# **Reinforcement Learning**

Yuntian Deng

Lecture 15

Readings: RN 22.1 - 21.3. PM 12.1, 12.5, 12.8.

CS 486/686: Intro to Al Lecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen 1 / 42

#### Outline

Learning Goals

Introduction to Reinforcement Learning

Passive Adaptive Dynamic Programming

Active Adaptive Dynamic Programming

**Temporal Difference Error** 

Q-Learning

SARSA

CS 486/686: Intro to AI Lecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen 2 / 42

## Learning Goals

- Trace and implement the passive adaptive dynamic programming algorithm.
- Explain the trade-off between exploration and exploitation.
- Trace and implement the active adaptive dynamic programming algorithm.
- Trace and implement the active Q-learning.
- Trace and implement the active SARSA.
- Understand the on-policy and off-policy RL.



#### Introduction to Reinforcement Learning

Passive Adaptive Dynamic Programming

Active Adaptive Dynamic Programming

Temporal Difference Error

**Q-Learning** 

#### SARSA

CS 486/686: Intro to AI Lecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen 4 / 42

# A Reinforcement Learning Agent

Let's consider fully observable, single-agent reinforcement learning. We will formalize this problem as a Markov decision process.

- Given the set of possible states S and the set of actions A.
- Observes the state and the rewards received.
- Carries out an action.
- ► Goal is to maximize its discounted reward (i.e. return).

Why is reinforcement learning challenging?

- Which action was responsible for this reward/punishment?
- How will this action impact my utility?
- Should I explore or exploit?

CS 486/686: Intro to AI Lecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen 5 / 42

Learning Goals

Introduction to Reinforcement Learning

Passive Adaptive Dynamic Programming

Active Adaptive Dynamic Programming

Temporal Difference Error

Q-Learning

SARSA

CS 486/686: Intro to AI Lecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen 6 / 42

## Passive Learning Agent

Fix a policy  $\pi$ .

- Goal is to learn  $V^{\pi}(s)$  (expected value of  $\pi$  for state s).
- Similar to policy evaluation.
- Does not know the transition model P(s'|s, a) nor the reward function R(s).
- Solution: Adaptive Dynamic Programming
  - Learn P(s'|s, a) and R(s) using the observed transitions and rewards.
  - Learn V<sup>π</sup>(s) by solving Bellman equations (exactly or iteratively).

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s').$$

A model-based approach - uses model of environment

CS 486/686: Intro to AI Lecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen 7 / 42

#### Passive ADP Algorithm

- 1. Repeat steps 2 to 5.
- 2. Follow policy  $\pi$  and generate an experience  $\langle s, a, s', r \rangle$ .
- 3. Update reward function:  $R(s) \leftarrow r$
- 4. Update the transition probability.

$$N(s, a) = N(s, a) + 1$$
  

$$N(s, a, s') = N(s, a, s') + 1$$
  

$$P(s'|s, a) = N(s, a, s')/N(s, a)$$

5. Derive  $V^{\pi}(s)$  by using the Bellman equations.

$$V(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))V(s')$$

CS 486/686: Intro to AI Lecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen 8 / 42

#### Passive ADP Example

$s_{11}$	+1
$s_{21}$	-1

• 
$$\pi(s_{11}) = down, \pi(s_{21}) = right$$

 $\blacktriangleright \ \gamma = 0.9$ 

$$R(s_{11}) = -0.04, R(s_{21}) = -0.04, R(s_{12}) = 1, R(s_{22}) = -1$$

$$\triangleright N(s,a) = 5, \forall s, a.$$

- ▶ N(s, a, s') = 3 for the intended direction.
- N(s, a, s') = 1 for a direction to the left or right of the intended direction.
- The current state is s<sub>11</sub>.
- We take an action to go 'down', and we actually land in s<sub>21</sub>

#### Passive ADP Example continued



- 1. No need to update the reward function.
- 2. Update the counts.

 $N(s_{11}, down) = 6 \text{ and } N(s_{11}, down, s_{21}) = 4.$ 

3. Solve the Bellman equations.

$$V(s_{11}) = -0.04 + 0.9(0.667V(s_{21}) + 0.167(1) + 0.167V(s_{11}))$$
  
$$V(s_{21}) = -0.04 + 0.9(0.6(-1) + 0.2V(s_{11}) + 0.2V(s_{21}))$$

The solutions are:

$$V(s_{11}) = -0.4378, V(s_{21}) = -0.8034$$

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen10 / 42

# Q: Updating the transition probabilities

**Q #1:** Suppose the agent is in state  $s_{23}$ . Presently,  $N(s_{23}, down) = 23$  and  $N(s_{23}, down, s_{24}) = 2$ .

The agent tries to move down, but accidentally moves right. After this experience, what is  $P(s_{24}|s_{23}, down)$ ?

	1	2	3	4
1				
2		Х		-1
3				+1

(A) 0.087

(B) 0.1

(C) 0.125

(D) 0.13

(E) 0.667

CS 486/686: Intro to AlLecturer: Yuntian Deng Slide

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen11 / 42

# Q: Updating the transition probabilities

**Q #1:** Suppose the agent is in state  $s_{23}$ . Presently,  $N(s_{23}, down) = 23$  and  $N(s_{23}, down, s_{24}) = 2$ .

The agent tries to move down, but accidentally moves right. After this experience, what is  $P(s_{24}|s_{23}, down)$ ?

	1	2	3	4
1				
2		Х		-1
3				+1

- (A) 0.087
- (B) 0.1
- (C) 0.125
- (D) 0.13
- (E) 0.667

 $\rightarrow$  Correct answer is (C).  $P(s_{24}|s_{23}, down) = 3/24 = 0.125$ . CS 486/686: Intro to AlLecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen11 / 42 Learning Goals

Introduction to Reinforcement Learning

Passive Adaptive Dynamic Programming

Active Adaptive Dynamic Programming

Temporal Difference Error

**Q-Learning** 

SARSA

CS 486/686: Intro to AlLecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen12 / 42

#### Active ADP

The passive ADP agent learns the expected value of a fixed policy.

What action should the agent take at each step? Two things are useful for the agent to do:

- 1. exploit: take an action that maximizes V(s).
- 2. explore: take an action that is different from the optimal one.

 $\rightarrow$  Actions serve two purposes: (1) provide rewards, (2) gather more data to learn better model. (long-term benefits)

The greedy agent seldom converges to the optimal policy and sometimes converges to horrible policies because the learned model is not the same as the true environment.

There is a trade-off between exploitation and exploration.

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen13 / 42

## Tradeoff Between Exploitation and Exploration

1.  $\epsilon$ -greedy exploration strategy:

- Select random action with probability  $\epsilon$ , and
- Select the best action with probability  $1 \epsilon$ .
- We can decrease  $\epsilon$  over time.
- 2. Softmax selection using Gibbs/Boltzmann distribution.

• Choose action 
$$a$$
 with probability  $\frac{e^{Q(s,a)/T}}{\sum_a e^{Q(s,a)/T}}$ .

- ► T > 0 is the temperature. When T is high, the distribution is close to uniform. When T is low, the higher-valued actions have higher probabilities.
- 3. Initialize the values optimistically to encourage exploration.

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen14 / 42

### Optimistic Utility Values to Encourage Exploration

We will learn  $V^+(s)$  (the optimistic estimates of V(s)).

$$\begin{split} V^+(s) &\leftarrow R(s) + \gamma \max_a f\bigg(\sum_{s'} P(s'|s,a) V^+(s'), N(s,a)\bigg) \\ f(u,n) &= \begin{cases} R^+, \text{ if } n < N_e \\ u, \text{otherwise} \end{cases} \end{split}$$

f(u,n) trade-offs exploitation and exploration.

- R<sup>+</sup> is the optimistic estimate of the best possible reward obtainable in any state.
- If we haven't visited (s, a) at least N<sub>e</sub> times, assume its expected value is R<sup>+</sup>. → Make state attractive for exploration initially.
- Otherwise, use the current utility value  $(V^+(s))$ .

CS 486/686: Intro to AlLecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen15 / 42

## Active ADP Algorithm

- 1. Initialize  $R(s), V^+(s), N(s, a), N(s, a, s')$ .
- 2. Repeat steps 3 to 7 until we have visited each (s, a)at least  $N_e$  times and the  $V^+(s)$  values converged.
- 3. Determine the best action a for current state s using  $V^+(s)$ .

$$a = \arg\max_{a} f\left(\sum_{s'} P(s'|s, a) V^{+}(s'), N(s, a)\right), \ f(u, n) = \begin{cases} R^{+}, \ \text{if } n < N_{e} \\ u, \text{otherwise} \end{cases}$$

- 4. Take action a and generate an experience  $\langle s, a, s', r \rangle$
- 5. Update reward function:  $R(s) \leftarrow r$
- 6. Update the transition probability.

$$\begin{split} N(s,a) &= N(s,a) + 1, N(s,a,s') = N(s,a,s') + 1 \\ P(s'|s,a) &= N(s,a,s') / N(s,a) \end{split}$$

7. Update  $V^+(s)$  using the Bellman updates.

$$V^+(s) \leftarrow R(s) + \gamma \max_{a} f\left(\sum_{s,a} P(s'|s,a)V^+(s'), N(s,a)\right)$$

6 / 42 CS 486/686: In Slides S Alice Gao / Blak

#### An Active ADP Example

$s_{11}$	+1
$s_{21}$	-1

• 
$$\pi(s_{11}) = down, \pi(s_{21}) = right$$

$$\triangleright \gamma = 0.9$$

▶ 
$$N_e = 10, R^+ = 5.$$

$$R(s_{11}) = -0.04, R(s_{21}) = -0.04, R(s_{12}) = 1, R(s_{22}) = -1$$

$$\triangleright N(s,a) = 5, \forall s, a.$$

▶ 
$$N(s, a, s') = 3$$
 for the intended direction.

N(s, a, s') = 1 for any other direction with positive transition probability.

The current estimates:

$$V^+(s_{11}) = -0.6573, V^+(s_{21}) = -0.9002$$

CS 486/686: Intro to AlLecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen17 / 42

Learning Goals

Introduction to Reinforcement Learning

Passive Adaptive Dynamic Programming

Active Adaptive Dynamic Programming

Temporal Difference Error

**Q-Learning** 

#### SARSA

CS 486/686: Intro to AlLecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen18 / 42

## Bellman Equations for Q(s, a)

Q(s, a) is the expected value of performing action a in state s. We can define Bellman equations for both V(s) and Q(s, a).

Bellman equations for V(s):

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} P(s'|s, a) V(s')$$

Bellman equations for Q(s, a):

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

Learning V(s) and Q(s, a) are equivalent! What is the advantage of learning Q(s, a)?

CS 486/686: Intro to AlLecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen19 / 42

#### Temporal Difference Error

Assume that we observed  $\langle s_1, a, r_1, s_2 \rangle$ . Based on this transition, what should  $Q(s_1, a)$  satisfy?

Start with the Bellman equations for  $Q(s_1, a)$ .

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

 $Q(s_1, a)$  should be computed by the RHS of the above equation. Assume that this transition always occurs ( $P(s_2|s_1, a) = 1$ ). Thus,  $Q(s_1, a)$  should be

$$R(s_1) + \gamma \max_{a'} Q(s_2, a')$$

Temporal difference (TD) error:

$$(R(s_1) + \gamma \max_{a'} Q(s_2, a')) - Q(s_1, a)$$

CS 486/686: Intro to AlLecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen20 / 42

Learning Goals

Introduction to Reinforcement Learning

Passive Adaptive Dynamic Programming

Active Adaptive Dynamic Programming

Temporal Difference Error

**Q-Learning** 

#### SARSA

CS 486/686: Intro to AlLecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen21 / 42

#### Q-Learning Updates

Given an experience  $\langle s,a,s',r\rangle$  , update Q(s,a) as follows:

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

An alternative version:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left( R(s) + \gamma \max_{a'} Q(s',a') \right)$$

CS 486/686: Intro to AlLecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen22 / 42

#### Passive Q-Learning Algorithm

- 1. Repeat steps 2 to 4.
- 2. Follow policy  $\pi$  and generate an experience  $\langle s, a, s', r \rangle$ .
- 3. Update reward function:  $R(s) \leftarrow r$
- 4. Update Q(s, a) by using the temporal difference update rules:

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

The learning rate  $\alpha$ :

 $\alpha$  controls the size of each update. If  $\alpha$  decreases as N(s,a) increases, Q values will converge to the optimal values. For example,  $\alpha(N(s,a)) = \frac{10}{9+N(s,a)}.$ 

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen23 / 42

## Active Q-Learning Algorithm

(with optimistic utilities for exploration)

- 1. Initialize R(s), Q(s, a), N(s, a).
- 2. Repeat steps 3 to 6 until we have visited each (s, a) at least  $N_e$  times and the Q(s, a) values have converged.
- 3. Determine the best action a for current state s using  $V^+(s)$ .

$$a = \arg \max_{a} f\left(Q(s, a), N(s, a)\right), \ f(u, n) = \begin{cases} R^+, \ \text{if } n < N_e \\ u, \text{otherwise} \end{cases}$$

- 4. Take action a and generate an experience  $\langle s,a,s',r\rangle$
- 5. Update reward function:  $R(s) \leftarrow r$
- 6. Update Q(s, a) using the temporal difference update rules.

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

CS 486/686: Intro to AILecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen24 / 42

#### Active Q-Learning Algorithm

(with  $\epsilon$ -greedy for exploration)

- 1. Initialize  $R(s), Q(s, a), \epsilon$ .
- 2. Repeat steps 3 to 6 until the Q(s, a) values have converged.
- 3. Determine the action a for the current state:

$$a = \begin{cases} \text{random action, with probability } \epsilon \\ \arg \max_a Q(s, a), \text{with probability } 1 - \epsilon \end{cases}$$

- 4. Take action a and generate an experience  $\langle s,a,s',r\rangle$
- 5. Update reward function:  $R(s) \leftarrow r$
- 6. Update Q(s, a) using the temporal difference update rules.

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

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen25 / 42

#### Active Q-Learning Algorithm

(with softmax sampling for exploration)

- 1. Initialize R(s), Q(s, a), T.
- 2. Repeat steps 3 to 6 until the Q(s, a) values have converged.
- 3. Determine the action a for the current state:

$$a \sim \langle P(a_0), P(a_1), \dots \rangle = \frac{\langle e^{Q(s,a_0)/T}, e^{Q(s,a_1)/T}, \dots \rangle}{\sum_{i=0}^{|\mathcal{A}|-1} e^{Q(s,a_i)/T}}$$

- 4. Take action a and generate an experience  $\langle s,a,s',r\rangle$
- 5. Update reward function:  $R(s) \leftarrow r$
- 6. Update Q(s, a) using the temporal difference update rules.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \bigg( R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \bigg)$$

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen26 / 42

# Active Q-Learning: Example Iteration

Recall the grid world from previous lectures:

	1	2	3	4
1				
2		Х		-1
3				+1

On iteration i, the agent generates the following experience:  $\langle s,a,s',r\rangle=\langle s_{32},{\rm right},s_{33},-0.04\rangle$ 

The current Q-values are as follows:

	$s_{11}$	$s_{12}$	$s_{13}$	$s_{14}$	$s_{21}$	$s_{23}$	$s_{31}$	$s_{32}$	$s_{33}$
up	0.1	0.2	0.25	0.3	0.27	0.3	0.3	0.55	0.4
right	0.1	0.15	0.2	0.32	0.39	-0.8	0.5	0.7	0.9
down	0.2	0.24	0.2	0.45	-0.8	0.4	0.45	0.5	0.8
left	0.1	0.15	0.22	0.2	0.27	0.05	0.3	0.4	0.5

Execute the Q-learning update. Say  $\gamma = 1$  and  $\alpha = 0.1$ .

CS 486/686: Intro to AlLecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen27 / 42

### Properties of Q-Learning

1. Learns Q(s, a) instead of V(s).

- 2. **Model-free:** no need to learn the transition probabilities P(s'|s, a).
- 3. Learns an approximation of the optimal Q-values as long as the agent explores sufficiently.
- 4. The smaller  $\alpha$  is, the closer it will converge to the optimal Q-values, but the slower it will converge.

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen28 / 42

# ADP v.s. Q-Learning

1. Requires learning the transition probabilities?

 $\rightarrow$  ADP requires learning the transition probabilities. Q-learning does not.

## ADP v.s. Q-Learning

1. Requires learning the transition probabilities?

 $\rightarrow$  ADP requires learning the transition probabilities. Q-learning does not.

2. How much computation is performed per experience?

 $\rightarrow$  ADP requires more computation per experience. It tries to maintain the consistency in the utility values between neighbouring states using Bellman equations.

Q-learning requires less memory and computation time.

# ADP v.s. Q-Learning

1. Requires learning the transition probabilities?

 $\rightarrow$  ADP requires learning the transition probabilities. Q-learning does not.

2. How much computation is performed per experience?

 $\rightarrow$  ADP requires more computation per experience. It tries to maintain the consistency in the utility values between neighbouring states using Bellman equations.

Q-learning requires less memory and computation time.

3. How fast does it learn?

 $\rightarrow$  ADP converges much faster than Q-learning. Q-learning learns slower and shows much higher variability.

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen29 / 42

Learning Goals

Introduction to Reinforcement Learning

Passive Adaptive Dynamic Programming

Active Adaptive Dynamic Programming

Temporal Difference Error

**Q-Learning** 

#### SARSA

CS 486/686: Intro to AILecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen30 / 42

#### SARSA Updates

**SARSA** update rule: Given an experience  $\langle s, a, r, s', a' \rangle$ , update Q(s, a) as follows:

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

where a' is the actual action taken in state s'.

**Q-learning** update rule: Given an experience  $\langle s, a, s', r \rangle$ , update Q(s, a) as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left( r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

where a' is the optimal action in state s' given current Q values.

CS 486/686: Intro to AlLecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen31 / 42

#### Active SARSA Algorithm

1. Initialize R(s), Q(s, a).

- 2. Repeat steps 3 to 8 until the Q(s, a) values have converged.
- If starting new episode, determine a for initial state s<sub>0</sub> using current policy (determined by exploration strategy). s ← s<sub>0</sub>.
- 4. Take action a and generate an experience  $\langle s, a, r, s' \rangle$ .
- 5. Update reward function:  $R(s) \leftarrow r$
- 6. Determine action a' for state s' using current policy.
- 7. Update Q(s, a) using the temporal difference update rules.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left( R(s) + \gamma Q(s',a') - Q(s,a) \right)$$

8. Update  $a \leftarrow a'$ ,  $s \leftarrow s'$ .

CS 486/686: Intro to AILecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen32 / 42

## Active SARSA: Example Iteration

Recall the grid world from previous lectures:

	1	2	3	4
1				
2		Х		-1
3				+1

On iteration i, the agent generates the following experience:  $\langle s,a,r,s',a'\rangle = \langle s_{13},{\rm down},-0.04,s_{23},{\rm right}\rangle$ 

The current Q-values are as follows:

	$s_{11}$	$s_{12}$	$s_{13}$	$s_{14}$	$s_{21}$	$s_{23}$	$s_{31}$	$s_{32}$	$s_{33}$
up	0.1	0.2	0.25	0.3	0.27	0.3	0.3	0.55	0.4
right	0.1	0.15	0.2	0.32	0.39	-0.8	0.5	0.7	0.9
down	0.2	0.24	0.2	0.45	-0.8	0.4	0.45	0.5	0.8
left	0.1	0.15	0.22	0.2	0.27	0.05	0.3	0.4	0.5

Execute the SARSA update. Say  $\gamma = 1$  and  $\alpha = 0.1$ .

CS 486/686: Intro to AlLecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen33 / 42

# Q-Learning vs. SARSA

- Q-learning is off-policy whereas SARSA is on-policy.
- For a greedy agent, they are the same. If the agent explores, they are significantly different.
- Q-learning is more flexible: It learns to behave well even when the exploration policy is random or adversarial.
- SARSA is more realistic: It can avoid exploration with large penalties. It learns what will actually happen instead of what the agent would like to happen.
- Q-learning is more appropriate for offline learning when the agent does not explore. SARSA is more appropriate when the agent explores.
- https://towardsdatascience.com/ intro-to-reinforcement-learning-temporal-difference-le

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen34 / 42

**Q #2:** In a high-stakes online learning problem (e.g. self-driving car), would it be better to use Q-learning or SARSA?

- (A) Q-learning
- (B) SARSA
- (C) I don't know

**Q #2:** In a high-stakes online learning problem (e.g. self-driving car), would it be better to use Q-learning or SARSA?

- (A) Q-learning
- (B) SARSA
- (C) I don't know

 $\rightarrow$  Correct answer is (B). SARSA is often more likely to learn less risky policies.

- **Q #3:** Which of the following is true?
- (A) Q-learning is model-free and off-policy, while SARSA is model-based and on-policy.
- (B) Q-learning is model-based and off-policy, while SARSA is model-free and on-policy.
- (C) Q-learning is model-free and on-policy, while SARSA is model-based and off-policy.
- (D) Q-learning is model-based and on-policy, while SARSA is model-free and off-policy.
- (E) None of the above.

- **Q #3:** Which of the following is true?
- (A) Q-learning is model-free and off-policy, while SARSA is model-based and on-policy.
- (B) Q-learning is model-based and off-policy, while SARSA is model-free and on-policy.
- (C) Q-learning is model-free and on-policy, while SARSA is model-based and off-policy.
- (D) Q-learning is model-based and on-policy, while SARSA is model-free and off-policy.
- (E) None of the above.

 $\rightarrow$  Correct answer is (E). Q-learning is off-policy and SARSA is on-policy. Both are model-free.

Let's assume the Q function is:					
	1	2	3	4	
1			$Q_{13}$		
2		Х	$Q_{23}$	-1	
3				+1	

Assuming we are at  $s_{13}$  and we want to update  $Q(s_{13}, down)$ .  $Q_{13}$  is (left=0.3, right=0.1, down=0.4, up=0.1).  $Q_{23}$  is (left=0.3, right=-0.4, down=0.7, up=0.3).

According to SARSA, we take (s, a, r, s', a') according to policy π. So a' ~ π(s), which we assume uses ε-greedy.

▶ a' is not necessarily  $argmax_{a'}Q(s_{23}, a')$ , which is going down.

CS 486/686: Intro to Al Lecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen37 / 42

Let's assume the Q function is.					
	1	2	3	4	
1			$Q_{13}$		
2		Х	$Q_{23}$	-1	
3				+1	

Lat's assume the O function is

Assuming we are at  $s_{13}$  and we want to update  $Q(s_{13}, down)$ .  $Q_{13}$  is (left=0.3, right=0.1, down=0.4, up=0.1).  $Q_{23}$  is (left=0.3, right=-0.4, down=0.7, up=0.3).

- Let's assume that a' happens to sample 'right'.
- We update using SARSA rule:  $Q(s_{13}, down) = Q(s_{13}, down) + \alpha(r + \gamma Q(s', a') - Q(s, a)).$
- We update using SARSA rule:  $Q(s_{13}, down) = 0.4 + 0.4 * (-0.04 + 0.9 * -0.4 0.4) = 0.265$

CS 486/686: Intro to AlLecturer: Yuntian Deng

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen38 / 42

Let's assume the Q function is:					
	1	2	3	4	
1			$Q_{13}$		
2		Х	$Q_{23}$	-1	
3				+1	

Assuming we are at  $s_{13}$  and we want to update  $Q(s_{13}, down)$ .  $Q_{13}$  is (left=0.3, right=0.1, down=0.4, up=0.1).  $Q_{23}$  is (left=0.3, right=-0.4, down=0.7, up=0.3).

- Now  $Q_{13}$  is (left=0.3, right=0.3, down=0.26, up=0.1).
- The best move becomes 'Moving Left'.
- We can see that 'SARSA' is pretty conservative.

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen39 / 42

Let's assume the Q function is:				
	1	2	3	4
1			$Q_{13}$	
2		Х	$Q_{23}$	-1
3				+1

Assuming we are at  $s_{13}$  and we want to update  $Q(s_{13}, down)$ .  $Q_{13}$  is (left=0.3, right=0.1, down=0.4, up=0.1).  $Q_{23}$  is (left=0.3, right=-0.4, down=0.7, up=0.3).

According to Q-Learning, we take (s, a, r, s') according to policy π.

▶ a' is forced to  $argmax_{a'}Q(s_{23}, a')$ , which is going down.

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen40 / 42

Let's assume the Q function is:						
	1	2	3	4		
1			$Q_{13}$			
2		Х	$Q_{23}$	-1		
3				+1		

Assuming we are at  $s_{13}$  and we want to update  $Q(s_{13}, down)$ .  $Q_{13}$  is (left=0.3, right=0.1, down=0.4, up=0.1).  $Q_{23}$  is (left=0.3, right=-0.4, down=0.7, up=0.3).

- We update using SARSA rule:  $Q(s_{13}, down) = Q(s_{13}, down) + \alpha(r + \gamma max_{a'}Q(s', a') Q(s, a)).$
- We update using SARSA rule:  $Q(s_{13}, down) = 0.4 + 0.4 * (-0.04 + 0.9 * 0.7 - 0.4) = 0.476$

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Va

Slides: Alice Gao / Blake Vanberlo / Wenhu Chen41 / 42

Let's assume the Q function is:						
	1	2	3	4		
1			$Q_{13}$			
2		Х	$Q_{23}$	-1		
3				+1		

Assuming we are at  $s_{13}$  and we want to update  $Q(s_{13}, down)$ .  $Q_{13}$  is (left=0.3, right=0.1, down=0.4, up=0.1).  $Q_{23}$  is (left=0.3, right=-0.4, down=0.7, up=0.3).

- Now  $Q_{13}$  is (left=0.3, right=0.1, down=0.476, up=0.1).
- ► The best move of 'Moving Down' is enhanced.
- We can see that 'Q-Learning' is pretty aggressive.

CS 486/686: Intro to AILecturer: Yuntian Deng Slides: Alice Gao / Blake Vanberlo / Wenhu Chen42 / 42