COMS W4701: Artificial Intelligence

Lecture 5b: RL Control

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

Today

- State-action (Q) values

- Behavior and target policies
- Exploration, ε-greedy policies

- SARSA
- Q-learning

Solving Sequential Decision Problems

	Evaluate a fixed policy π: Solve for V^{π}	**Learn optimal value function V^* or optimal policy π^***
Dynamic Programming (known model T, R)	• Formulate and solve linear system of equations • Iterative policy evaluation	• Value iteration for V^*, followed by policy extraction for π^*
Reinforcement Learning (no model)	• First-visit Monte Carlo • Constant-α Monte Carlo • TD(0)	???

State-Action Values

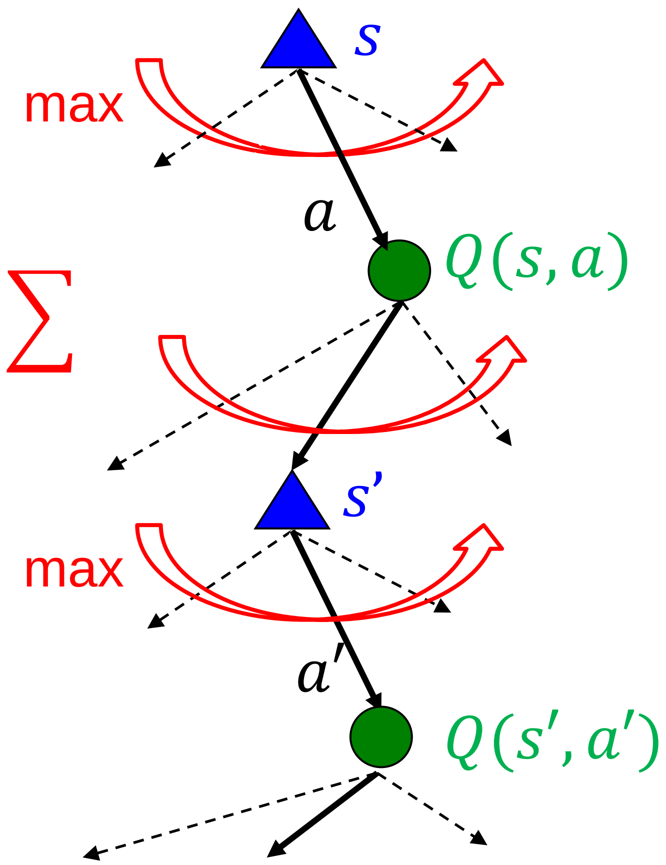
- How do we compute a policy π given its values V^{π}?

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

don't have these!

- Idea: Directly *learn* **state-action values** $Q^{\pi}(s,a)$ of the chance nodes using RL

- $Q^{\pi}(s,a)$ is the *expected* utility of s after taking action a and then following π thereafter

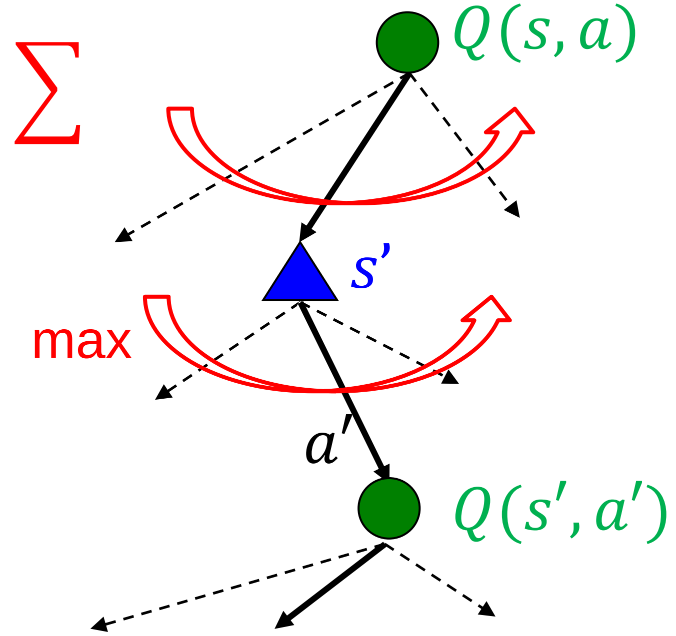


Values and Policy

- **Q values** can be defined as functions of V^* or recursively as functions of themselves

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

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



- To learn policies, we will first learn Q values
- Then use these to compute V^* or π^*

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

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

Example: Mini-Gridworld

- Suppose we have found the following Q values:

A	B	C

$Q(s,a)$	$s = A$	$s = B$	$s = C$
$a = Left$	$\mathbf{4.061}$	$\mathbf{4.364}$	0.485
$a = Right$	1.152	2.364	$\mathbf{1.394}$

- Value function: $V^* = \max\limits_{\boldsymbol{a}} \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) = (4.061, 4.364, 1.394)$

- Policy: $\pi^* = \operatorname*{\mathbf{argmax}}\limits_{\boldsymbol{a}} \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) = (Left, Left, Right)$

Behavior Policies

- Suppose our agent is given some initial (possibly random) policy π
- *Control* problem: We want to *improve* and make changes to π over time

- In RL, the agent always behaves according to the **behavior policy**
- E.g., $\pi(s) = \text{argmax}_a\, Q(s,a)$ is a *greedy* behavior policy
- But this is only our **target policy** (optimal) when the Q-values are correct!

- How to behave if we are also learning Q-values at the same time?
- Instead of always acting greedily, add in some *exploration*

ε-Greedy Policies

- **ε-greedy behavior policy**: Choose best action most of the time, but with small probability ε, execute random action instead

- *Exploit* and choose action $a = \underset{a'}{\operatorname{argmax}}\, Q(s, a')$ with probability $1 - \varepsilon$
- *Explore* and choose action uniformly at random with probability ε

- Exploration may lead to smaller *short-term* rewards, but crucial for discovering better actions and higher *long-term* rewards
- Exploration rate ε is yet another tunable agent parameter

TD Learning for Control

- Q is just a more detailed version of V, so we can apply a TD approach to learn these values as well:

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

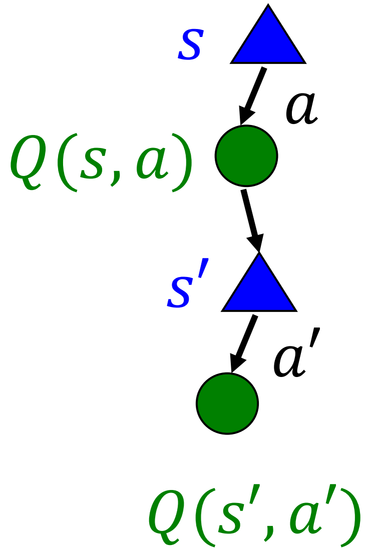
- As in TD(0), we see the transition (s,a,r,s') and then update $Q(s,a)$
- What do we use for a', the action in the successor state s'?

- In TD(0), the target uses $V^{\pi}(s')$, or the value of s' following π
- So maybe we should select a' according to our *behavior policy* (ε-greedy)
- Another possibility: Select a' according to our *target policy* (greedy)

On-Policy Learning: SARSA

- **On-policy** learning: Q-value update is based on the successor Q-value corresponding to the *action that is actually taken according to the behavior policy* π

- **Given:** Learning rate α, exploration rate ε, discount factor γ
- **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'), $a' \leftarrow \pi(s')$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$
  - $s \leftarrow s'$, $a \leftarrow a'$



Example: Mini-Gridworld

- Suppose we currently have $Q(A, L) = 1$, $Q(B, L) = 2$, $Q(B, R) = 1.5$
- Behavior policy is ε-greedy; $\alpha = 0.5$, $\gamma = 0.8$

A	B	C

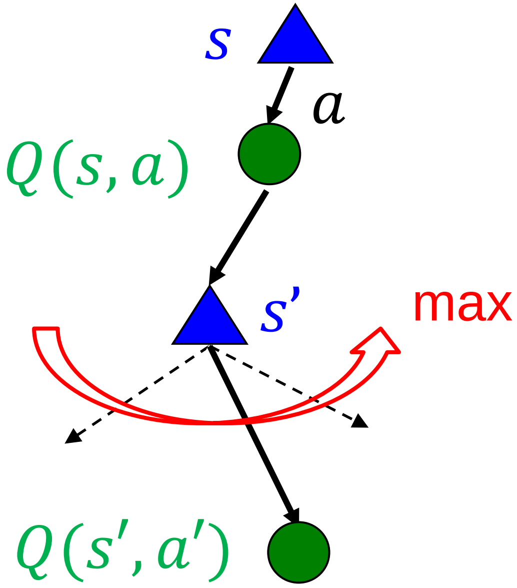
- Observed (s, a, r, s') sequence: $A, L, +3, B$
- Suppose behavior policy generates $a' = R$ (*explore*)
- Target: $r + \gamma \boldsymbol{Q}(\boldsymbol{B}, \boldsymbol{R}) = 3 + 0.8(1.5) = 4.2$
- Q-value update: $Q(A, L) \leftarrow 1 + 0.5(4.2 - 1) = 2.6$

Off-Policy Learning: Q-Learning

- **Off-policy** learning: Q-value update is based on the successor Q-value corresponding to the *action that "should be" taken according to the target policy*

- **Given:** Learning rate α, exploration rate ε, discount factor γ
- **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')
  - $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s', a') - Q(s,a)\right)$
  - $s \leftarrow s'$

Action a' selected according to greedy policy ($\varepsilon = 0$)



Example: Mini-Gridworld

- Suppose we currently have $Q(A, L) = 1$, $Q(B, L) = 2$, $Q(B, R) = 1.5$
- Behavior policy is ε-greedy; $\alpha = 0.5$, $\gamma = 0.8$

A	B	C

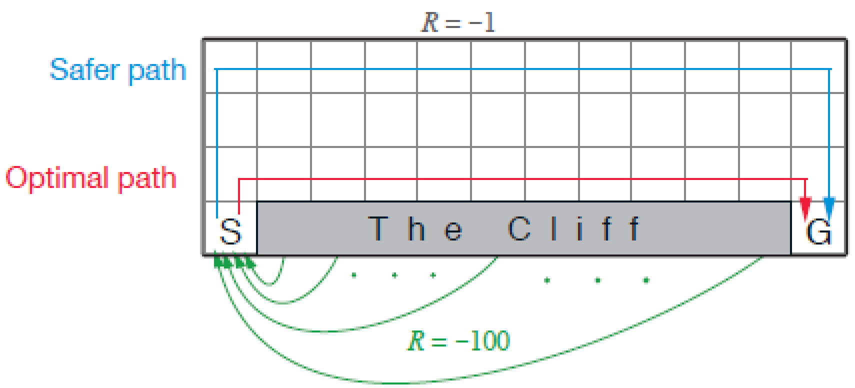
- Observed (s, a, r, s') sequence: $A, L, +3, B$
- Target: $r + \gamma \boldsymbol{\max_a Q(B, a)} = r + \gamma \boldsymbol{Q(B, L)} = 3 + 0.8(2) = 4.6$
- Q-value update: $Q(A, L) \leftarrow 1 + 0.5(4.6 - 1) = 2.8$

Example: Cliff Walking

- Deterministic grid world with one terminal (goal) state
- Living reward is -1 in all states except for "cliff", which rewards -100

- Since transitions are deterministic, the optimal action at each state is to head in goal direction while ignoring the cliff

- We can use TD control to learn Q values and the corresponding policy

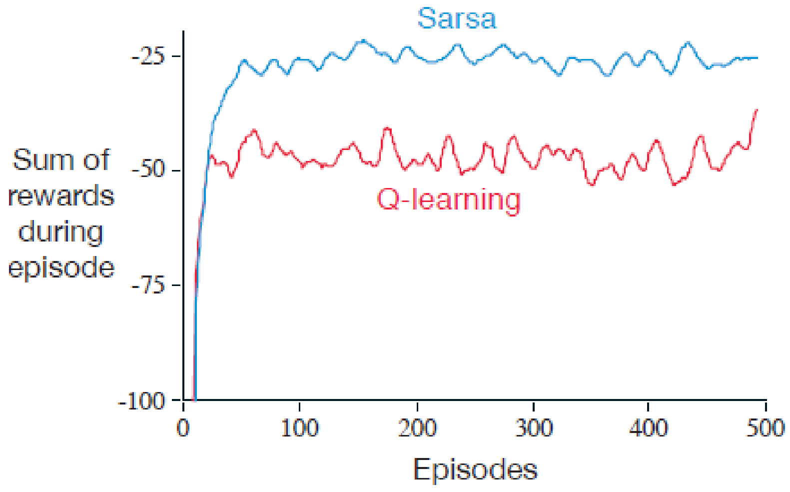


Example: Cliff Walking

- With Q-learning, all learned Q-values will reflect the assumption that the agent will always act greedily (take action to move toward goal)
- Agent prefers "optimal path" as $\max_a Q(s,a)$ is highest next to the cliff

- With SARSA, learned Q-values will be overall lower, esp near cliff
- Reflect all instances when agent chose to "explore" and jump off the cliff for -100 reward

- Learned policy will be to take "safer" path!



SARSA vs Q-Learning

- SARSA learns the optimal behavior policy, e.g. ε-greedy
- Each Q-value update just requires knowledge of the next action
- Learned values are generally lower, reflecting suboptimal actions taken

- Q-learning learns the optimal target policy, e.g. greedy
- Each Q-value update requires a *max* over successor state actions
- Learned values are *not* affected by suboptimal actions from exploration

- SARSA & Q-learning are the same if behavior & target policies are as well

Solving Sequential Decision Problems

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

Summary

- The RL control problem involves learning optimal policies and values from data

- We need to explore to try new states and actions; keep track of *state-action (Q) values*
- Simple approach is a ε-greedy behavior policy

- We can learn and update Q values using temporal difference learning

- SARSA (on-policy): Update Q value using Q value of action taken in successor state
- Q-learning (off-policy): Update current Q value using *best* Q value in successor state