MDP
- What if we don't know the model, i.e.,  $\mathcal{P}$  or  $\mathcal{R}$  or both?
- We distinguish between 2 problems for unknown MDPs:
  - **Model-free Prediction:** Evaluate the future, given the policy  $\pi$ .  
*(estimate the value function)*
  - **Model-free Control:** Optimize the future by finding the best policy  $\pi$ .  
*(optimize the value function)*

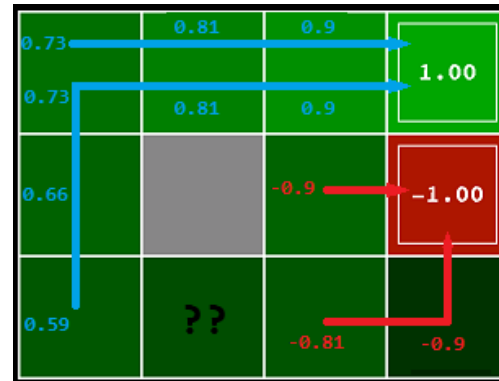
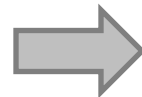
# Recap

## Monte Carlo and TD Methods

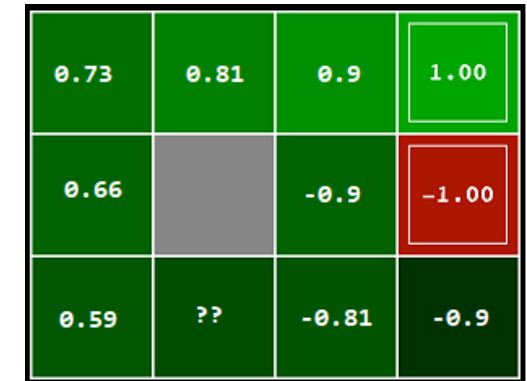
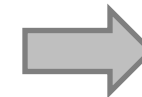
- Idea:
  - Use the samples to estimate the true V- and Q-value functions for the policy  $\pi$



Given Policy



Randomly select state  
and follow policy  
&  
Compute discounted  
return for each state



Average the  
values on each  
state

<https://medium.com/@zsalloum/monte-carlo-in-reinforcement-learning-the-easy-way-564c53010511>

# Recap

## Monte Carlo and TD Methods

- TD(0) vs. MC Policy Evaluation

- Goal: learn value function  $v_\pi$  online from experience when we follow policy  $\pi$

- Simplest TD learning algorithm: TD(0)
- Update value **towards estimation  $\hat{G}$** :

$$V(s) \leftarrow V(s) + \alpha(\hat{G} - V(s))$$
$$\hat{G} = r + \gamma V(s') \text{ (estimated return)}$$

- $\hat{G}$  is called the TD target
- $\hat{G} - V(s)$  is called the TD error.

- Update  $V(s)$  incrementally after each episode.
- For each state  $s$  with **actual return  $G$** :

$$N(s) \leftarrow N(s) + 1 \text{ (just increment visit counter)}$$
$$V(s) \leftarrow V(s) + \frac{1}{N(s)} (G - V(s)) \text{ (update a bit } \rightarrow \text{ reduce error)}$$

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

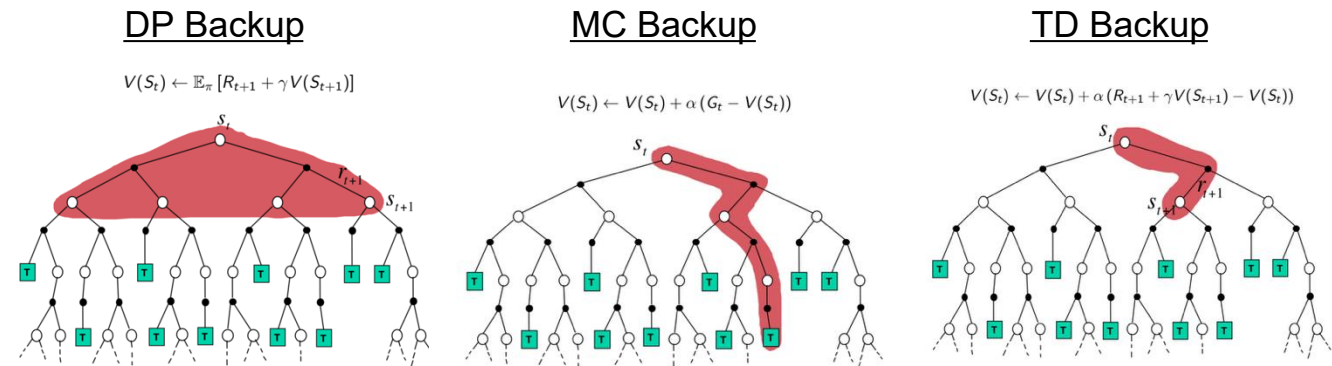
$$V(s) \leftarrow V(s) + \alpha(G - V(s)).$$

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} [\text{Target} - \text{OldEstimate}]$$

# Recap

## Monte Carlo and TD Methods

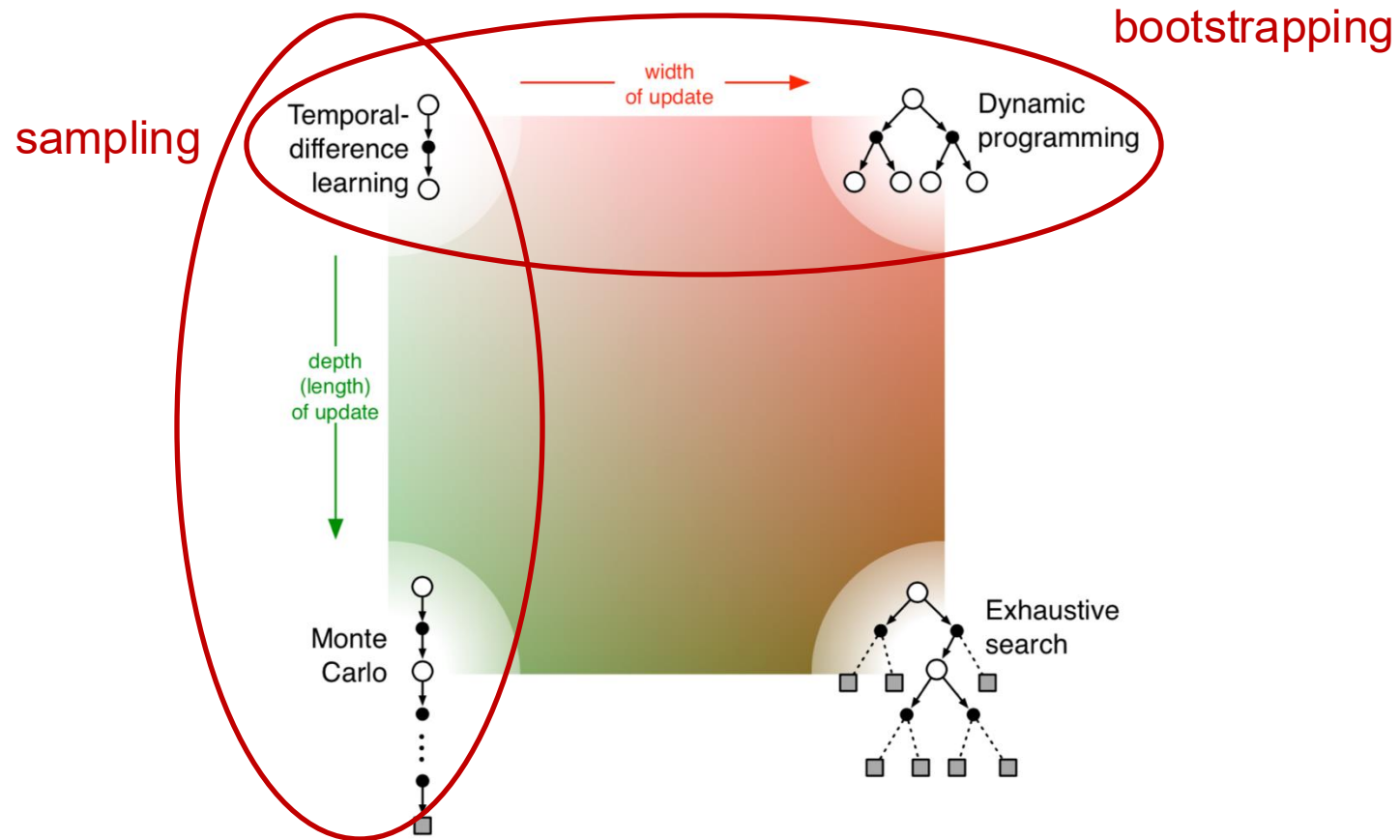
- Which one should I use? Does it make any difference?
  - Bias/Variance Trade-Off
  - MC has high variance, but zero bias
    - Good convergence (even with FA)
    - insensitive to initialization (no bootstrapping), simple to understand
  - TD has low variance, but some bias
    - TD(0) converges to  $\pi_{\nu}(s)$  (be careful with FA: bias is a risk)
    - sensitive to initialization (because of the bootstrapping)
    - Usually more efficient in practice



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

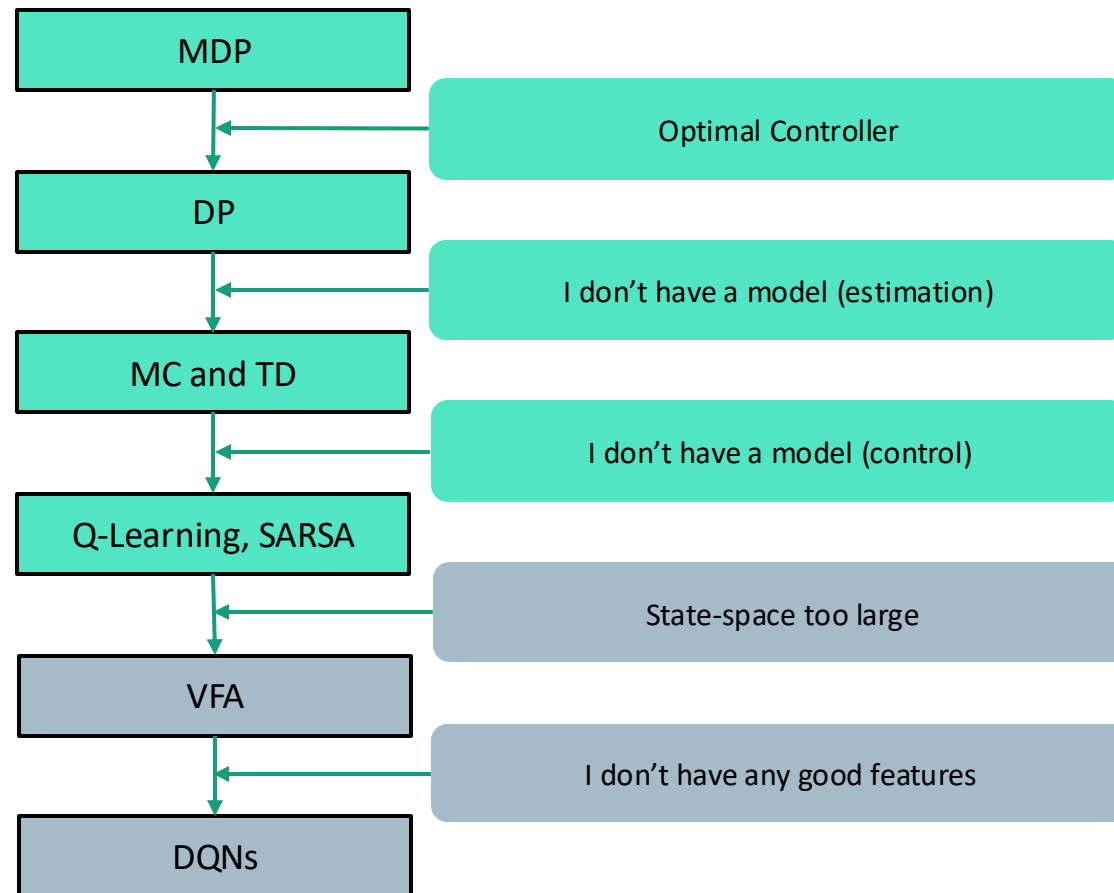
# Recap

## A Unified View of Prediction Algorithms



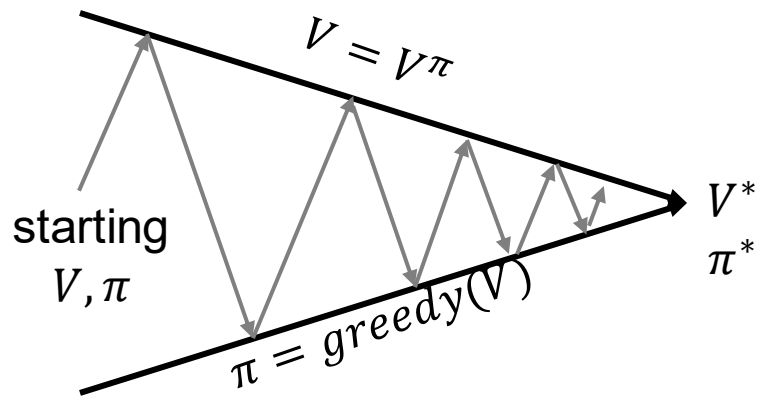
Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

# Overview



# Q-Learning and SARSA Algorithms

- The (model-free) control problem:
  - **Given** experience samples  $s(s, a, r, s')$
  - **Learn** a close-to optimal policy  $\pi$
- Simple idea:
  - If we have calculated the value function for a given policy  $\pi$  (e.g., from MC/TD policy evaluation from last week), we can use it for deriving a better policy  $\pi'$  through greedy policy improvement over  $V(s)$

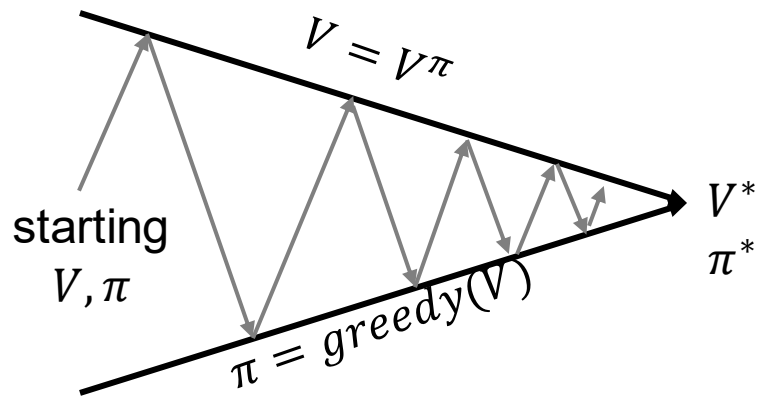


**Policy Evaluation:** Estimate  $V = v_\pi$   
e.g., Iterative Policy Evaluation

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g., Greedy Policy Improvement

# Q-Learning and SARSA Algorithms

- The (model-free) control problem:
  - **Given** experience samples  $s(s, a, r, s')$
  - **Learn** a close-to optimal policy  $\pi$
- Simple idea:
  - If we have calculated the value function for a given policy  $\pi$  (e.g., from MC/TD policy evaluation from last week), we can use it for deriving a better policy  $\pi'$  through greedy policy improvement over  $V(s)$



**Policy Evaluation:** Estimate  $V = v_\pi$   
e.g., Monte Carlo Policy Evaluation

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g., Greedy Policy Improvement

# Q-Learning and SARSA Algorithms

- The (model-free) control problem:
  - **Given** experience samples  $s(s, a, r, s')$
  - **Learn** a close-to optimal policy  $\pi$
- Simple idea:
  - If we have calculated the value function for a given policy  $\pi$  (e.g., from MC/TD policy evaluation from last week), we can use it for deriving a better policy  $\pi'$  through greedy policy improvement over  $V(s)$

$$\pi'(s) = \arg \max_{a \in A} \left\{ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\pi(s') \right\}, s \in S, V^{\pi'}(s) \geq V^\pi(s)$$

**Requires a model!**

- Greedy policy improvement over  $Q(s, a)$  is model-free

$$\pi'(s) = \arg \max_{a \in A} Q(s, a)$$

# Recap: Dynamic Programming

- How do we find optimal controllers for given (known) MDPs?
- Unfortunately, we need some definitions:
  - state-value function  $V$  for policy  $\pi$

$$s \xrightarrow{\pi(s), R_0} S_1 \xrightarrow{\pi(S_1), R_1} S_2 \xrightarrow{\pi(S_2), R_2} S_3 \dots S_{h-1} \xrightarrow{\pi(S_{h-1}), R_{h-1}} S_h$$
$$V^\pi(s) \triangleq Q^\pi(s, \pi(s)) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s \right].$$

- state-action-value function  $Q$  for policy  $\pi$

$$s \xrightarrow{a, R_0} S_1 \xrightarrow{\pi(S_1), R_1} S_2 \xrightarrow{\pi(S_2), R_2} S_3 \dots S_{h-1} \xrightarrow{\pi(S_{h-1}), R_{h-1}} S_h$$
$$Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s, A_0 = a \right].$$

# Recap: Dynamic Programming

- State-action-value function  $Q$  for policy  $\pi$

$$s \xrightarrow{a, r_0} s_1 \xrightarrow{\pi(s_1), r_1} s_2 \xrightarrow{\pi(s_2), r_2} s_3 \dots s_{h-1} \xrightarrow{\pi(s_{h-1}), r_{h-1}} s_h$$

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right]$$



# Recap: Dynamic Programming

- How do we find optimal controllers for given (known) MDPs?
- Unfortunately, we need some definitions:
  - Bellman Equation for  $Q$ , given policy  $\pi$

$$s \xrightarrow{a, \mathcal{R}(s, a)} s' \xrightarrow{\pi(s'), R_1} s_2 \xrightarrow{\pi(s_2), R_2} s_3 \dots s_{h-1} \xrightarrow{\pi(s_{h-1}), R_{h-1}} s_h$$

$$Q^\pi(s, a) = \underbrace{\mathcal{R}(s, a)}_{\text{first step}} + \gamma \underbrace{\sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) Q^\pi(s', \pi(s'))}_{\text{subsequent steps}}$$

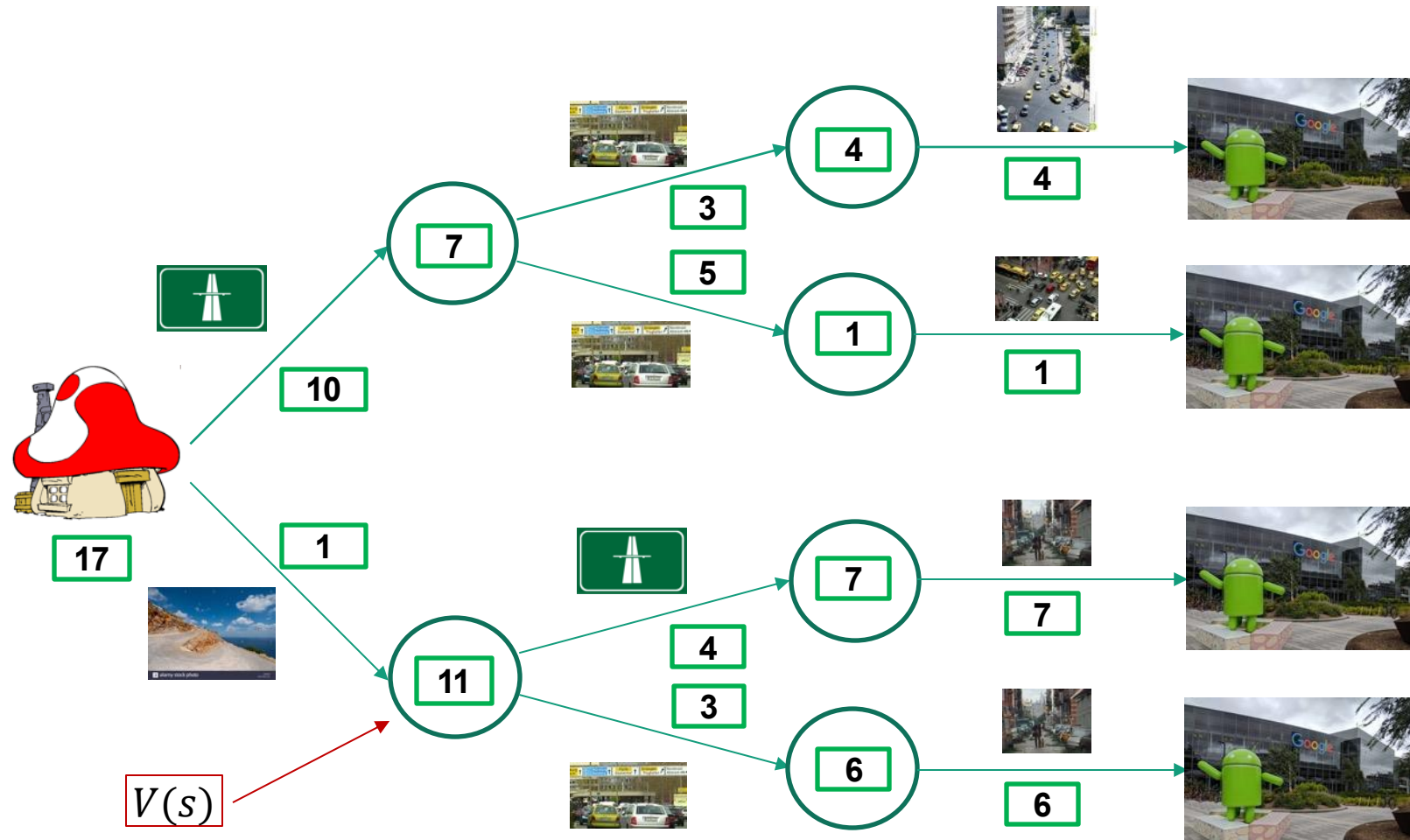
# Recap: Dynamic Programming

- How do we find optimal controllers for given (known) MDPs?
- Unfortunately, we need some definitions:
  - Bellman Optimality Equation for  $Q$

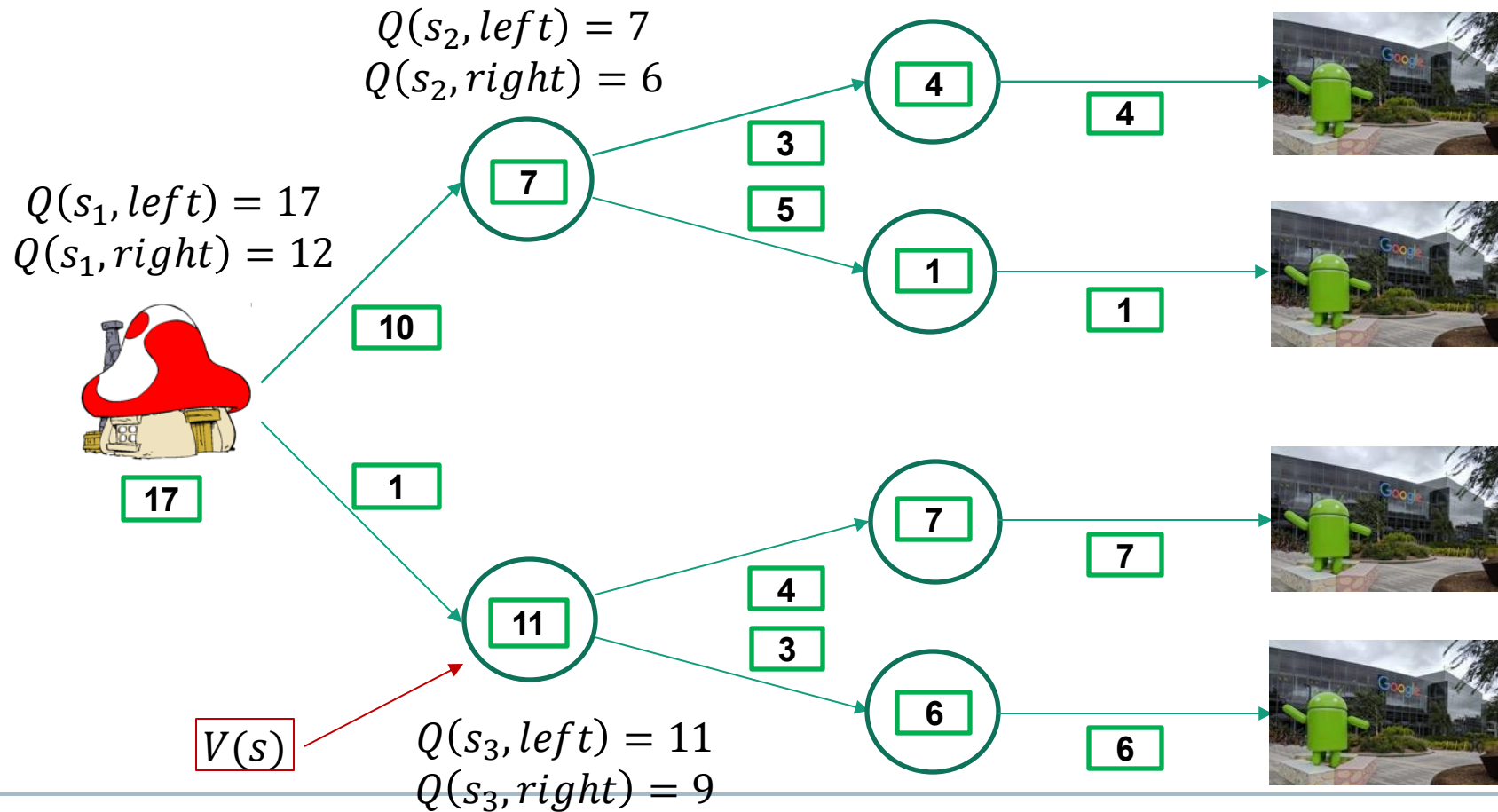
$$s \xrightarrow{a, \mathcal{R}(s, a)} s' \xrightarrow{\pi^*(s'), R_1} S_2 \xrightarrow{\pi^*(S_2), R_2} S_3 \dots S_{h-1} \xrightarrow{\pi^*(S_{h-1}), R_{h-1}} S_h$$

$$Q^{\pi^*}(s, a) = \underbrace{\mathcal{R}(s, a)}_{\text{first step}} + \gamma \underbrace{\sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \max_{a' \in \mathcal{A}} Q^{\pi^*}(s', a')}_{\text{subsequent steps}}$$

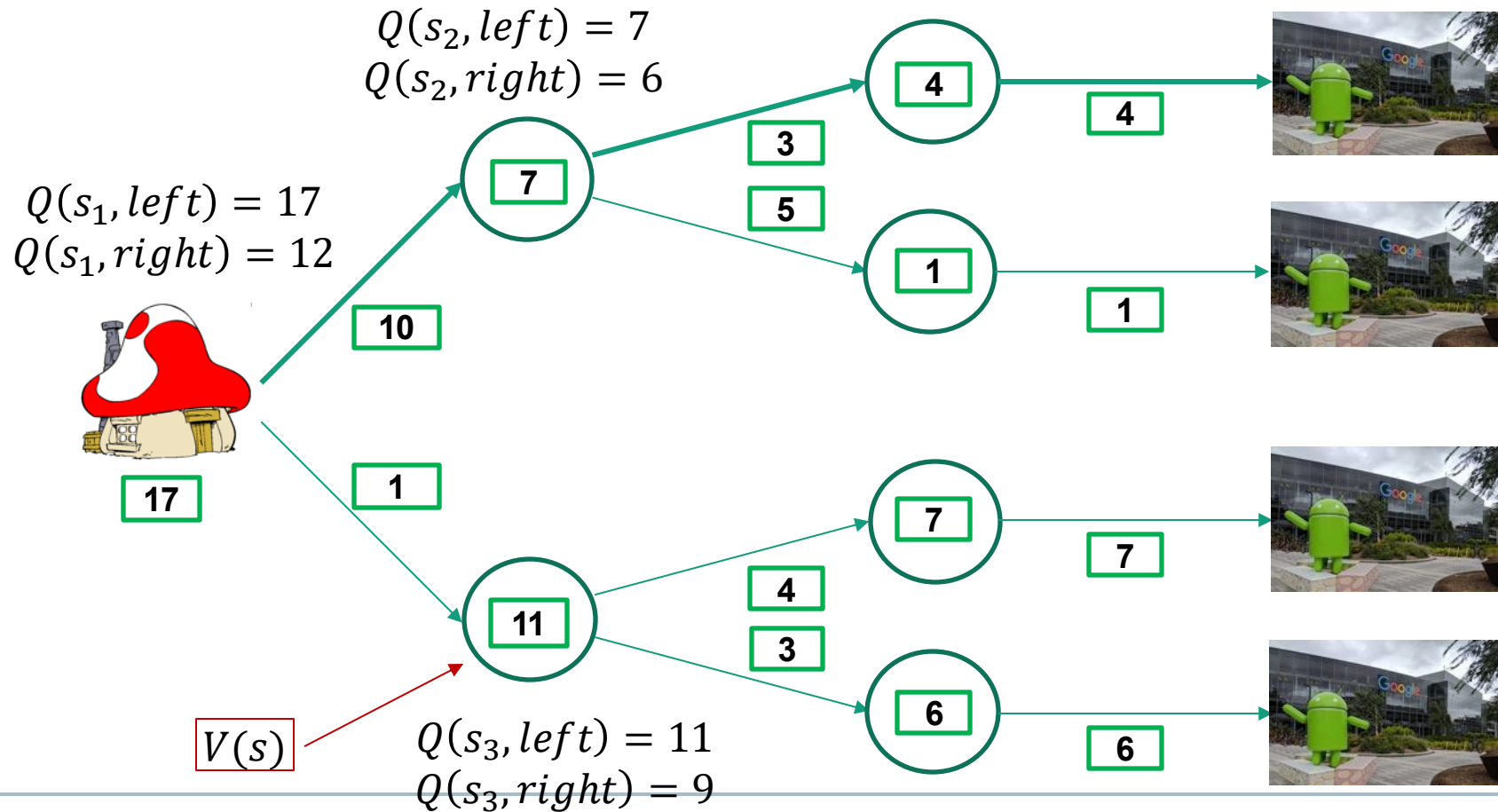
# Recap: Dynamic Programming



# Recap: Dynamic Programming



# Recap: Dynamic Programming



# Recap: Dynamic Programming

- How do we find optimal controllers for given (known) MDPs?
- Unfortunately, we need some definitions:
  - Greedy Policy Improvement over  $Q$

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^\pi(s, a)$$

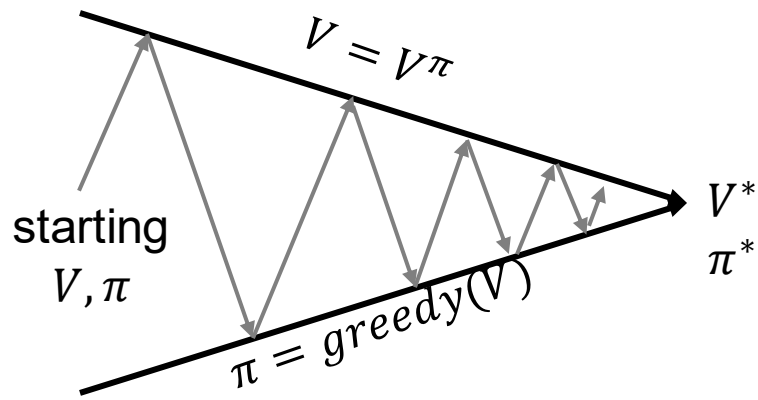
$$\forall s \in \mathcal{S}, \quad Q^{\pi'}(s, \pi'(s)) \geq Q^\pi(s, \pi(s))$$

- Greedy Policy Improvement over  $V$

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \left\{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) V^\pi(s') \right\}$$
$$\forall s \in \mathcal{S}, \quad V^{\pi'}(s') \geq V^\pi(s')$$

# Q-Learning and SARSA Algorithms

- The (model-free) control problem:
  - **Given** experience samples  $s(s, a, r, s')$
  - **Learn** a close-to optimal policy  $\pi$
- Simple idea:
  - If we have calculated the value function for a given policy  $\pi$  (e.g., from MC/TD policy evaluation from last week), we can use it for deriving a better policy  $\pi'$  through greedy policy improvement over  $V(s)$

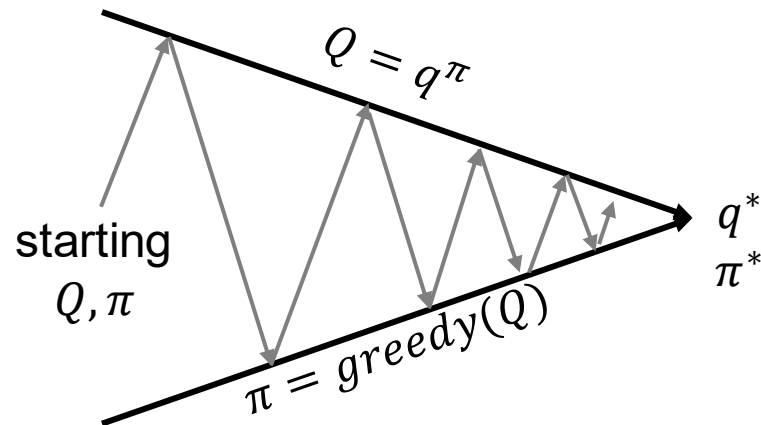


**Policy Evaluation:** Estimate  $V = v_\pi$   
e.g., Monte Carlo Policy Estimation

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g., Greedy Policy Improvement

# Q-Learning and SARSA Algorithms

- The (model-free) control problem:
  - **Given** experience samples  $s(s, a, r, s')$
  - **Learn** a close-to optimal policy  $\pi$
- Simple idea:
  - If we have calculated the value function for a given policy  $\pi$  (e.g., from MC/TD policy evaluation from last week), we can use it for deriving a better policy  $\pi'$  through greedy policy improvement over  $Q(s)$



**Policy Evaluation:** Estimate  $Q = q_\pi$   
e.g., Monte Carlo Policy Evaluation

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g., Greedy Policy Improvement over  $Q$

# Q-Learning and SARSA Algorithms

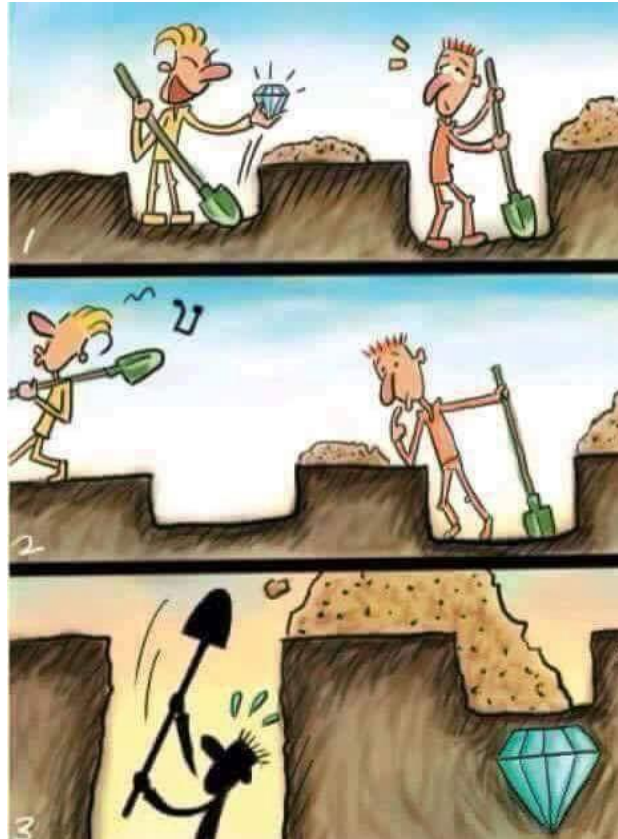
- Idea: your agent should not be afraid of trying something new 😊



# Q-Learning and SARSA Algorithms

## Exploration vs. Exploitation

---



<https://medium.com/deep-math-machine-learning-ai/ch-12-1-model-free-reinforcement-learning-algorithms-monte-carlo-sarsa-q-learning-65267cb8d1b4>

# Q-Learning and SARSA Algorithms

## Exploration vs. Exploitation

---



- There are two doors in front of you.
- You open the left door and get reward 0  
 $V(\text{left}) = 0$
- You open the right door and get reward +1  
 $V(\text{right}) = +1$
- You open the right door and get reward +3  
 $V(\text{right}) = +2$
- You open the right door and get reward +2  
 $V(\text{right}) = +2$
- $\vdots$
- Are you sure you've chosen the best door?

# Q-Learning and SARSA Algorithms

- Greedy policy improvement problems:
  - Executes the best action according to the estimated value function
  - Non-optimal initial choices may disorient exploration
  - Areas of the state space remain unexplored!

- Solution:  $\epsilon$ -greedy exploration

- Seems simplistic but very difficult to do better in practice!
- Either take the best action or explore the action space
- $\epsilon$  = “probability of exploration”

See “David Silver: Lectures on Reinforcement Learning. UCL Course on RL. 2015.”  
Lecture 5 on Control, slide 12 for a proof on this



- It can be proven that any  $\epsilon$ -greedy policy  $\pi'$  is an improvement over the  $\epsilon$ -greedy policy  $\pi$ ,  $v_{\pi'}(s) \geq v_{\pi}(s)$

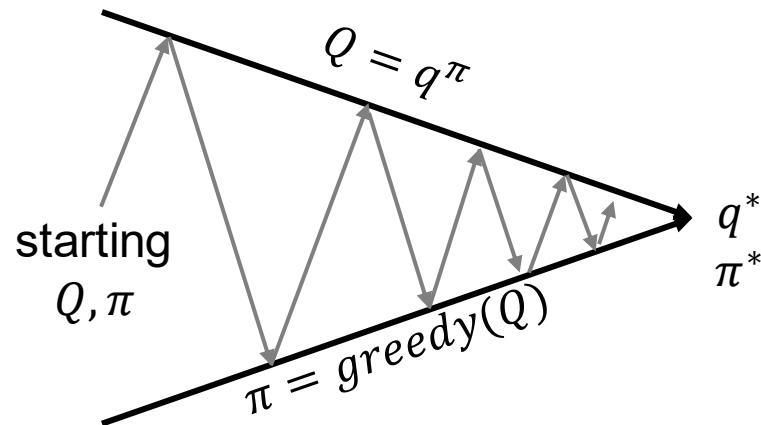
- GLIE MC Control:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1 \quad Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve policy based on new action-value function ( $\epsilon \leftarrow \frac{1}{k}, \pi \leftarrow \epsilon$ -greedy( $Q$ ))
- Converges to the optimal action-value function  $Q(S_t, A_t) \rightarrow q_*(s, a)$

# Q-Learning and SARSA Algorithms

- The (model-free) control problem:
  - **Given** experience samples  $s(s, a, r, s')$
  - **Learn** a close-to optimal policy  $\pi$
- Simple idea:
  - If we have calculated the value function for a given policy  $\pi$  (e.g., from MC/TD policy evaluation from last week), we can use it for deriving a better policy  $\pi'$  through greedy policy improvement over  $Q(s)$

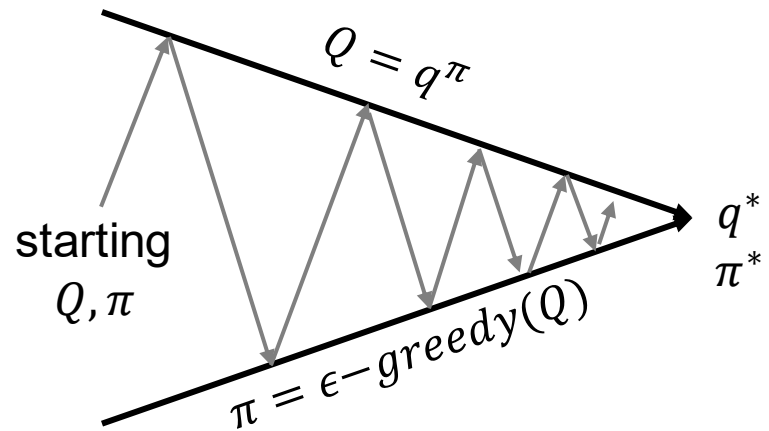


**Policy Evaluation:** Estimate  $Q = q_\pi$   
e.g., Monte Carlo Policy Evaluation

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g., Greedy Policy Improvement over  $Q$

# Q-Learning and SARSA Algorithms

- Greedy policy improvement problems:
  - Executes the best action according to the estimated value function
  - Non-optimal initial choices may disorient exploration
  - Areas of the state space remain unexplored!
- Solution:  $\epsilon$ -greedy exploration

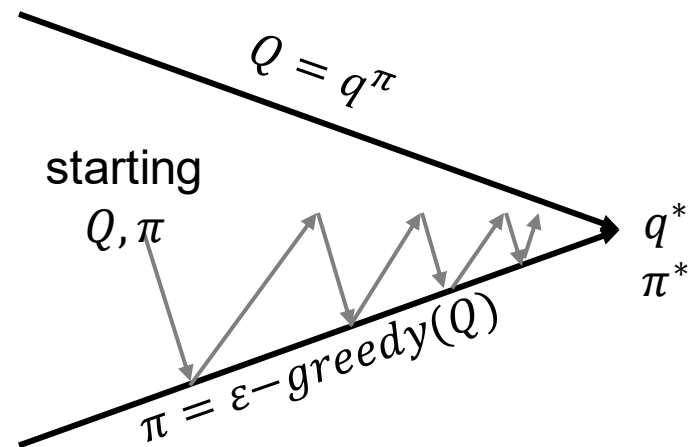


**Policy Evaluation:** Estimate  $Q = q_\pi$   
e.g., Monte Carlo Policy Evaluation

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g.,  $\epsilon$ -greedy Policy Improvement over  $Q$

# Q-Learning and SARSA Algorithms

- Greedy policy improvement problems:
  - Executes the best action according to the estimated value function
  - Non-optimal initial choices may disorient exploration
  - Areas of the state space remain unexplored!
- Solution:  $\epsilon$ -greedy exploration



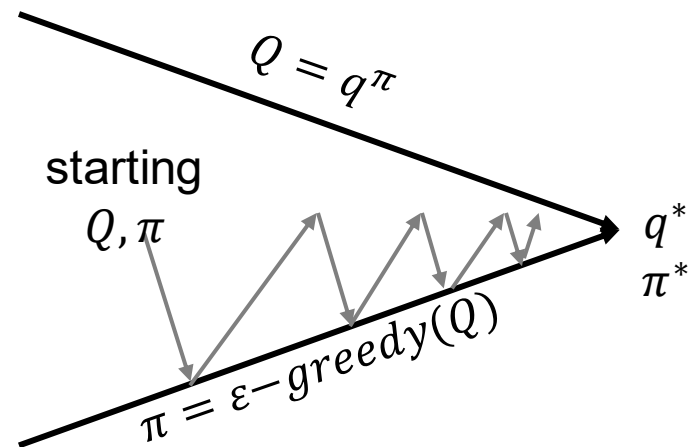
Every episode:

**Policy Evaluation:** Estimate  $Q \approx q_\pi$   
e.g., Monte Carlo Policy Evaluation

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g.,  $\epsilon$ -greedy Policy Improvement over  $Q$

# Q-Learning and SARSA Algorithms

- Greedy policy improvement problems:
  - Executes the best action according to the estimated value function
  - Non-optimal initial choices may disorient exploration
  - Areas of the state space remain unexplored!
- Solution:  $\epsilon$ -greedy exploration



Every time step:

**Policy Evaluation:** Estimate  $Q \approx q_\pi$   
e.g., SARSA

**Policy Improvement:** Generate  $\pi' \geq \pi$   
e.g.,  $\epsilon$ -greedy Policy Improvement over  $Q$

# Q-Learning and SARSA Algorithms

## SARSA algorithm (on-policy control)

- Apply TD to  $Q(s, a)$
- Use  $\epsilon$ -greedy policy improvement
- Update at every time-step

### Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\epsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

  Initialize  $S$

  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

  Loop for each step of episode:

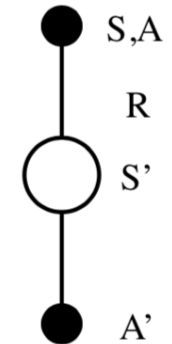
    Take action  $A$ , observe  $R, S'$

    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -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

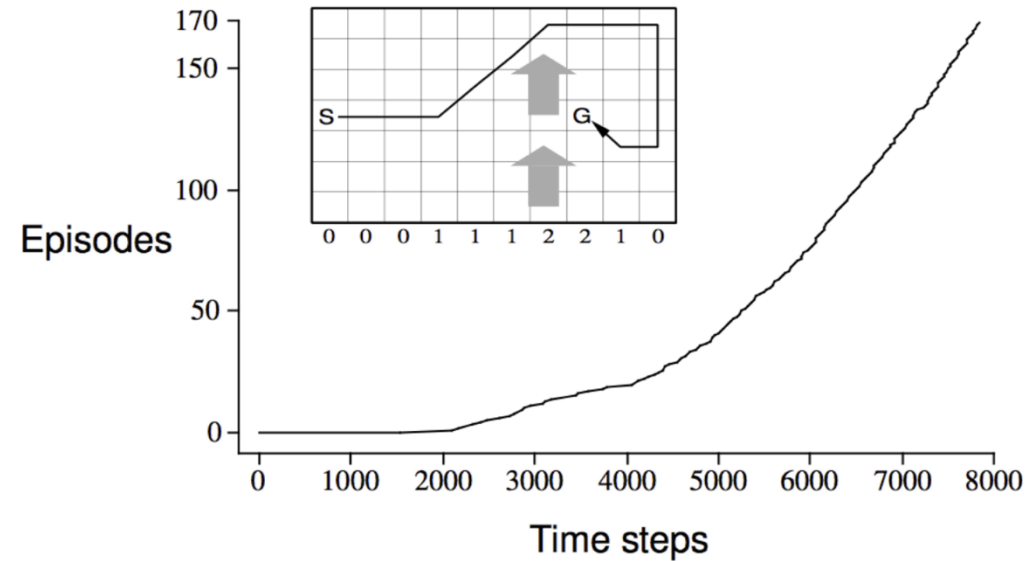
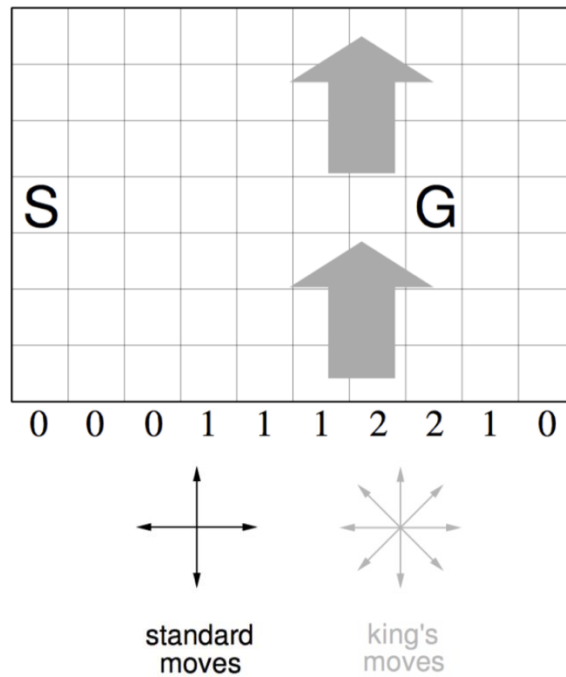


*Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.*

# Q-Learning and SARSA Algorithms

## SARSA algorithm (on-policy control)

- Example: Windy Gridworld
  - Reward: -1 per time step; no discount



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

# Q-Learning and SARSA Algorithms

## SARSA algorithm (on-policy control)

---

### Pros & Cons

- + Processes each sample immediately
- + Minimal update cost per sample
- Poses constraints on sample collection (on-policy)
- Requires a huge number of samples
- Requires careful schedule for the learning rate
- Makes minimal use of each sample
- The ordering of samples influences the outcome
- Exhibits instabilities under approximate representations
- Requires careful handling on the policy greediness

# Q-Learning and SARSA Algorithms

## Q-Learning algorithm (off-policy control)

- Evaluate one policy while following another
- Can re-use experience gathered from old policies

### Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

  Initialize  $S$

  Loop for each step of episode:

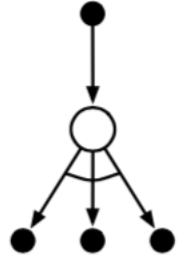
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

    Take action  $A$ , observe  $R, S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

  until  $S$  is terminal



*Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.*

# Q-Learning and SARSA Algorithms

## Q-Learning algorithm (off-policy control)

---

### Pros and Cons

- + **Processes each sample immediately**
- + **Minimal update cost per sample**
- + **Poses no constraints on sample collection (off-policy)**
- Requires a huge number of samples
- Requires careful schedule for the learning rate
- Makes minimal use of each sample
- The ordering of samples influences the outcome
- Exhibits instabilities under approximate representations



# Model-free Control: Remarks

## Monte Carlo Control

- We only studied TD-based control in this lecture.
- Note: there is also an MC-based way to do control:  
(see Sutton and Barto's RL book pp. 97 – 103)

### Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$  (arbitrarily), for all  $s \in \mathcal{S}$   
 $Q(s, a) \in \mathbb{R}$  (arbitrarily), for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$   
 $Returns(s, a) \leftarrow$  empty list, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose  $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0)$  randomly such that all pairs have probability  $> 0$

Generate an episode from  $S_0, A_0$ , following  $\pi$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :

Append  $G$  to  $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$

*Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.*

## Off-policy vs. On-policy

---

*Exercise 6.11* Why is Q-learning considered an *off-policy* control method?



# Q-Learning vs. SARSA with greedy policy

## Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$   
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$   
Loop for each episode:  
  Initialize  $S$   
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)  
  Loop 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) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$   
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$   
Loop for each episode:  
  Initialize  $S$   
  Loop for each step of episode:  
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)  
    Take action  $A$ , observe  $R, S'$   
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$   
     $S \leftarrow S'$   
  until  $S$  is terminal

*Exercise 6.12* Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates? □

# Q-Learning vs. SARSA

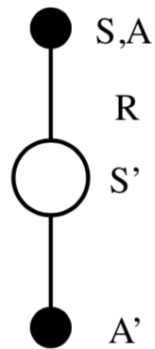
---

- Q-Learning estimates the return (total discounted future reward) for state-action pairs assuming a greedy policy (although it may follow an explorative policy)
- Instead, SARSA estimates the return for state-action pairs assuming the current policy (that it also follows)
- If the current policy is also a greedy policy, then the distinction disappears.
- SARSA will also get to the Q-Learning result if we decay  $\varepsilon$  (carefully!)

# Q-Learning vs. SARSA

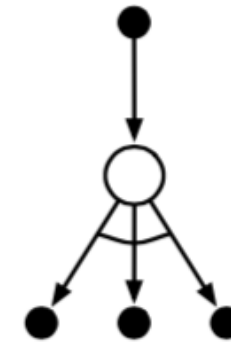
## SARSA algorithm (on-policy control)

- + Processes each sample immediately
- + Minimal update cost per sample
- Requires a huge number of samples
- Requires careful schedule for the learning rate
- Makes minimal use of each sample
- The ordering of samples influences the outcome
- Exhibits instabilities under approximate representations
- Poses constraints on sample collection (on-policy)
- Requires careful handling on the policy greediness



## Q-Learning algorithm (off-policy control)

- + Processes each sample immediately
- + Minimal update cost per sample
- + Poses no constraints on sample collection (off-policy)
- Requires a huge number of samples
- Requires careful schedule for the learning rate
- Makes minimal use of each sample
- The ordering of samples influences the outcome
- Exhibits (even more) instabilities under approximate representations



# Monte Carlo and TD Methods

---

Hands-On:

[https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld\\_td.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html)

# Double Q-Learning

## Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q_1(s, a)$  and  $Q_2(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , such that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

  Initialize  $S$

  Loop for each step of episode:

    Choose  $A$  from  $S$  using the policy  $\varepsilon$ -greedy in  $Q_1 + Q_2$

    Take action  $A$ , observe  $R, S'$

    With 0.5 probability:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left( R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A) \right)$$

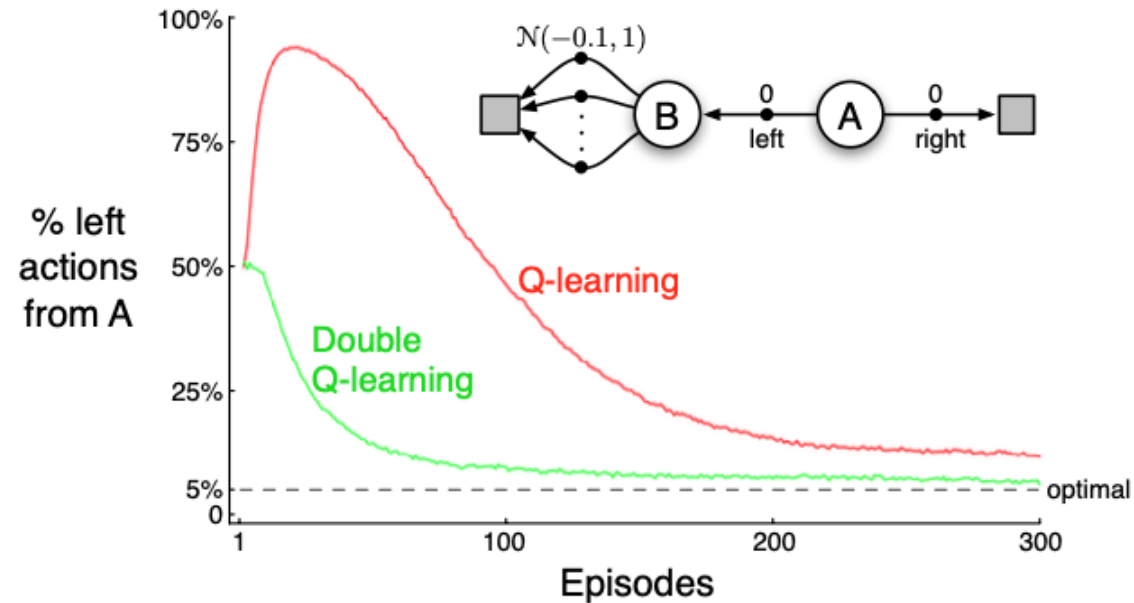
  else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left( R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A) \right)$$

$S \leftarrow S'$

  until  $S$  is terminal

# Double Q-Learning



**Figure 6.5:** Comparison of Q-learning and Double Q-learning on a simple episodic MDP (shown inset). Q-learning initially learns to take the left action much more often than the right action, and always takes it significantly more often than the 5% minimum probability enforced by  $\epsilon$ -greedy action selection with  $\epsilon = 0.1$ . In contrast, Double Q-learning is essentially unaffected by maximization bias. These data are averaged over 10,000 runs. The initial action-value estimates were zero. Any ties in  $\epsilon$ -greedy action selection were broken randomly.

# Teaser: Alphastar

## Challenge:

- Game theory: many „good“ strategies
- Imperfect information: crucial information is hidden
- Long-term planning: early actions pay off much later
- Real time: continual to game clock
- Large action space: hierarchical action space

## Solution:

- Many nice tricks 😊
- LSTMs, autoregressive policy heads with pointer networks, multi-agent centralized value baselines, ...
- It is really about population modelling!

## More Info:

<https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>

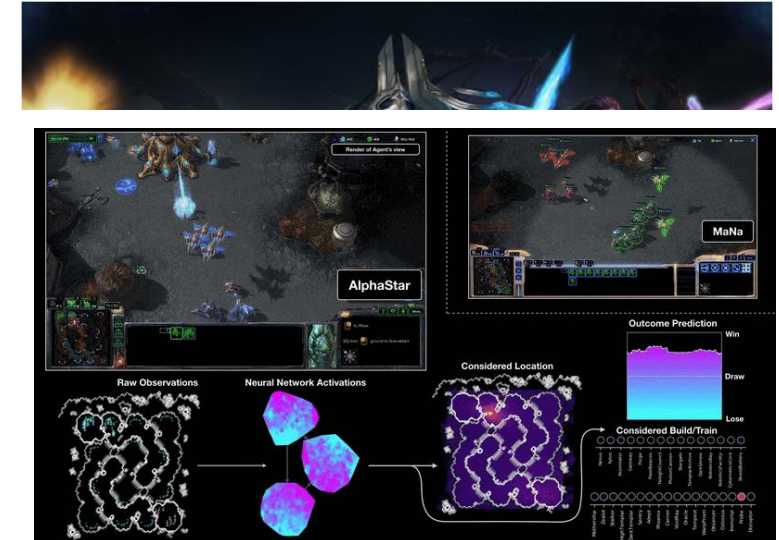
25.01.2019 16:29 Uhr

## Starcraft 2: DeepMind-KI schlägt Profi-Spieler

Die DeepMind-KI Alphastar hat professionelle Starcraft-2-Spieler besiegt. Als die KI ein einziges Match verlor, verhielt sie sich auch noch unsportlich.

Von Daniel Herbig

331



# References

---

## Books:

- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
- Bellman, R.E. 1957. Dynamic Programming. Princeton University Press.

## Lectures:

- UC Berkeley CS188 Intro to AI. [http://ai.berkeley.edu/lecture\\_slides.html](http://ai.berkeley.edu/lecture_slides.html)
- UCL Course on RL. <http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>
- Advanced Deep Learning and Reinforcement Learning (UCL + DeepMind). [http://www.cs.ucl.ac.uk/current\\_students/syllabus/compqi/compqi22\\_advanced\\_deep\\_learning\\_and\\_reinforcement\\_learning](http://www.cs.ucl.ac.uk/current_students/syllabus/compqi/compqi22_advanced_deep_learning_and_reinforcement_learning)
- Pieter Abbeel: CS 188 Introduction to Artificial Intelligence. Fall 2018

## Blogs etc.:

- [https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld\\_td.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html)