COMS W4701: Artificial Intelligence

Lecture 7: Reinforcement Learning

Tony Dear, Ph.D.
Department of Computer Science
School of Engineering and Applied Sciences

Today

- Reinforcement learning
- Passive RL (prediction) vs active RL (control)
- Monte Carlo methods (averaging samples)
- Temporal difference methods

Learning from Experience

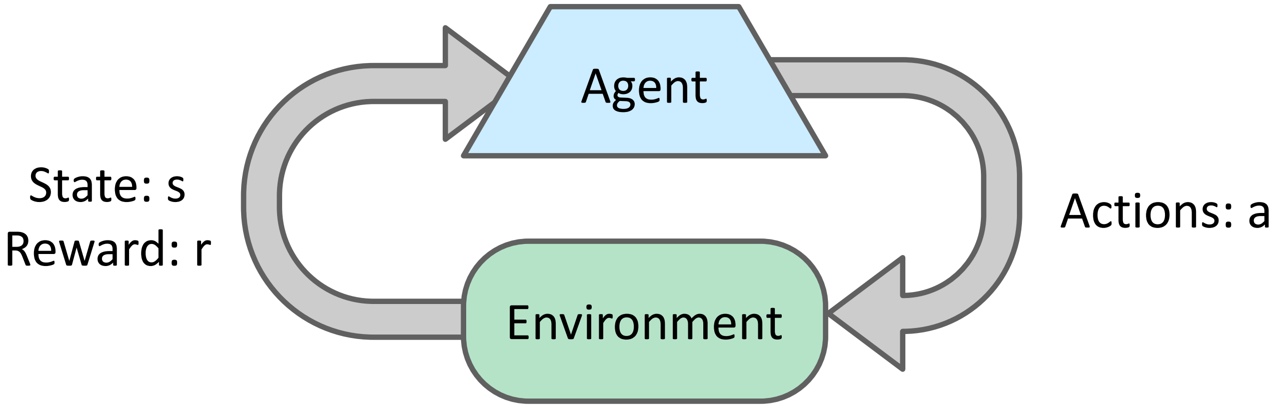
- Dynamic programming requires knowledge of environment *model*
- Agent is finding policy in advance (no actions taken)
- But models are often inaccessible or difficult to compute

- **Reinforcement learning**: Find optimal policies through *samples*
- Interact with environment, receive rewards, and formulate policies

- This generalizes the bandit problem (now with states *and* actions)

Reinforcement Learning

- We still have an underlying MDP
  - A set of states S
  - A set of actions A
  - A transition model $T(s, a, s')$
  - A reward function $R(s, a, s')$

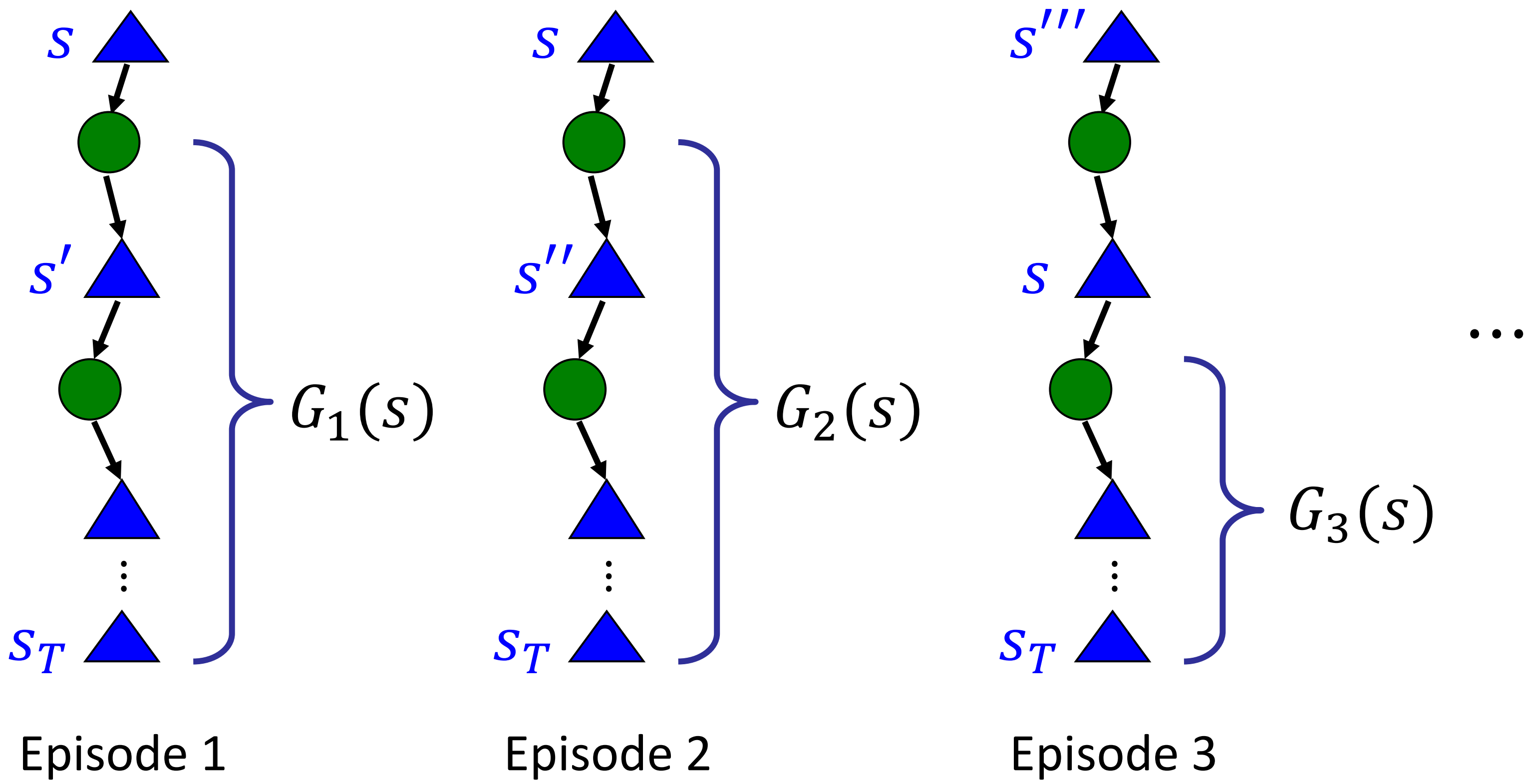


- Still looking for a policy or value function

- We no longer know (or use) T or R!
- Instead, we perform actions and receive feedback from environment

State Values from Sampling

- Idea: A state's value can be estimated from observed utilities *after* visiting that state
- **Monte Carlo**: Estimate state values by averaging utilities over multiple episodes



$$V(s) = \frac{1}{N}\sum_{i=1}^{N} G_i(s)$$

Monte Carlo Prediction

- **Prediction**: Estimate state values for a fixed policy π (policy evaluation)
- *First-visit* MC: A value is estimated after first visit to state within episode

- We generate many episodes of s, a, r sequences following π:

$$E_i = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T)$$

- Utility estimate of first appearance of s_t in episode E_i:

$$G_i(s_t) = \sum_{j=0}^{T-t-1} \gamma^j r_{j+t+1}$$

- V^π is estimated by averaging all individual utility samples:

$$V^\pi(s) = \frac{1}{N} \sum_i G_i(s)$$

Example: Mini-Gridworld

- States: A, B, C; actions: L, R; rewards received upon entering each state
- Policy: $\pi(s) = L$ for all states s
- Each episode ends after 5 actions (finite-horizon)

+3	-2	+1
A	B	C

- Episode 1: $(A, +3, A, -2, B, +1, C, -2, B, +3)$
- Episode 2: $(A, -2, B, +3, A, -2, B, +1, C, -2)$
- Episode 3: $(C, +1, C, -2, B, +3, A, -2, B, +3)$

Episode 1:
$G_1(A) = 3 + \gamma(-2) + \gamma^2(1) + \gamma^3(-2) + \gamma^4(3)$
$G_1(B) = 1 + \gamma(-2) + \gamma^2(3)$
$G_1(C) = -2 + \gamma(3)$

Episode 2:
$G_2(A) = -2 + \gamma(3) + \gamma^2(-2) + \gamma^3(1) + \gamma^4(-2)$
$G_2(B) = 3 + \gamma(-2) + \gamma^2(1) + \gamma^3(-2)$
$G_2(C) = -2$

Episode 3:
$G_3(A) = -2 + \gamma(3)$
$G_3(B) = 3 + \gamma(-2) + \gamma^2(3)$
$G_3(C) = 1 + \gamma(-2) + \gamma^2(3) + \gamma^3(-2) + \gamma^4(3)$

- $V^\pi(s) = \frac{1}{3}\left(G_1(s) + G_2(s) + G_3(s)\right)$

Finer Points

- Some states may be visited more often than others
- Values converge to true V^{π} after many, many visits

- Estimates of different state values are independent (in contrast to DP)
- Result: Computational complexity of estimating specific state values is independent of state space size!

- Can choose to focus on certain states and ignore others

Constant-α Monte Carlo

- The *online* version of MC prediction uses the following update to a state value $V^{\pi}(s_t)$:

$$V^{\pi}(s_t) \leftarrow \frac{NV^{\pi}(s_t) + G_t}{N+1} = V^{\pi}(s_t) + \frac{1}{N+1}\left(G_t - V^{\pi}(s_t)\right)$$

- Update is of the form "old value" + "weighted error"
- If the "error" $G_t - V^{\pi}(s_t) = 0$, no update would occur
- The weight $1/(N+1)$ shrinks as we see more samples over time

- **Constant-α MC**: We can use an arbitrary **learning rate** α

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha\left(G_t - V^{\pi}(s_t)\right)$$

Temporal-Difference Update

- There is another way that we can estimate G
- Recall from DP: $V^{\pi}(s_t)$ depends on values of successors from s_t

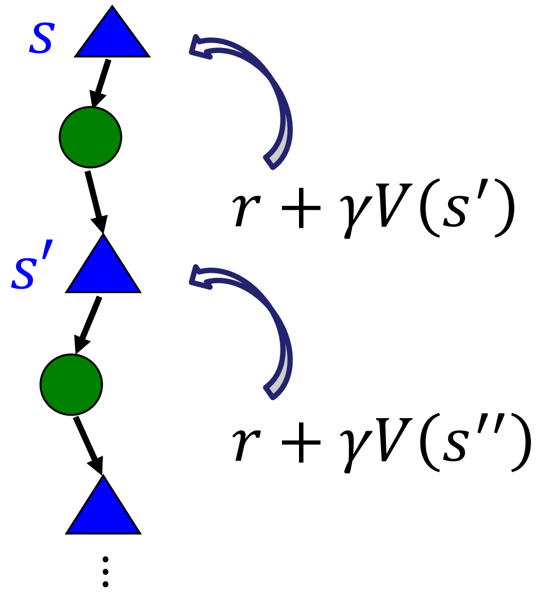
- **One-step TD update ($\boldsymbol{TD(0)}$):**

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha\big(\overbrace{r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)}^{\text{TD error } \delta_t}\big)$$

- Unlike DP but like MC, TD uses *samples* to estimate *expected values*
- Unlike MC but like DP, TD *bootstraps* by using *current estimates* $V^{\pi}(s')$ to update $V^{\pi}(s)$

$TD(0)$ for Prediction

- **Given:** Policy π, step size α between 0 and 1
- **Initialize** $V^{\pi}(s) \leftarrow 0$
- **Loop**:
  - **Initialize** starting state s if needed
  - **Generate** sequence $(s, \pi(s), r, s')$
  - $V^{\pi}(s) \leftarrow V^{\boldsymbol{\pi}}(s) + \alpha\big(r + \gamma V^{\boldsymbol{\pi}}(s') - V^{\pi}(s)\big)$
  - $s \leftarrow s'$



Example: Mini-Gridworld

- All values initialized to 0; $\gamma = 0.8$, $\alpha = 0.5$
- Policy to evaluate: $\pi(s) = L$ for all states

+3	-2	+1
A	B	C

- Observed state and reward sequence: $(A, +3, A, -2, B, +1, C, -2, B, +3, A)$

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha\left(r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)\right)$$

Transition	$(A, +3)$	$(A, -2)$	$(B, +1)$	$(C, -2)$	$(B, +3)$
$V^{\pi}(A)$	1.5	-0.25	-0.25	-0.25	-0.25
$V^{\pi}(B)$	0	0	0.5	0.5	1.65
$V^{\pi}(C)$	0	0	0	-0.8	-0.8

Optimality

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha\delta_t$$

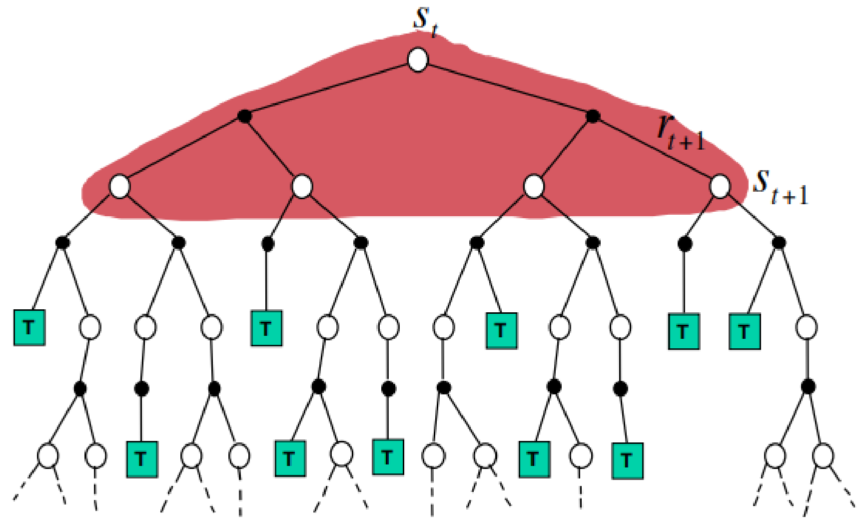
- TD methods perform updates immediately with no episodic structure (MC)
- Useful if problems have long episodes or are continuing tasks

- For sufficiently small α, average values of V^{π} converge to true values
- If α is constant, V^{π} prone to jumping around even near convergence

- In practice, we try to decrease α to 0 over time

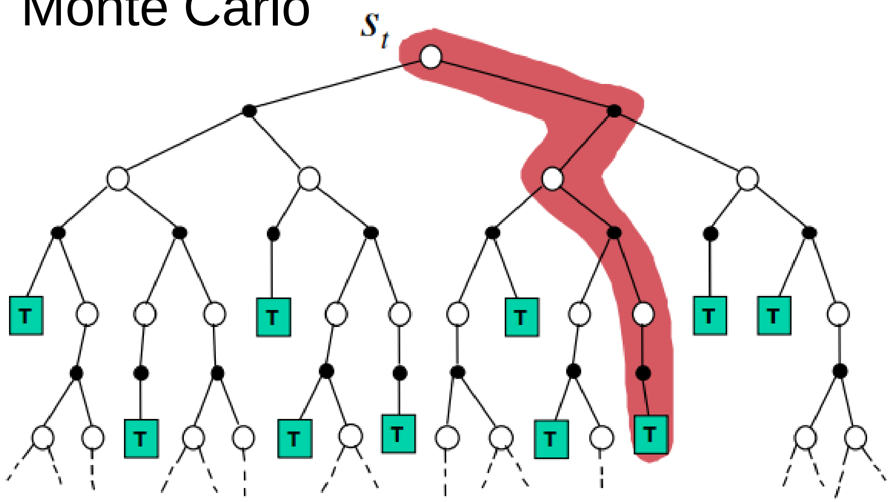
MDP Method Comparison

https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf

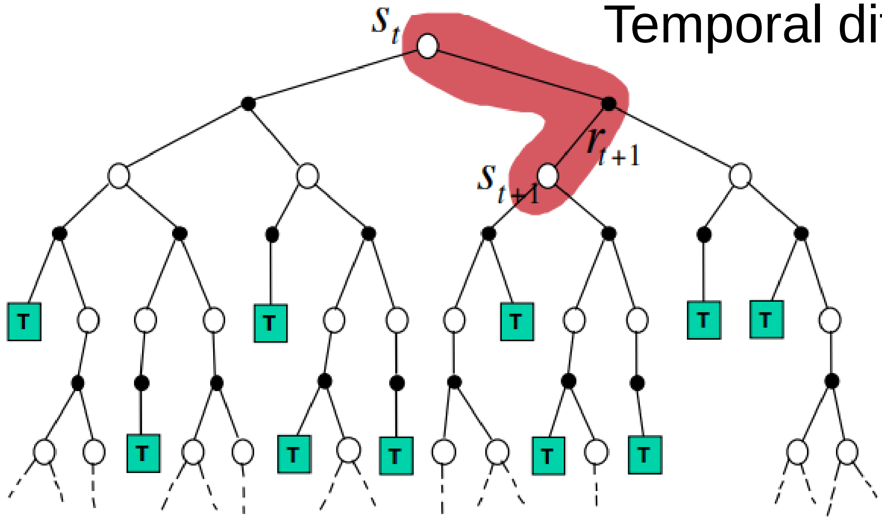


Dynamic programming

Monte Carlo



Temporal difference



ε–Greedy Policies

- **Control** problem: Learn a better or optimal policy instead of evaluating a fixed one
- How to choose which action to take?

- Recall bandits: exploration vs exploitation
- Exploit to maximize expected utility, explore to learn new information

- **ε-greedy policy**: Policy becomes *stochastic*; choose best action most of the time, but occasionally execute random action instead

$$\Pr(a|s) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A(s)|} & \text{for } a = \operatorname*{argmax}_{a'} Q(s, a') \\ \dfrac{\varepsilon}{|A(s)|} & \text{for all other actions } a \end{cases}$$

Q-Values

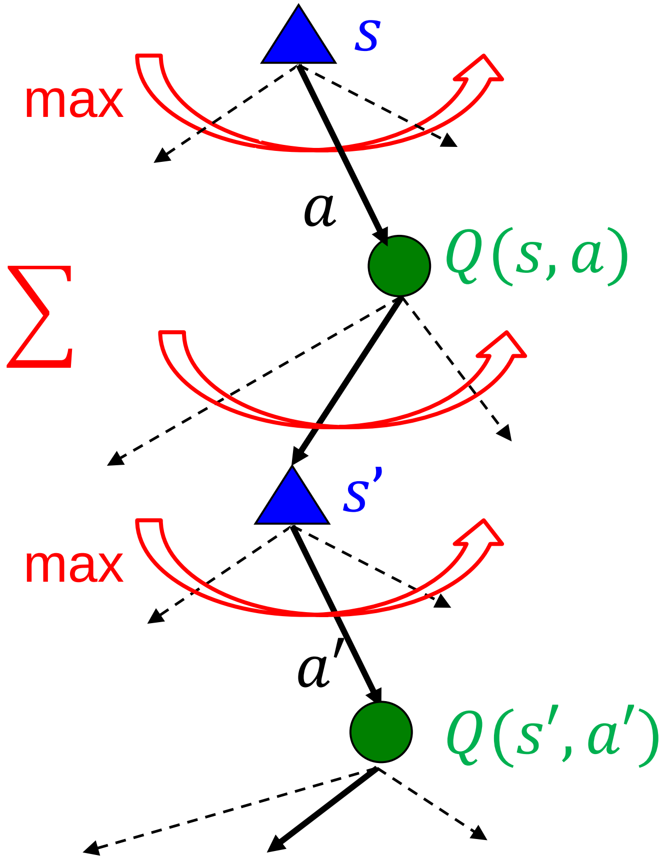
- Another issue: State values alone are insufficient for extracting a new policy without a model!

$$\pi(s) \leftarrow \operatorname*{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V(s')]$$

- Solution: Learn **Q (state-action) values** instead
  - Similar to action values in bandit problems

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

$$V^*(s) = \max_a Q(s,a) \qquad \pi^*(s) = \operatorname*{argmax}_a Q(s,a)$$



TD Learning for Control

- We can convert our TD learning rule for state values to one for Q-values
- Once we sufficiently learn the Q-values, we can extract a policy π
- Recall TD learning: Immediate, bootstrapped updates; no episodic structure

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\big(r + \gamma V^{\pi}(s') - V^{\pi}(s)\big)$$

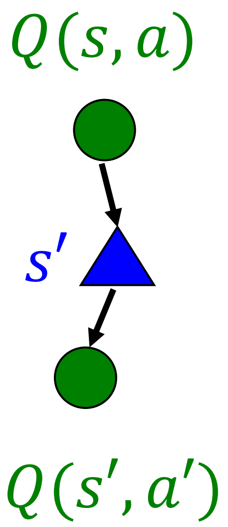
$$\Downarrow$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$$

- New issue: What is $Q(s', a')$? Specifically, what is a'? given by pi
- Approach 1: Use action a' that is actually taken from s' (can be exploratory action)
- Approach 2: Use action a' corresponding to *exploitative* action only (even if not taken)

SARSA

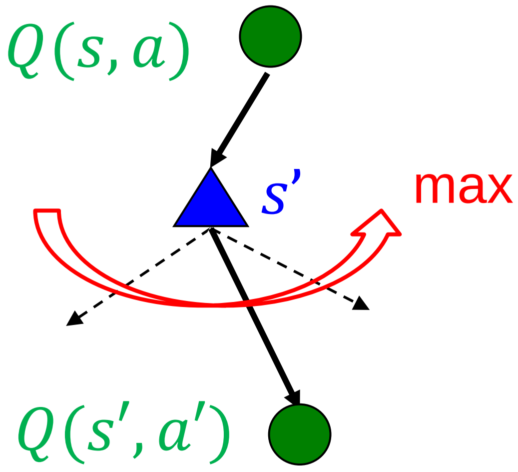
- **Given:** Step size α, exploration rate ε
- **Initialize** $Q(s,a) \leftarrow 0$, behavior policy π (e.g., ε-greedy)
- **Loop**:
  - **Initialize** starting state s, action $a = \pi(s)$ if needed
  - **Generate** sequence (s, a, r, s'), $a' \leftarrow \pi(s')$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$
  - $s \leftarrow s', a \leftarrow a'$



Q-Learning

- **Given:** Step size α, exploration rate ε
- **Initialize** $Q(s,a) \leftarrow 0$, behavior policy π (e.g., ε-greedy)

- **Loop**:
  - **Initialize** starting state s if needed, action $a = \pi(s)$
  - **Generate** sequence (s, a, r, s')
  - $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
  - $s \leftarrow s'$



Example: Mini-Gridworld

- Suppose currently $Q(A, L) = 1.5$, $Q(A, R) = 0$
- Behavior policy is ε-greedy

+3	-2	+1
A	B	C

- Observed (s, a, r, s') sequence: $A, L, +3, A$
- Suppose behavior policy generates $a' = R$ (*explore*)

$\gamma = 0.8$
$\alpha = 0.5$

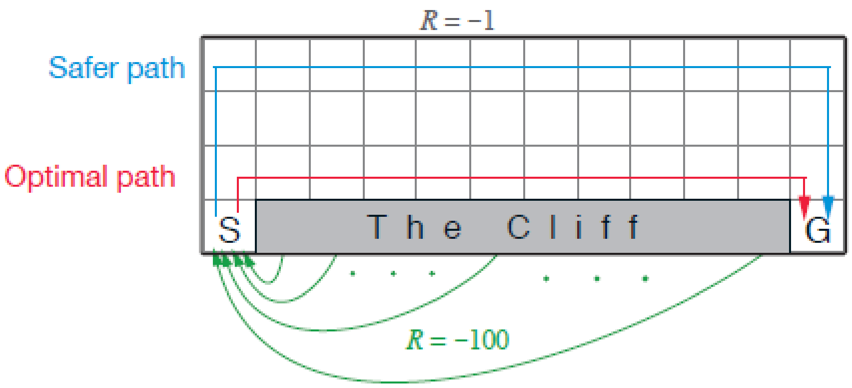
- SARSA: $Q(A, L) \leftarrow Q(A, L) + \alpha\big(r + \gamma \boldsymbol{Q}(\boldsymbol{A}, \boldsymbol{R}) - Q(A, L)\big) = 2.25$

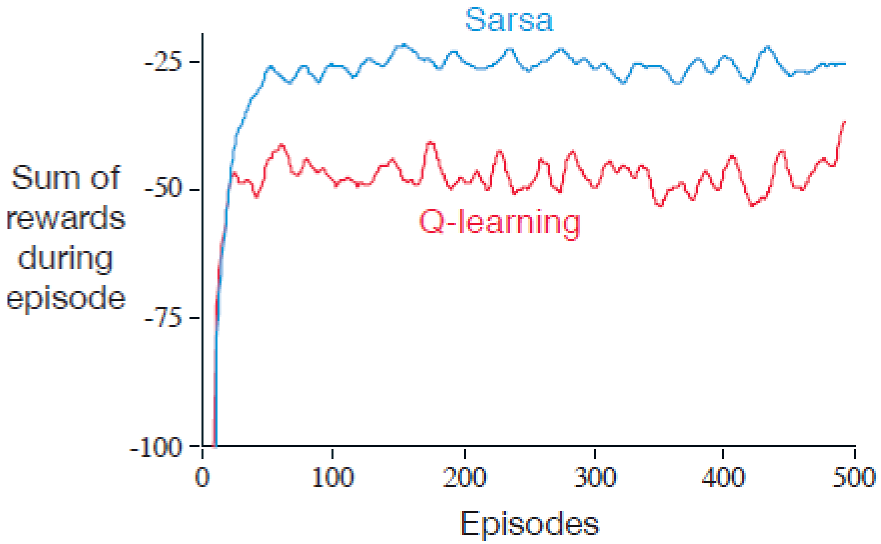
- Q-learning: $Q(A, L) \leftarrow Q(A, L) + \alpha\left(r + \gamma \max_{\boldsymbol{a}} \boldsymbol{Q}(\boldsymbol{A}, \boldsymbol{a}) - Q(A, L)\right) = 2.85$

Cliff Walking

- Start and goal terminal states, in addition to "cliff" terminal states
- Living reward of -1 in most states; "cliff" states reward -100

- SARSA learns "safer" path away from cliff, higher rewards on average
- Q-learning learns optimal path along cliff, despite lower rewards due to exploration





Solving Sequential Decision Problems

	Evaluate a fixed policy π: Solve for V^{π}	**Learn an optimal policy π^* or optimal value function V^***
Dynamic Programming (known model T, R)	• Solve a linear system • Iterative policy evaluation (step 1 of policy iteration)	• Value iteration • Policy iteration
Reinforcement Learning (no model)	• First-visit Monte Carlo • Constant-α Monte Carlo • TD(0)	• SARSA • Q-learning followed by max / argmax operations

Function Approximation*

- In real problems, often have too many state-action combinations
- States may share common *features*—no need to visit all of them!

- Familiar idea: Evaluation functions of states using features

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

- Now instead of storing $|S||A|$ tabular values, we only have n weight parameters

- As with games, evaluation function must reflect true utility
- Sharing common features among states can be misleading

Function Approximation*

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

- We now learn the function weights w_i instead of Q-values
- How to update from observed samples?
- Before:

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

$$Q(s,a) \leftarrow Q(s,a) + \alpha(sample - Q(s,a))$$

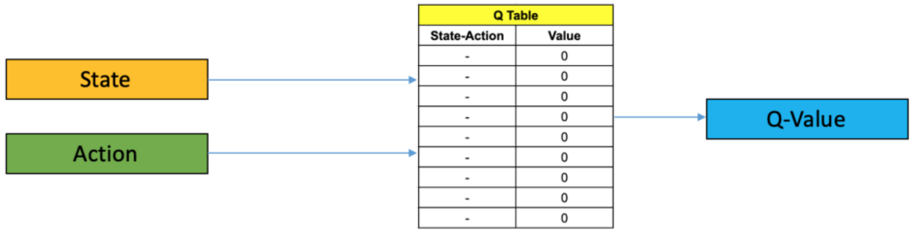
$$Q(s,a) \leftarrow Q(s,a) + \alpha(difference)$$

- Similar idea for function weights: $w_i \leftarrow w_i + \alpha(difference) f_i(s,a)$

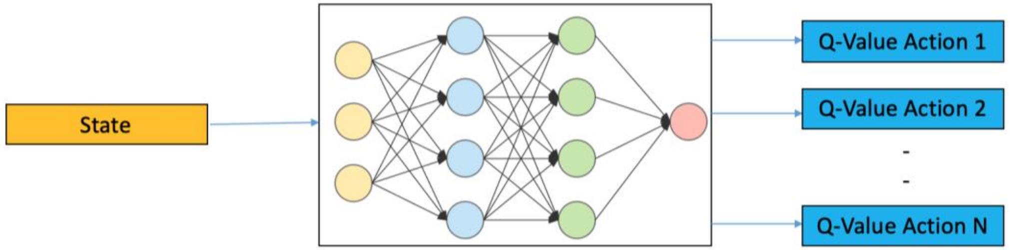
- Idea: Weights of *more active features* receive larger updates
- Any Q-value can potentially change whenever a feature weight is updated!

Deep Reinforcement Learning

- We’ve gone from learning a table of values to a bunch of feature weights
- Eval functions don’t have to be linear—they can be any black box that relates state-action pairs to (Q-)values
- **Deep reinforcement learning** uses neural networks as function approximators



Q Learning



Deep Q Learning

Policy Search*

- Instead of learning values and then extracting policy, we can also learn policy directly

- **Policy search**: Directly learn a policy represented by Q-functions $\hat{Q}_\theta(s, a)$
- Each combination of *parameters* θ produces a different policy

- Not the same as Q-learning!! We don't care about Q^*, just a good policy
- Same in game trees with evaluation functions—we don't care about true utilities if we have good actions/moves

- Policy search methods iteratively improve θ parameters using *policy gradients*

Summary

- Reinforcement learning: agents take actions, receive percepts, and tweak actions over time to maximize rewards

- Prediction: Evaluate a given policy
- Control: Learn an optimal policy

- Monte Carlo methods estimate by averaging samples of episodic returns
- Temporal difference methods bootstrap by using estimates to inform other estimates

- RL has many generalizations, subject of much current research