L17. Reinforcement Learning
(Temporal Difference Methods)

1. SARSA
2. Q-learning

Introduction

Dynamic Programming (DP) Methods

pros: Update estimates based in part on other learned estimates, without waiting for a final outcome (bootstrap)

cons: Need explicit model

Monte Carlo (MC) Methods

pros: Learn directly from raw experience without a model

cons: Need to wait until the end of episode to observe expected reward

Temporal-Difference (TD) Learning

pros: Learn directly from raw experience without a model

MC

+

pros: Update estimates based in part on other learned estimates, without waiting for a final outcome (bootstrap)

DP

Model free

On line
Incremental

TD Generalized Policy iteration for

TD Policy Evaluation + **TD Policy Improvement**

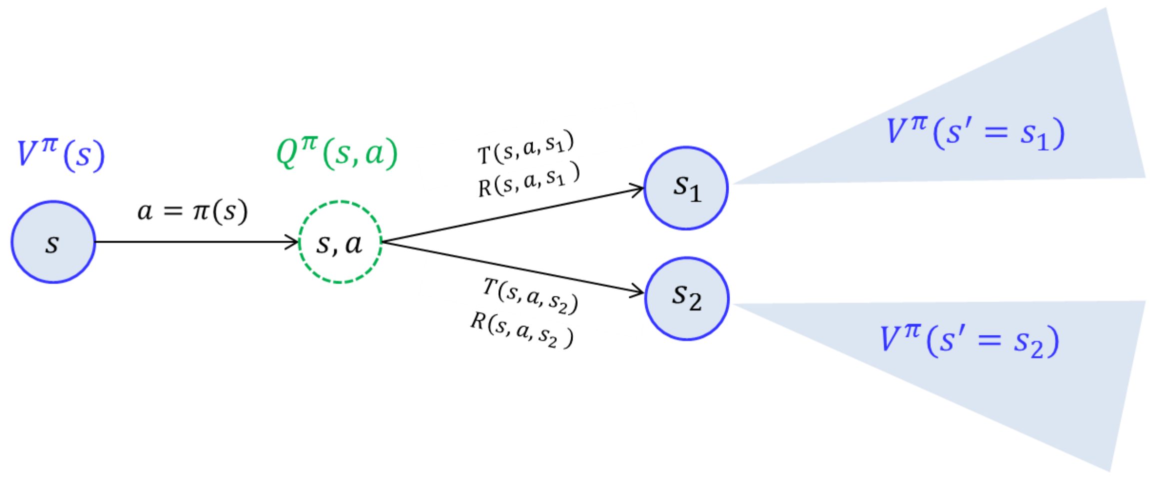
On-Policy TD Control (SARSA)

Off-Policy Q-learning Control

Recall : Value function

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}(U_t | s_t = s) \\
&= \mathbb{E}_{\pi}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s) \quad \text{Complete episode} \\
&= \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s\right) \\
&= \mathbb{E}_{\pi}\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s\right) \\
&= \mathbb{E}_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s)
\end{aligned}
$$

Bootstrapping



Monte Carlo Policy Evaluation

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}(U_t | s_t = s) \\
&= \mathbb{E}_{\pi}\left(\sum_{k=0}^{T} \gamma^k r_{t+k+1} \; | s_t = s\right) \\
&= \mathbb{E}_{\pi}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s)
\end{aligned}
$$

A sampled episode

Constant-α MC :

After visiting s_t and receiving utility $u_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$

$$V(s_t) \leftarrow V(s_t) + \alpha[u_t - V(s_t)]$$
$$V(s_t) \leftarrow V(s_t) + \alpha[\underbrace{r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T}_{\text{Target}} - V(s_t)]$$

- The target of update is $u_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$
- A sample reward u_t from a single episode is used for representing the expected reward. If the episode is long, u_t will be a lousy estimate (a single initialization)
- This is estimate because we use sampled value instead of expected utility

Temporal Difference Policy Evaluation

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}(U_t | s_t = s) \\
&= \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\, s_t = s\right) \\
&= \mathbb{E}_{\pi}\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\, s_t = s\right) \\
&= \mathbb{E}_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s)
\end{aligned}
$$

Bootstrapping

Temporal Difference Policy Evaluation ;$\boldsymbol{TD(0)}$:

After visiting s_t and transiting to s_{t+1} with a singe reward r_{t+1}

$$V(s_t) \leftarrow V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{Target}} - V(s_t)]$$

- Bootstrapping: the TD method updates the state value using the previous estimations
- The TD target is an estimate because
  - ✓ it uses the current estimate of $V(s_t)$,
  - ✓ it samples the expected value

$\mathbb{E}_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s)$

Algorithm : Tabular $TD(0)$ for estimating V^{π}

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated

Repeat (for each episode):
  Initialize s
  Repeat (for **each step** of episode)
  $a \leftarrow$ action given by π for s
  Take action a; observe reward r and next state s'
  $V(s) \leftarrow \mathrm{V(s)} + \alpha[r + \gamma V(s') - V(s)]$
  s ←s'
  Until s is terminal

- Simple backups (MC method and TD methods) : Use a single sample success state

Recall:

- Full Backups (DP approach) : Use complete distribution of all possible successors

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^{\pi}(s')\}$$

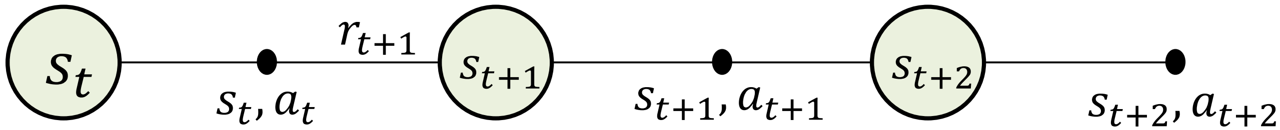
Advantages of TD Policy Evaluation (prediction)

What advantages do TD methods have over Monte Carlo and DP methods?

- TD methods learn their estimates on the basis of other estimates (Bootstrap)
- TD methods do not require a model of the environment, i.e., reward and state transition models
- TD methods can be naturally implemented in an **on-line**, fully incremental fashion:
  - ✓ Monte Carlo Method **must wait until the end of an episode**, because only then the return is revealed
  - ✓ TD methods operates with **a single transition of state and action** (a single time step)→ advantages for continuous task and learning
- TD methods and Monte Carlo methods converge to V^{π} in the mean for a constant step-size if it is sufficiently small, and with probability 1 if the step-size parameter decreases
- In practice, TD methods have usually been found to converge faster than $constnt - \alpha$ MC methods on stochastic tasks

Temporal Difference Policy Evaluation for Q function

As we estimate state value $V(s)$, we can estimate $Q(s,a)$ using a TD method



Temporal Difference Policy Evaluation for $Q(s,a)$ function

On each $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ for a single episode:

Note that the action taken is given as data

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\underbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})}_{\text{Target}} - \underbrace{Q(s_t, a_t)}_{\text{Current estimate}}]$$

TD Generalized Policy iteration for

TD Policy Evaluation + **TD Policy Improvement**

- **On-Policy TD Control (SARSA)**
- **Off-Policy TD Control (Q-learning)**

Estimation and prediction problem

Decision making problems

SARSA Algorithm

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
Initialize s
Choose a from s using policy derived from Q (e.g., $\epsilon - greedy$)
Repeat (for **each time step** of episode):

$$\left\{\begin{array}{l} \text{Take action } a \text{ given } s \text{, observe } r, s' \\ \text{Choose } a' \text{ from } s' \text{ using policy derived from } Q \text{ (e.g., } \epsilon - greedy) \\ Q_\pi(s, a) \leftarrow Q_\pi(s, a) + \eta\left(r + \gamma Q_\pi(s', a') - Q_\pi(s, a)\right) \\ s \leftarrow s'; a \leftarrow a'; \end{array}\right.$$

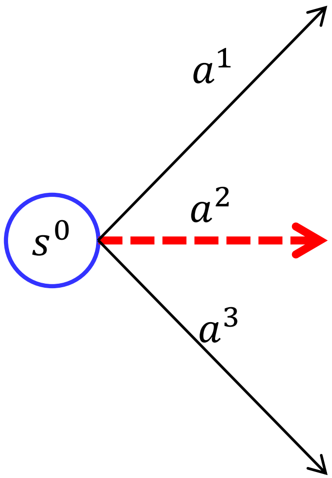
Behavioral policy
II
Estimation policy

Until s is terminal

- As in all on-policy methods, we continually estimate Q^π for the behavioral policy, and the same time change π toward greediness with respect to Q^π
- Converges with
  - ✓ All state-action pairs are visited an infinite number of times
  - ✓ The policy converges in the limit to the greedy policy (i.e., $\epsilon - greedy$ with $\epsilon = 1/t$)

Sarsa: On-Policy TD Control

S_t A_t R_{t+1} S_{t+1} A_{t+1} S_{t+2}

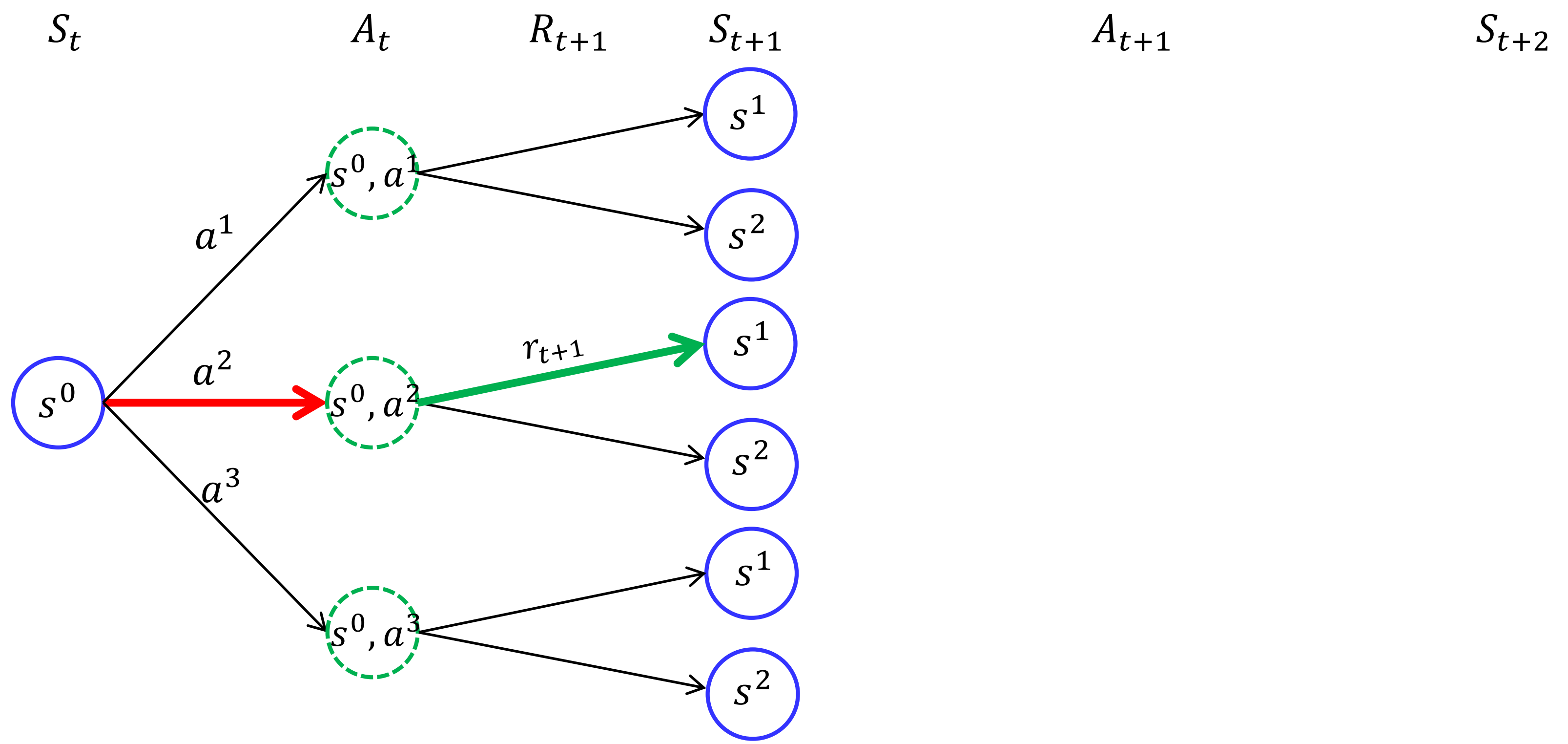


Choose a_t from $s_t = s^0$ using current Q

$$a_t = \begin{cases} \operatorname*{argmax}_a Q(s_t = s^0, a) & \text{with prob } 1-\epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

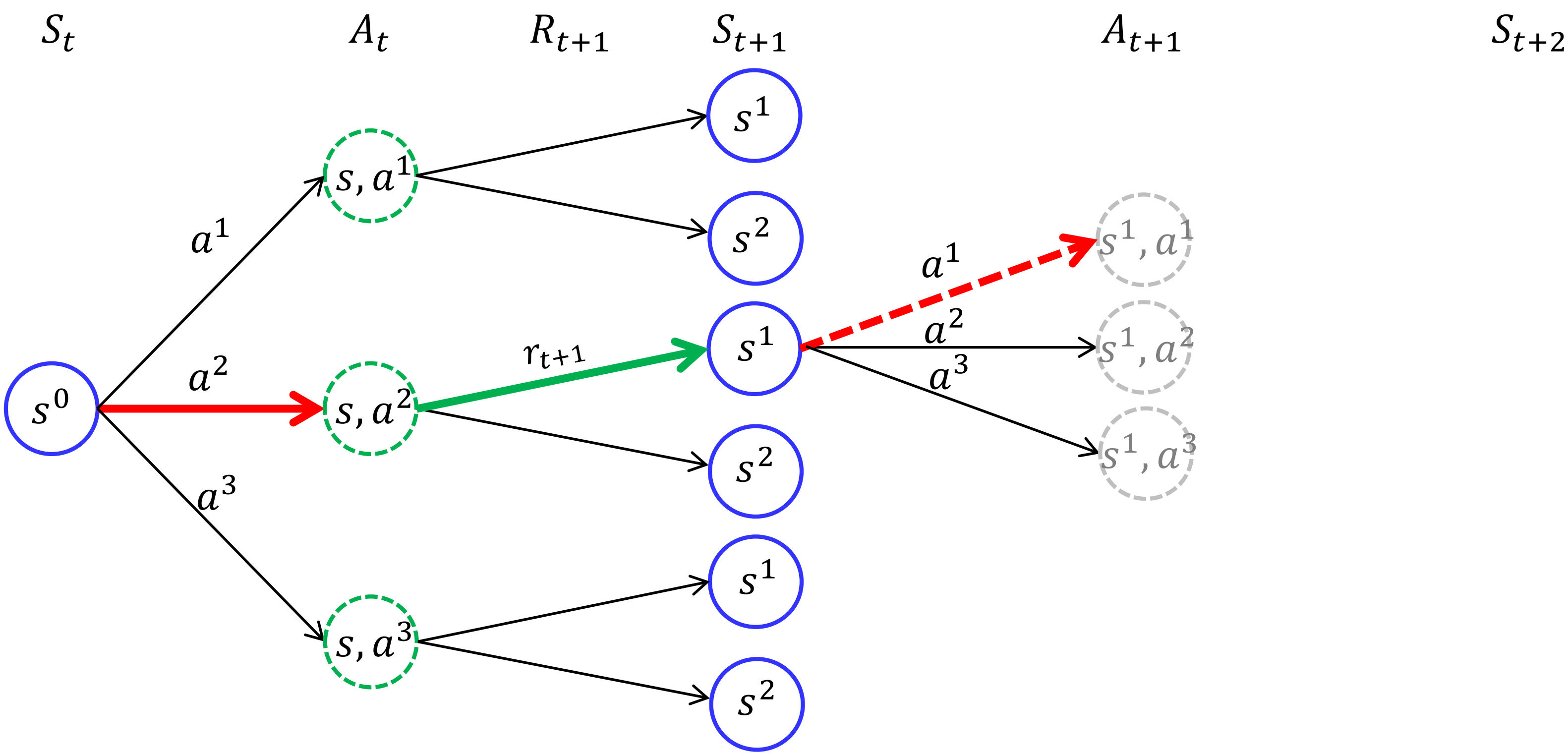
Assume a^2 is chosen

Sarsa: On-Policy TD Control



Take action $a_t = a^2$ given $s_t = s^0$ and observe r_{t+1} and $s_{t+1} = s^1$

Sarsa: On-Policy TD Control

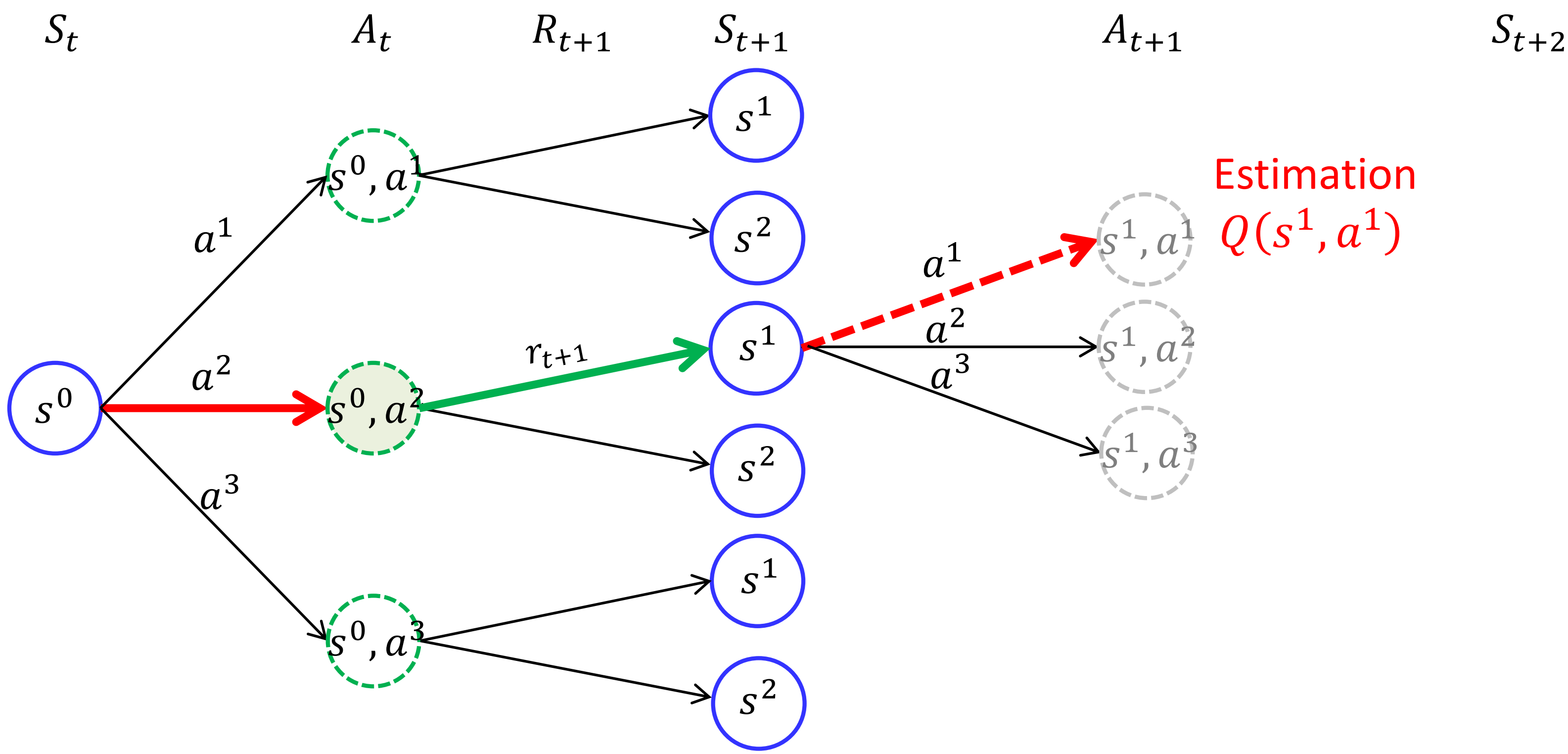


Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

$$a_{t+1} = \begin{cases} \arg\max_a Q(s_{t+1} = s^1, a) & \text{with prob } 1 - \epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

Assume a^1 is chosen

Sarsa: On-Policy TD Control

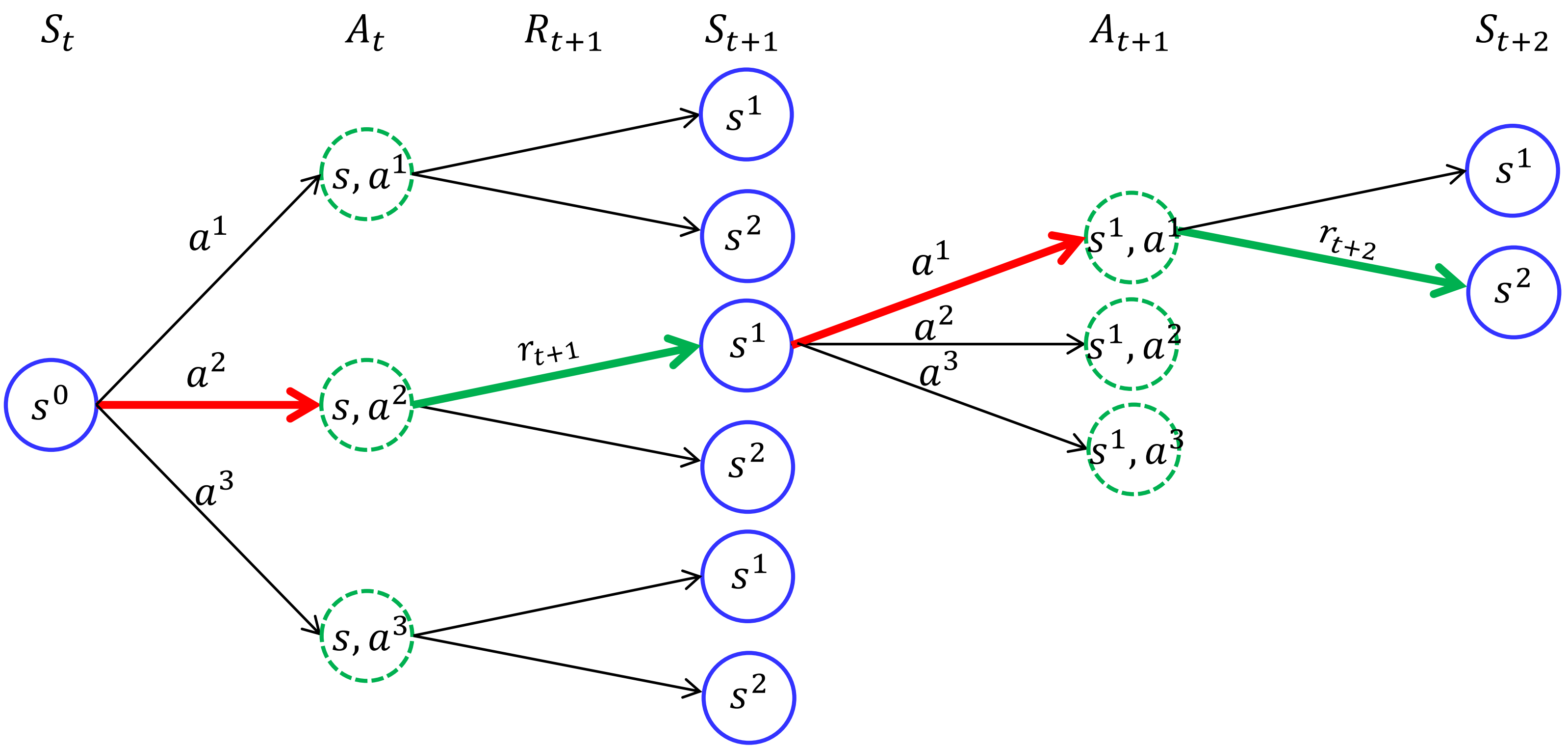


Update Q function with the estimation $Q(s_{t+1}, a_{t+1})$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

$$\rightarrow Q(s^0, a^2) \leftarrow Q(s^0, a^2) + \alpha[r_{t+1} + \gamma Q(s^1, a^1) - Q(s^0, a^2)]$$

Sarsa: On-Policy TD Control



Take action $a_{t+1} = a^1$ given $s_{t+1} = s^1$ and observe r_{t+2} and $s_{t+2} = s^2$

Why Q-learning is considered as Off-Policy method

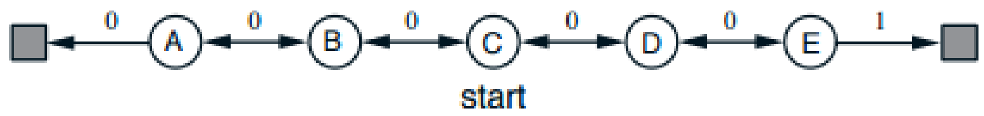
$$\pi(s) \qquad \pi(s')$$

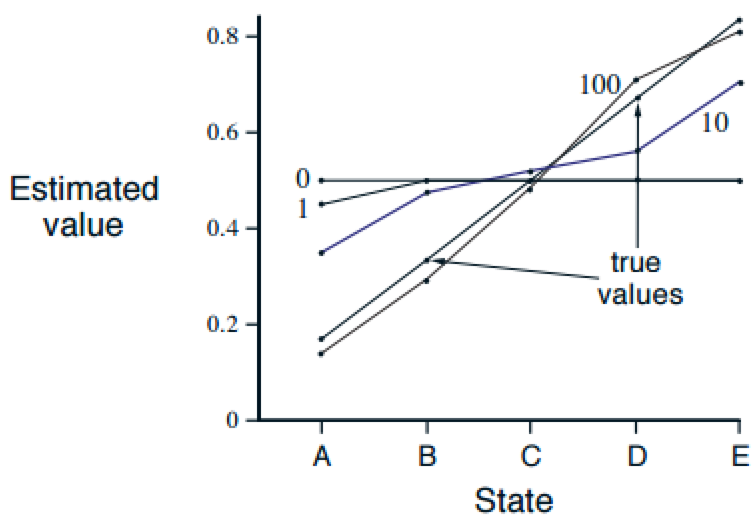
$$(s, a, r, s', a')$$

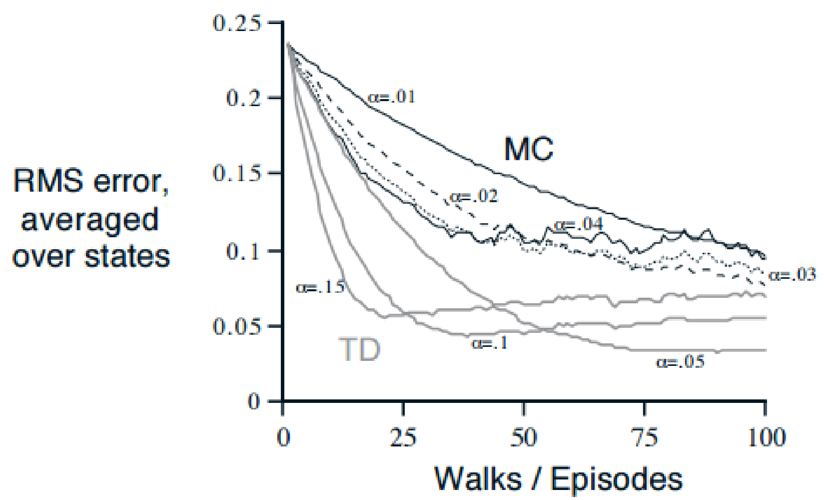
$$Q(s,a) \leftarrow Q(s,a) + \eta\big(r + \gamma Q(s',a') - Q_\pi(s,a)\big)$$

Sarsa: On-Policy TD Control : Windy Grid world Example

A small Markov process for generating random walk

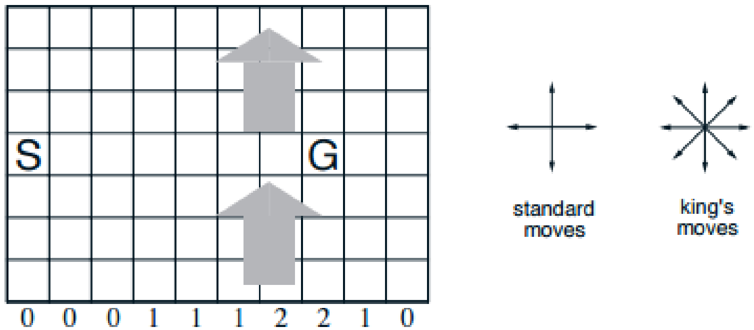


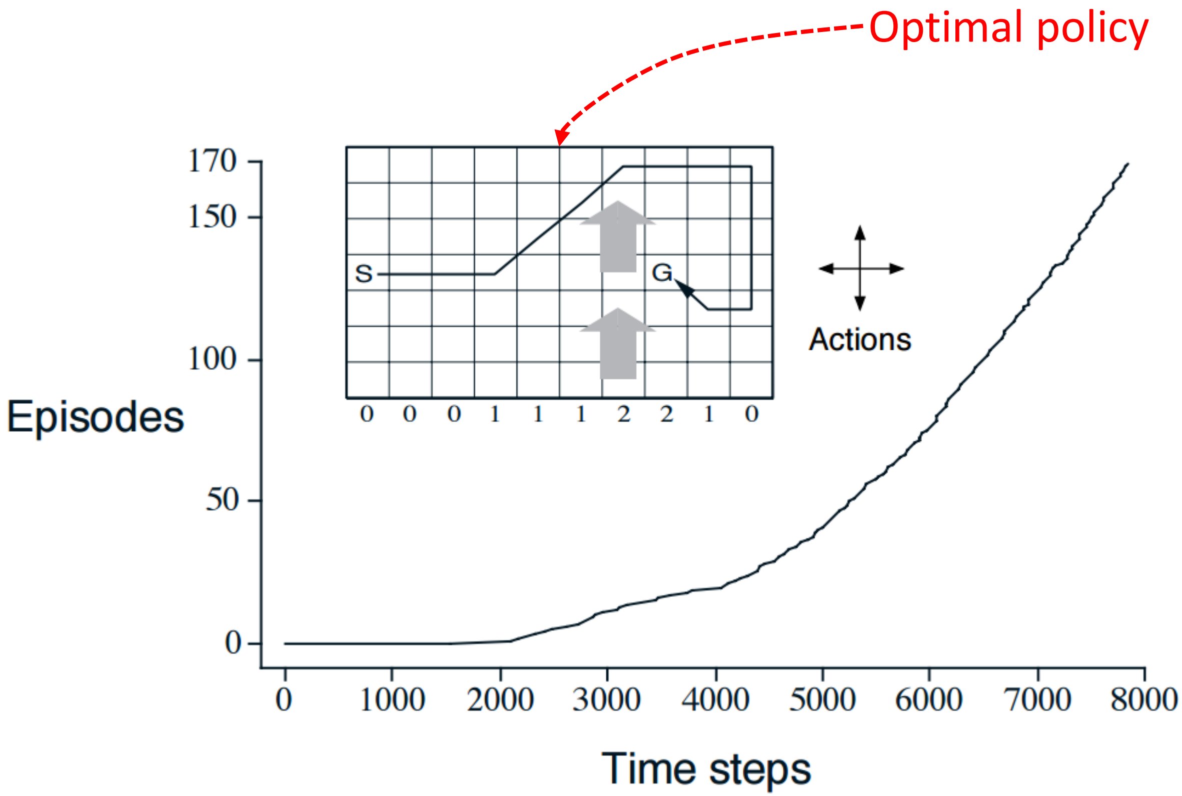




Sarsa: On-Policy TD Control : Windy Grid world Example

State transition is encoded by this figure





Classification of RL

How to estimate $V^*(s)$ and $Q^*(s,a)$

Monte Carlo method — **Temporal Difference methods**

How to explore ?		Non-Bootstrap	Bootstrap
	On-policy	On-policy Monte Carlo Control	SARSA
	Off-policy	Off-policy Monte Carlo Control	Q-Learning (SARSmaxA)

- Episodic based
- Single-data-point based

Q-Learning: Off-Policy TD Control

On-Policy TD Control (SARSA)

Choose a' from s' using policy derived from Q (e.g., $\epsilon - greedy$)

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

Off-Policy TD Control (Q-learning)

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

- The max over a rather than taking the a based on the current policy is the principle difference between Q-learning and SARSA.
- The learned action-value function Q directly approximates Q^*independent of the policy being followed
- Converges with
  - ✓ All state-action pairs are visited an infinite number of times
  - ✓ The policy converges in the limit to the greedy policy (i.e., $\epsilon - greedy$ with $\epsilon = 1/t$)

Q-Learning: Off-Policy TD Control

Q learning

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each time step of episode):
    Choose a from s using policy derived from Q (e.g., $\epsilon - greedy$) Behavioral policy
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
    $s \leftarrow s'$
  Until s is terminal

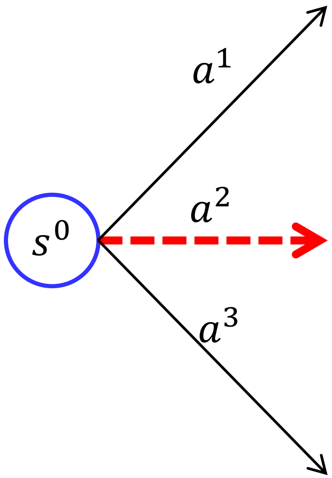
Estimation policy
(Always try to estimate the optimal policy)
-Estimation can be greedy)

$a^* = \operatorname{argmax}_{a'} Q(s',a)$ is not used in the next state!!!

At the next state s', Choose a using policy derived from Q (e.g., $\epsilon - greedy$)

Q-Learning: Off-Policy TD Control

S_t A_t R_{t+1} S_{t+1} $\max A_{t+1}$ S_{t+2}

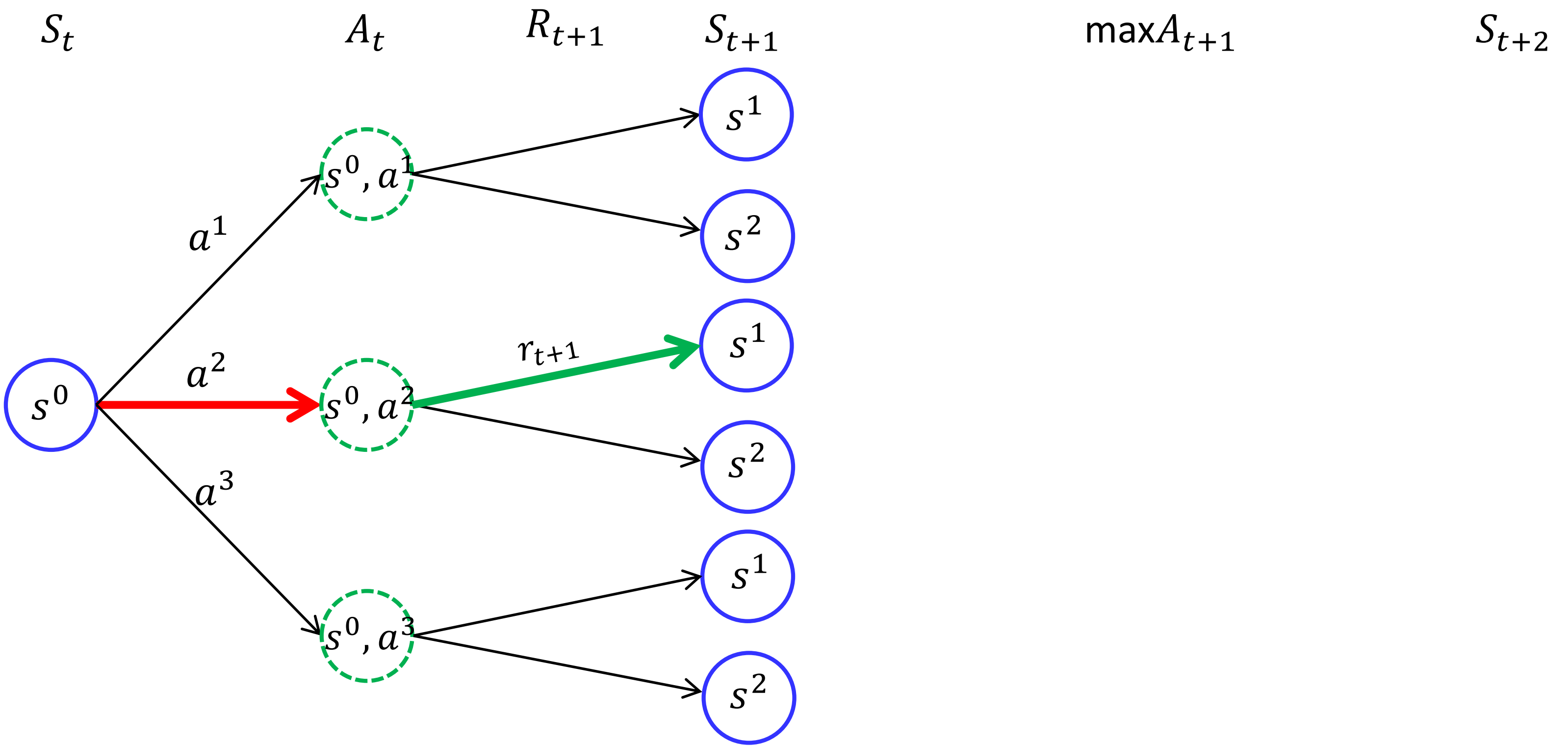


Choose a_t from $s_t = s^0$ using current Q

$$a_t = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(s_t = s^0, a) & \text{with prob } 1-\epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

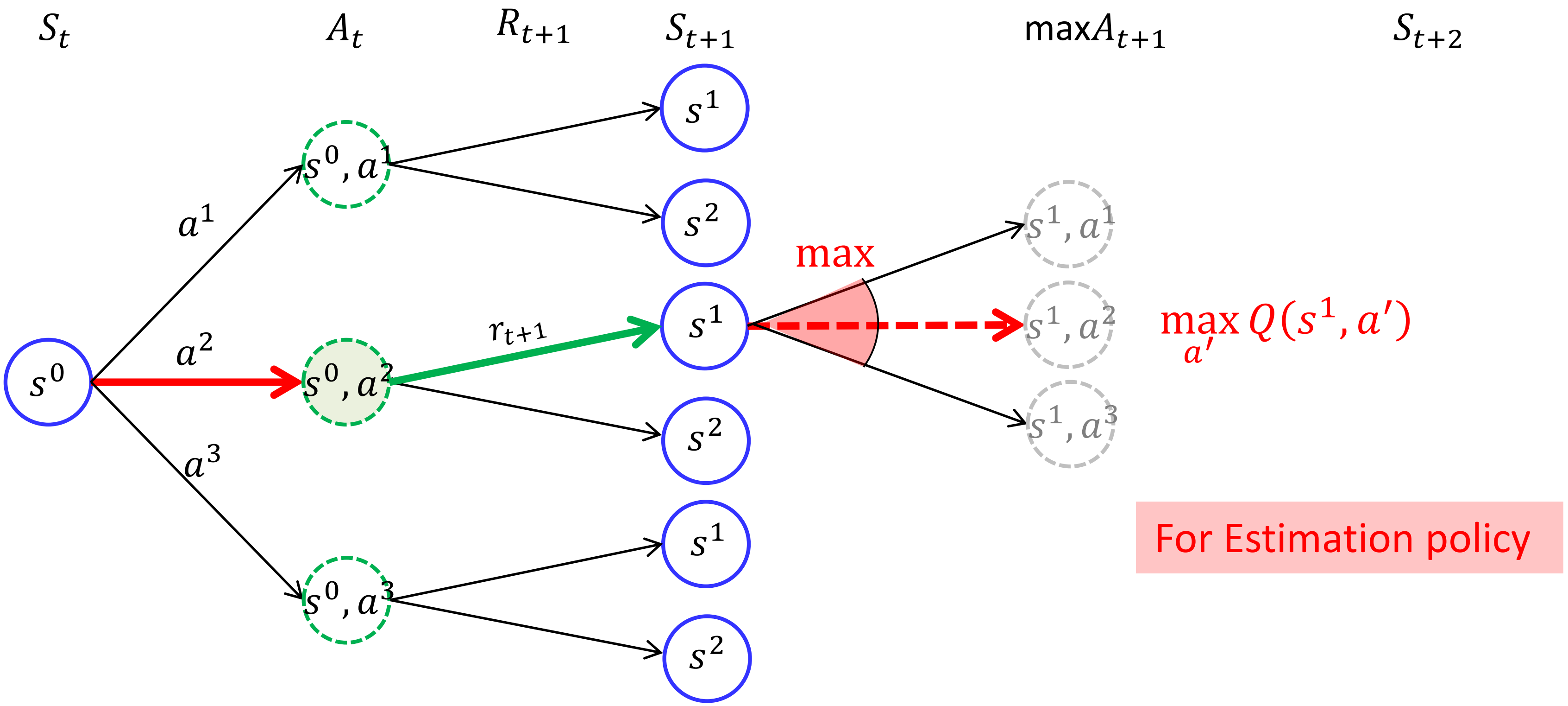
Assume a^2 is chosen

Q-Learning: Off-Policy TD Control



Take action $a_t = a^2$ given $s_t = s^0$ and observe r_{t+1} and $s_{t+1} = s^1$

Q-Learning: Off-Policy TD Control

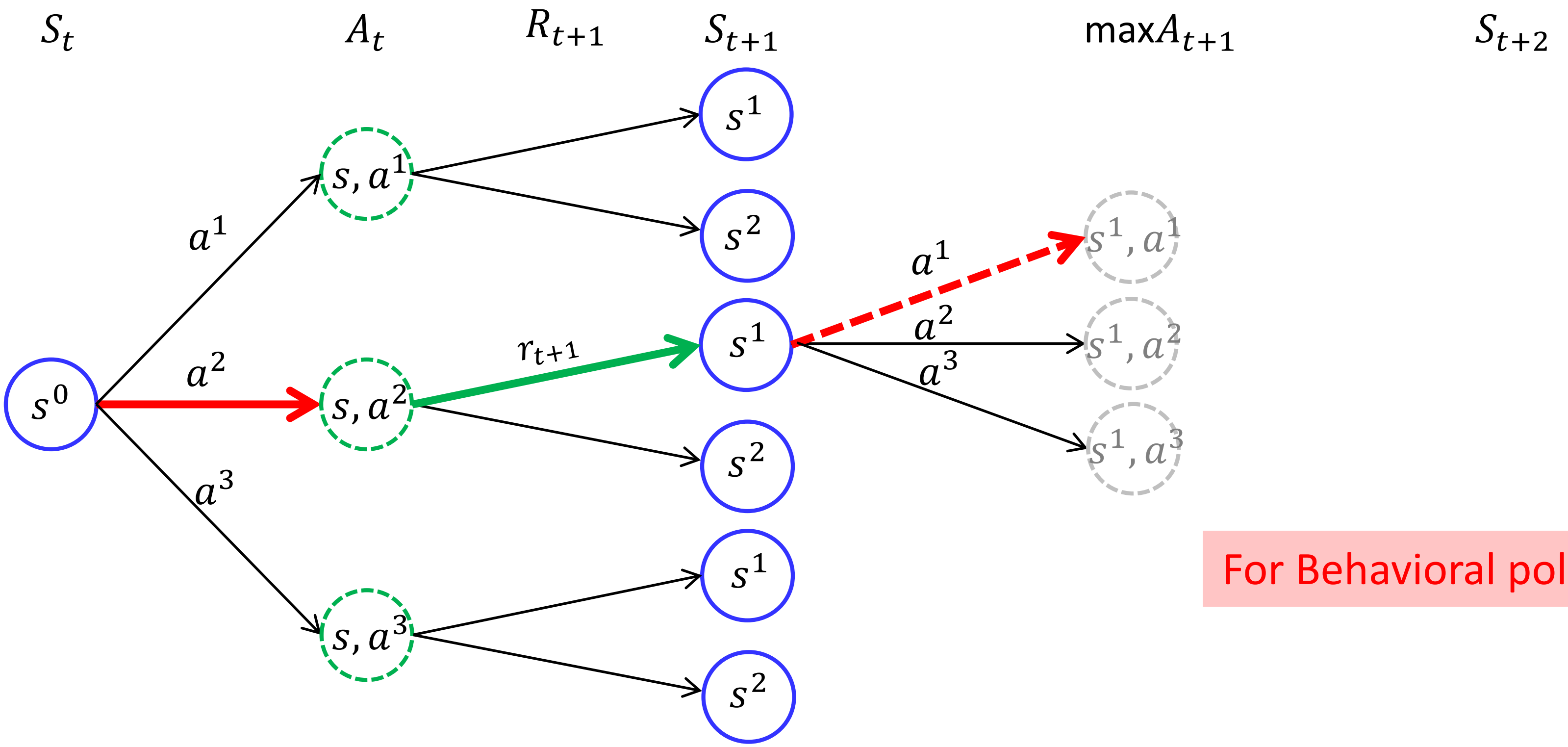


Update Q function with the $\max_{a'} Q(s^1, a')$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s, a') - Q(s_t, a_t) \right]$$

$$\rightarrow Q(s^0, a^2) \leftarrow Q(s^0, a^2) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s^1, a') - Q(s^0, a^2) \right]$$

Q-Learning: Off-Policy TD Control



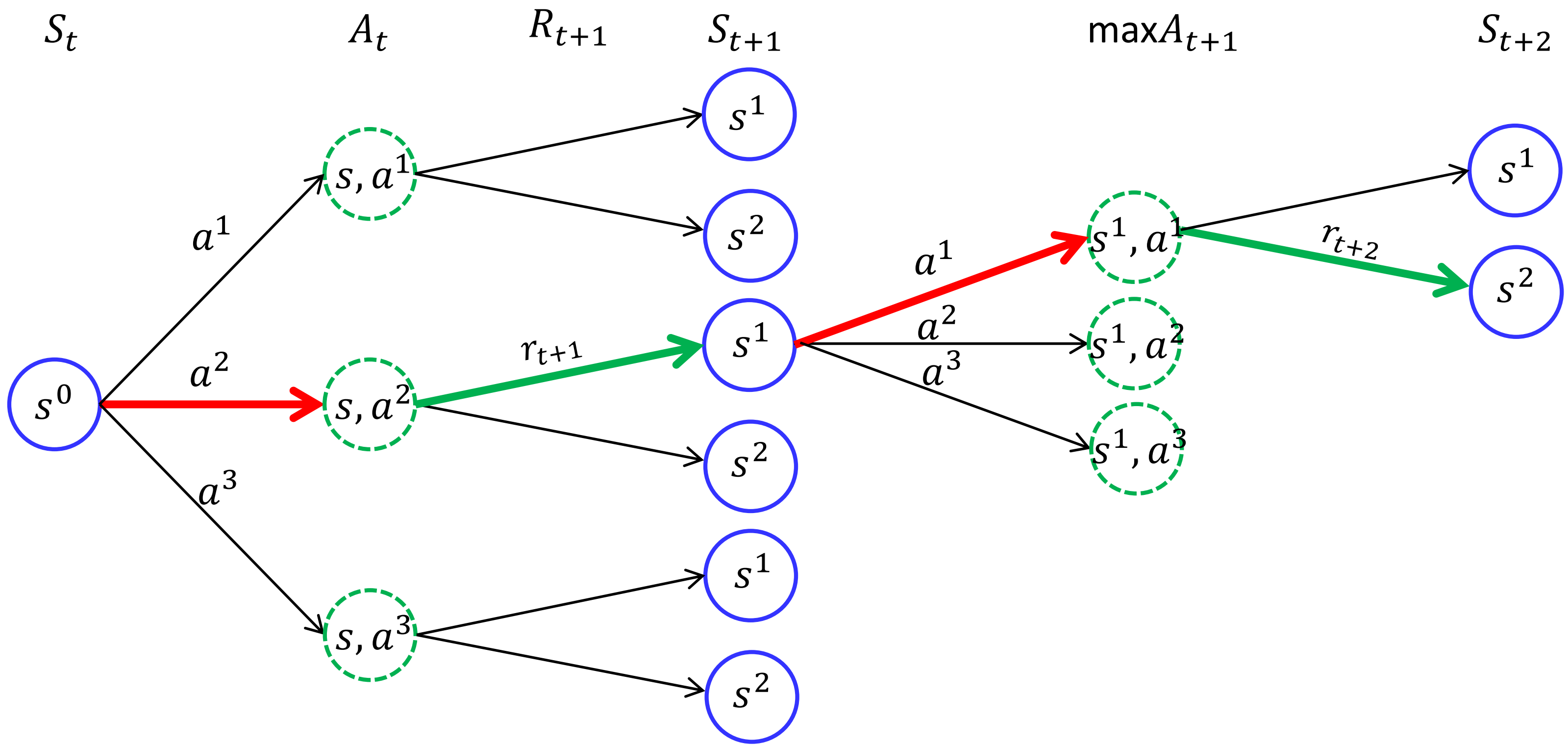
For Behavioral policy

Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

$$a_{t+1} = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(s_{t+1} = s^1, a) & \text{with prob } 1 - \epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

Assume a^1 is chosen

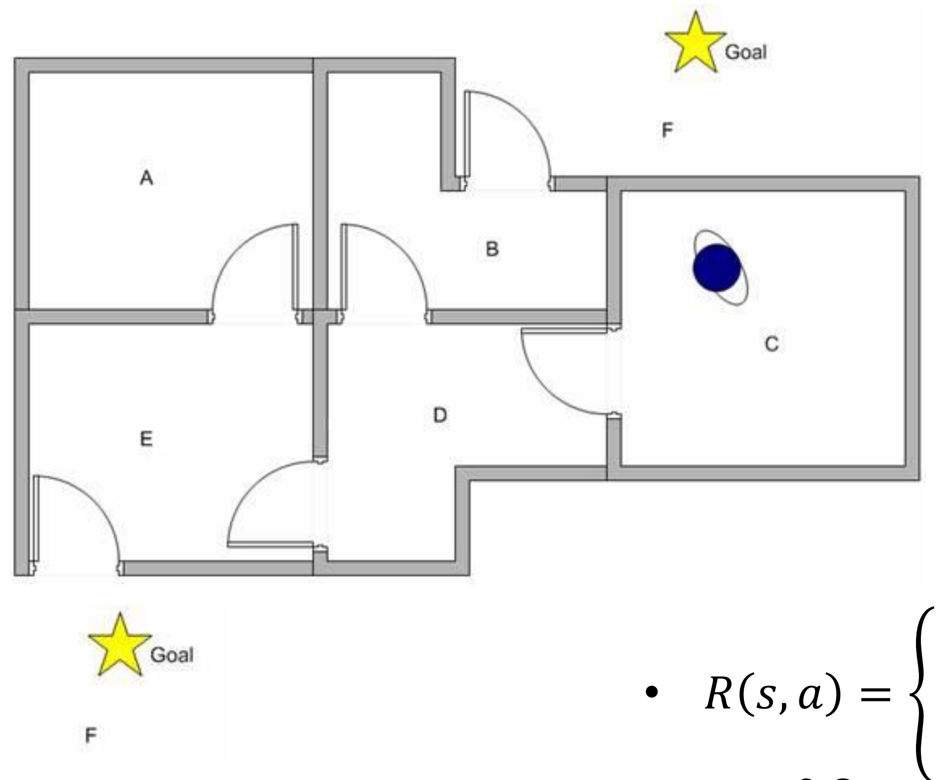
Q-Learning: Off-Policy TD Control



Take action $a_{t+1} = a^1$ given $s_{t+1} = s^1$ and observe r_{t+2} and $s_{t+2} = s^2$

Q-learning Step-by-Step Example

- The agent can pass one room to another but has no knowledge of the building
- That is, it does not know which sequence of doors the agent must pass to go outside the building
- Assume the agent is now in room C, and would like to reach outside the building (state F)

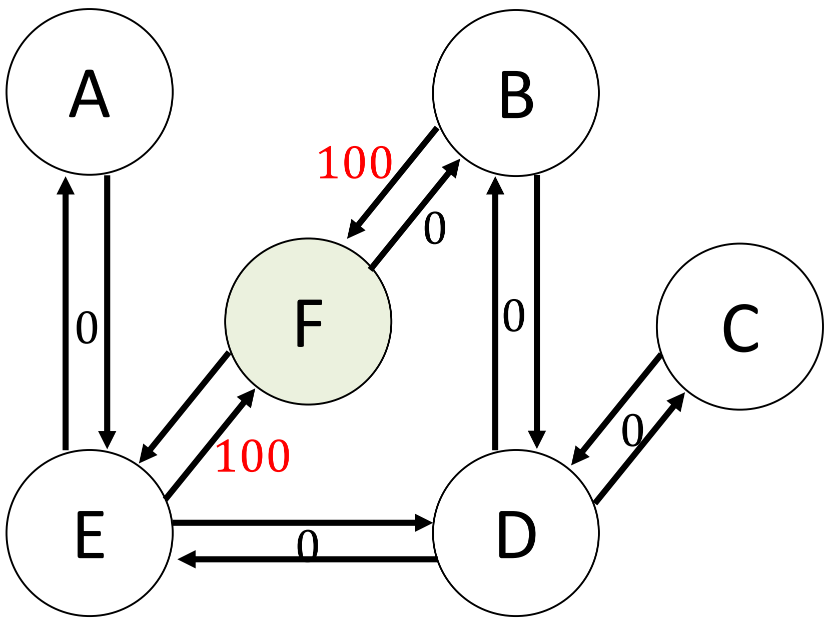


$MDP = \{\mathcal{S}, \mathcal{A}, T, R, \gamma\}$

- $s \in \mathcal{S} = \{A, B, C, D, E, F\}$
- $a \in \mathcal{A} = \{A, B, C, D, E, F\}$
  e.g., $\mathcal{A}(s = D) = \{B, C, E\}$
- $T(s, a) = \begin{cases} 1, \text{if move is allowed} \\ 0, \text{if move is not allowed} \end{cases}$
  e.g., $T(C, D) = 1$
- $R(s, a) = \begin{cases} 0 & \text{if move to } a \text{ is allowed and } a \neq F \\ 100 & \text{if move to } a \text{ is allowed and } a = F \end{cases}$
- $\gamma = 0.8$

Example from: http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/Q-Learning-Example.htm

Q-learning Step-by-Step Example



$R(s,a) =$

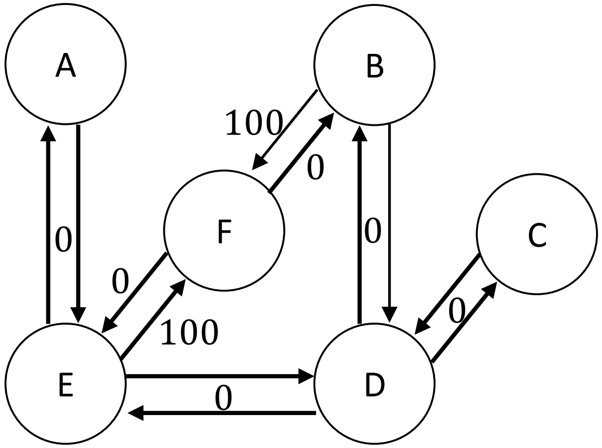
s \ a	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

Q Learning update rule:

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

Q-learning Step-by-Step Example

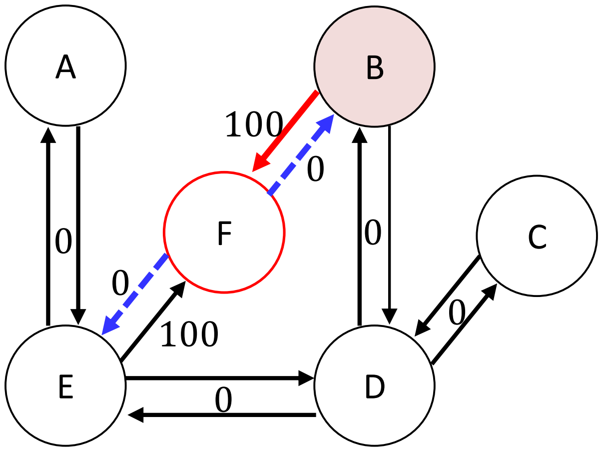
Q Learning update rule: $Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma \max_{a} Q(s',a) - Q(s,a) \right)$



$$Q(s,a) = \begin{array}{c c} & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-learning Step-by-Step Example

Q Learning update rule: $Q(s,a) \leftarrow Q(s,a) + \eta\left(r + \gamma \max_a Q(s',a) - Q(s,a)\right)$



$$Q(s,a) = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

1. Assume the initial state is B and take action F randomly (stochastic policy):

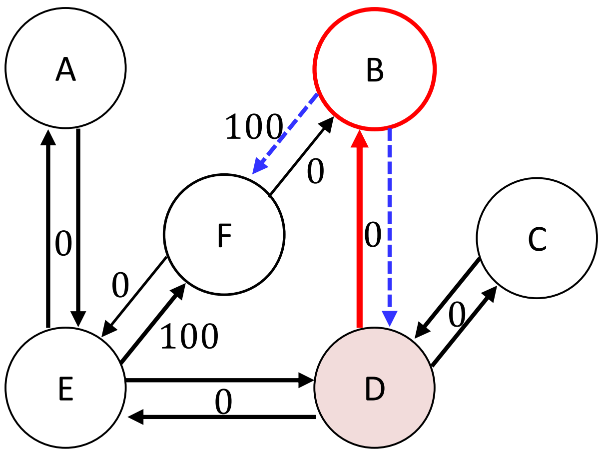
$$Q(B,F) \leftarrow Q(B,F) + 0.5\left(R(B,F) + 0.8 \max_a\{Q(F,B), Q(F,E)\} - Q(B,F)\right)$$

$$Q(B,F) \leftarrow 0 + 0.5(100 + 0.8 \times 0 - 0) = 50$$

2. Because the state F is the final state, the episode is over

Q-learning Step-by-Step Example

Q Learning update rule: $Q(s,a) \leftarrow Q(s,a) + \eta\left(r + \gamma \max_{a} Q(s',a) - Q(s,a)\right)$



$$Q(s,a) = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

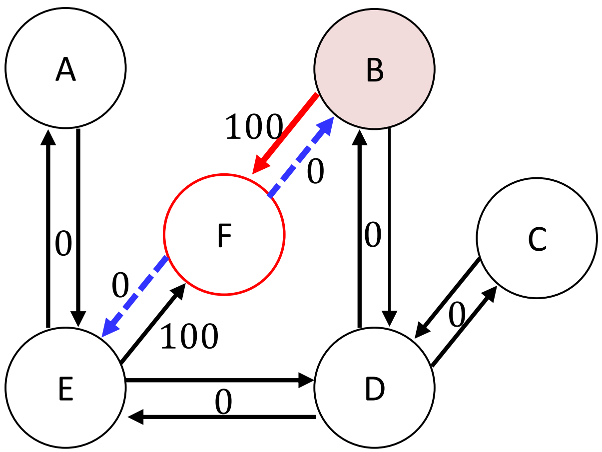
1. Assume the initial state is D and take action B randomly (stochastic policy):

$$Q(D,B) \leftarrow Q(D,B) + 0.5\left(R(D,B) + 0.8 \max_{a}\{\overset{50}{Q(B,F)}, \overset{0}{Q(B,D)}\} - Q(D,B)\right)$$

$$Q(D,B) \leftarrow 0 + 0.5(0 + 0.8 \times 50 - 0) = 20$$

Q-learning Step-by-Step Example

Q Learning update rule: $Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma \max_a Q(s',a) - Q(s,a) \right)$



$$Q(s,a) = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 75 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

1. Assume the initial state is D and take action B randomly (stochastic policy):

$$Q(D,B) \leftarrow Q(D,B) + 0.5 \left(R(D,B) + 0.8 \max_a \{ \overset{50}{Q(B,F)}, \overset{0}{Q(B,D)} \} - Q(D,B) \right)$$

$$Q(D,B) \leftarrow 0 + 0.5(0 + 0.8 \times 50 - 0) = 20$$

2. The next state is B and take an action of F randomly):

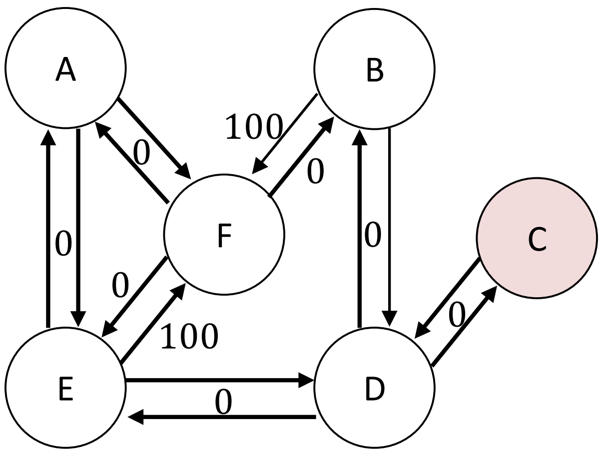
$$Q(B,F) \leftarrow Q(B,F) + 0.5 \left(R(B,F) + 0.8 \max_a \{ \overset{0}{Q(F,B)}, \overset{0}{Q(F,E)} \} - Q(B,F) \right)$$

$$Q(B,F) \leftarrow 50 + 0.5(100 + 0.8 \times 0 - 50) = 75$$

3. Because the state F is the final state, the episode is over

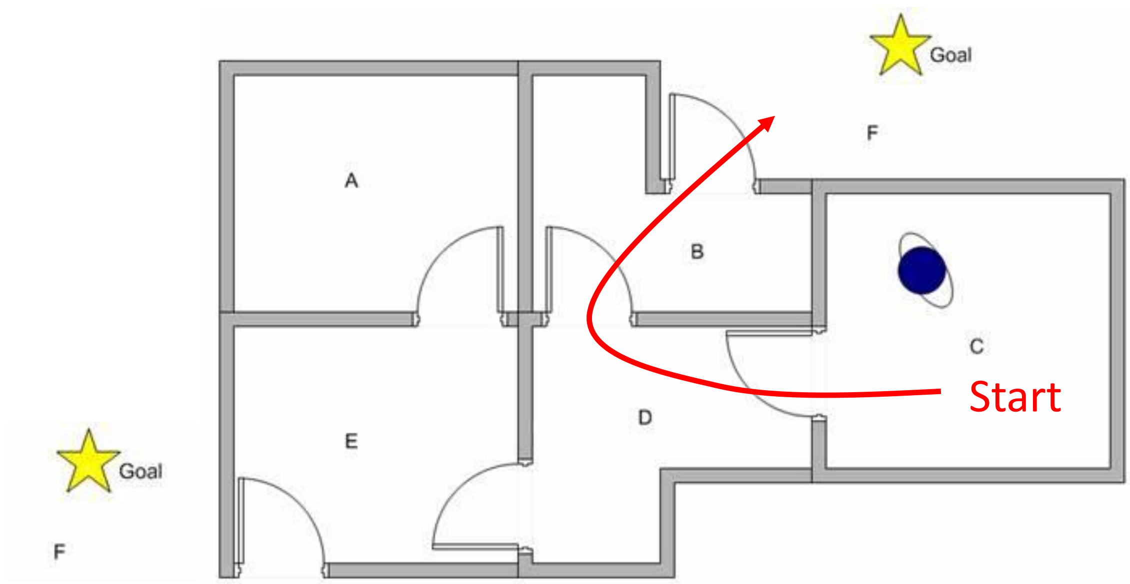
Q-learning Step-by-Step Example

Q Learning update rule: $Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma \max_{a} Q(s',a) - Q(s,a) \right)$

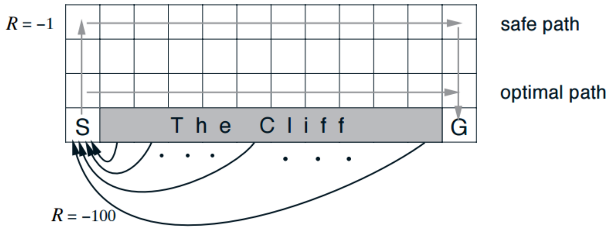


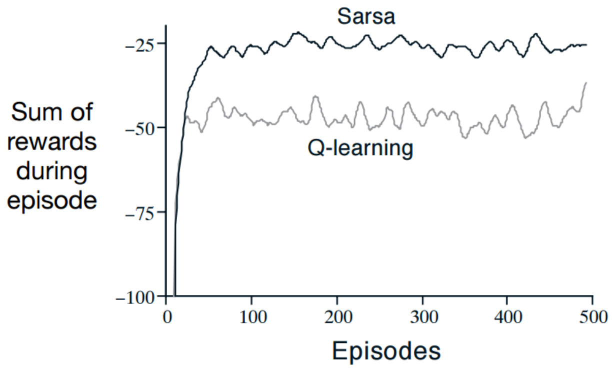
$$Q^*(s,a) = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

After convergence



Example 6.6 Cliff Walking





The path away from the cliff

- Take longer
- A wrong action will not hurt you as much

walk near the cliff

- Faster
- a wrong action deterministically causes falling off the cliff.

- Sarsa learns about a policy that sometimes takes optimal actions (as estimated) and sometimes explores other actions (Estimation policy = Behavioral policy)
  - Sarsa will learn to be careful in an environment where exploration is costly

- Q-learning learns about the policy that doesn't explore and only takes optimal (as estimated) actions
  - The optimal policy does not capture the risk of exploratory action

The cliff example shows why such a non-optimal policy could be sometimes very useful

Why Q-learning is considered as Off-Policy method

- Q-learning updates are done regardless to the actual action chosen for next state (behavioral policy)
- That is, for estimation, it just assumes that we are always choosing the argmax one

$$a_{t+1} = \underset{a}{\operatorname{argmax}}\, Q(s_{t+1} = s^1, a)$$

Behavioral Policy π_B

Estimation Policy π_E

$$a'_B = \pi_B(s) \neq a'_E = \underset{a'}{\operatorname{argmax}}\, Q(s', a')$$

Used to generated data

Used to estimate $Q(s, a)$

Take action a'_B and transit to the next state

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

$$Q(s,a) \leftarrow Q(s,a) + \eta\big(r + \gamma Q(s',a'_E) - Q_\pi(s,a)\big)$$

Why Q-learning is considered as Off-Policy method

Change to the stochastic policy

$$\pi(s) \qquad \pi(s')$$

$$(s, a, r, s', a')$$

$$Q(s,a) \leftarrow Q(s,a) + \eta\big(r + \gamma Q(s',a') - Q_\pi(s,a)\big)$$

Change to the stochastic policy

$$\pi_b(s) \qquad \pi_b(s)$$

$$(s, a, r, s') \rightarrow a$$

$$a'_E = \underset{a'}{\operatorname{argmax}}\, Q(s', a')$$

$$Q(s,a) \leftarrow Q(s,a) + \eta\big(r + \gamma Q(s',a'_E) - Q_\pi(s,a)\big)$$

Greedy policy (deterministic)

Consider the extreme case:

Suppose you were to take a completely random action on each step (if epsilon greedy exploration is used, set epsilon to 1).

- Sarsa is literally learning the value of the random policy while acting randomly

- Q-learning is learning the value of the optimal policy, but is *acting* randomly.

Off-Policy TD Control (Q-learning)

- Based on a single transition, i.e., state-action pair
- Online setting: Learn and take action continuously
- Exploration and Exploitation : Nee to learn and optimize at the same time
- Monte Carlo vs. Bootstrapping

What is the next?