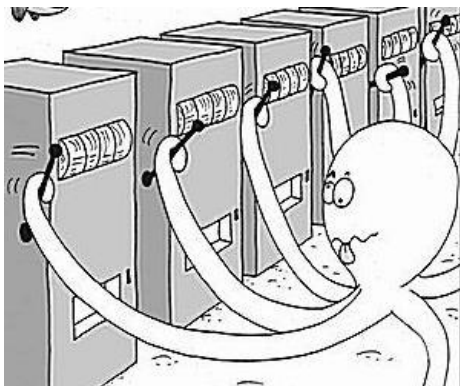


# **Lecture 21**

## **Introduction to Reinforcement Learning**

- **Monte Carlo method**
  - ✓ Policy Evaluation
  - ✓ Policy Improvement
  - ✓ Policy Iteration (Monte Carlo control)
- **Temporal Different method**
  - ✓ SARSA
  - ✓ Q-learning
- **Policy Gradient Method**

## An $n$ -Armed Bandit Problem



- There are  $n$  machine
- Each machine  $i$  returns a reward  $r \sim P(\theta_i)$  :  $\theta_i$  is unknown
- $a_t \in \{1, \dots, n\}$  : the choice of machine at time  $t$
- $r_t$  : the reward at time  $t$
- Policy  $\pi$  maps all the history to new action:

$$\pi: [(a_1, r_1), (a_2, r_2), \dots, (a_{t-1}, r_{t-1})] \rightarrow a_t$$

Find the optimal policy  $\pi^*$  that maximizes  $E[\sum_{t=1}^T r_t]$  or  $E[r_T]$

Acquiring new information  
**(exploration)**

trade-off

capitalizing on the information  
available so far  
**(exploitation)**

**Internet add** : Advertising showing strategy for users

**Finance** : Portfolio optimization under unknown return profiles (risk vs mean profit)

**Health care** : Choosing the best treatment among alternatives

**Internet shopping** : Choosing the optimum price (sales v.s. profits)

**Experiment design** : Sequential experimental design (or sequential simulation parameter)

## An $n$ -Armed Bandit Problem

### The value of the action $Q^*(a)$ :

The true value of action  $a$  is defined as the mean reward received when that action is selected

$$Q^*(a) = E[r]$$

At any time there will be at least one action whose **value of the action** is the largest

- **Select greedy action (Exploiting)**: the action with the largest value of action
- **Select non-greedy action (Exploring)**: the action enabling you to improve your estimate of the non-greedy actions' value

### Exploiting

Go to a company to make money  
right after undergraduate



### Exploring

Go to a graduate school to explore my  
intellectual capability?



How to balance between exploitation and exploration ?

## Action-Value Methods

- Action-value method estimates the value of an action and use this to select the action
- Natural way to estimate the *action-value* is by averaging the rewards actually received when the action was selected
- If at time  $t$ , **action  $a$  has been chosen  $k_a$  times prior to  $t$** , yielding rewards,  $r_1, r_2, \dots, r_{k_a}$ , the estimated action value  $Q_t(a)$  for  $a$  at time  $t$  is defined as

$$Q_t(a) = \frac{r_1 + r_2 + \dots + r_{k_a}}{k_a}$$

**If  $k_a = 0$ ,  $Q_t(a) = 0$**   
**If  $k_a = \infty$ ,  $Q_t(a) \rightarrow Q^*(a)$**

- Greedy action selection rule is

$$a_t = \operatorname{argmax}_a Q_t(a)$$

- ✓ Always exploits current knowledge to maximize immediate reward, that is it spends no time at all sampling apparently inferior actions to see if they might really be better

## Action-Value Methods

- If at  $t$ th play, action  $a$  has been chosen  $k_a$  times prior to  $t$ , yielding rewards,  $r_1, r_2, \dots, r_{k_a}$ , the estimated action value for  $a$  is defined as

$$Q_t(a) = \frac{r_1 + r_2 + \dots + r_{k_a}}{k_a}$$

If  $k_a = 0$ ,  $Q_t(a) = 0$   
If  $k_a = \infty$ ,  $Q_t(a) \rightarrow Q^*(a)$

### *greedy* action selection rule

$$a_t = \underset{a}{\operatorname{argmax}} Q_t(a)$$

- ✓ Always exploits current knowledge to maximize immediate reward
- ✓ spends no time at all sampling apparently inferior actions to see if they might really be better

### $\epsilon$ - *greedy* action selection rule

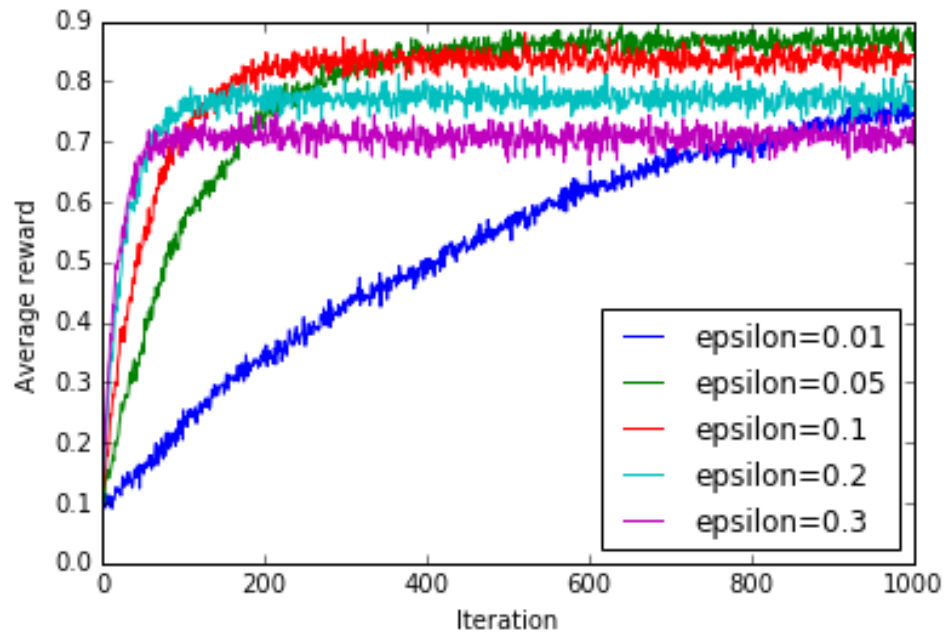
$$\pi(a) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)| & \text{If } a = a^* \\ \epsilon/|A(s)| & \text{If } a \neq a^* \end{cases}$$

$$\left(1 - \epsilon + \frac{\epsilon}{|A(s)|}\right) \times 1 + \frac{\epsilon}{|A(s)|} (|A(s)| - 1) = 1$$

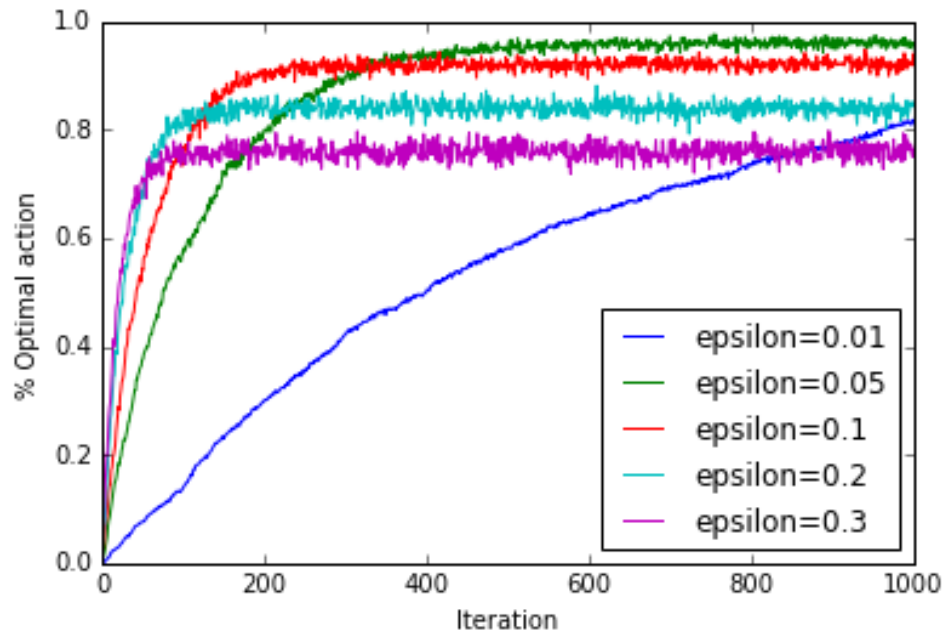
- ✓ In the limit as the number of plays increases, every  $a$  will be sampled an infinite number of times, guaranteeing  $k_a \rightarrow \infty$  thus  $Q_t(a) \rightarrow Q^*(a)$
- ✓ The probability of selecting the optimum action converges to greater than  $1 - \epsilon$

## Action-Value Methods

Average reward  $E[r_t]$



% of selecting  $a^*$



## Relationship with model based approach

		Only Exploitation	Exploration vs Exploitation
Optimum action	Single state	Model is known  Optimization	Model is unknown  Bandit
			Contextual Bandit
Optimum policy	Multiple state	Dynamic Programing	Reinforcement learning

- We going to learn Dynamic Programming approach first, and move to Reinforcement learning





### Markov Decision Process (Offline)

- Have mental model of how the world works
- Find policy to collect the maximum rewards

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



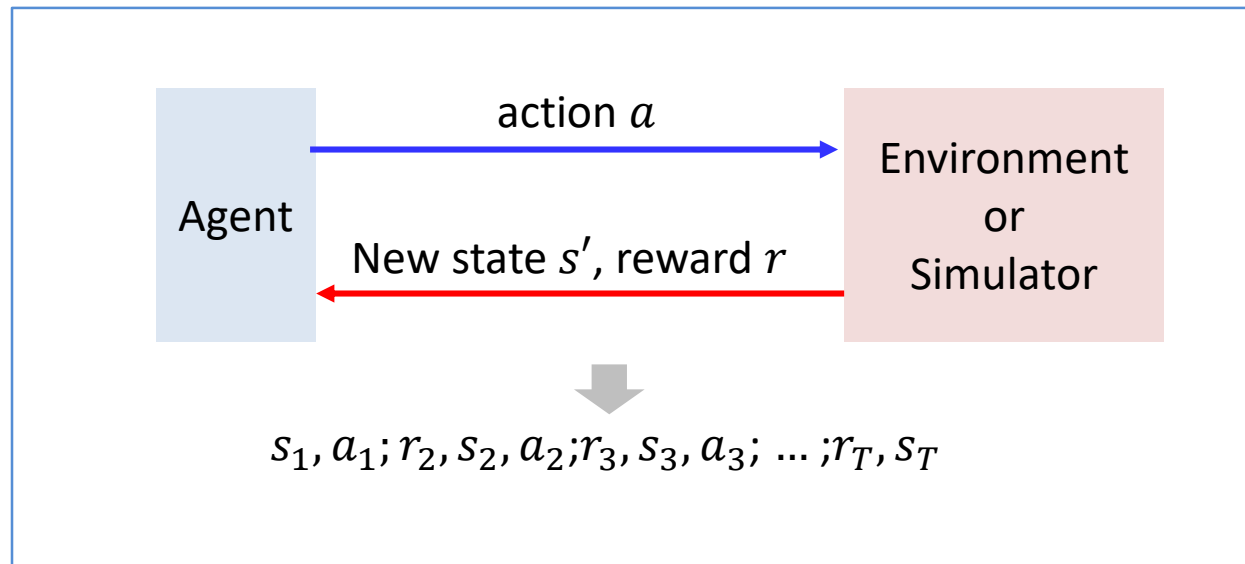
### Reinforcement Learning (Offline & Online)

- Don't know how the world works
- Perform a sequence of actions in the world to maximize the rewards

$$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T \rightarrow Q^*(s, a) \rightarrow \pi^*(s)$$

- Reinforcement learning is really the way humans work:
  - we go through life, taking various actions, getting feedback.
  - We get rewarded for doing well and learn along the way.

## Reinforcement Learning Template



### Template for Reinforcement Learning

For  $t = 1, 2, 3, \dots$

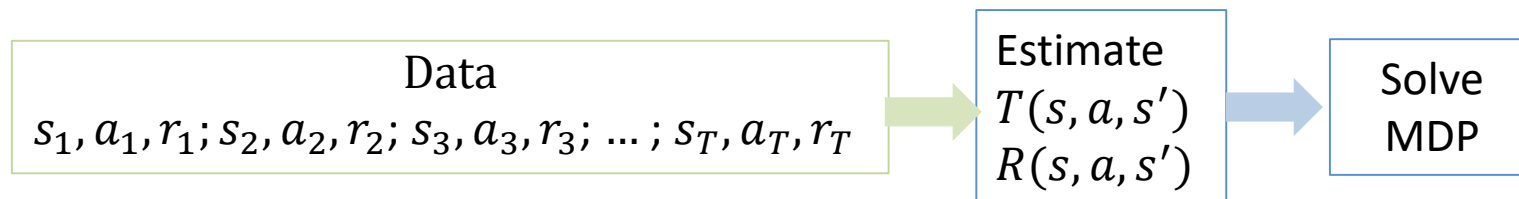
Choose action  $a_t = \pi(s_t)$  (how?) : **Decision making**

Receive reward  $r_{t+1}$  and observe new state  $s_{t+1}$  (Environment)

Update parameters associated with  $V(t)$ ,  $Q(s, a)$  (how?) : **Learning**

## Road Map

- Model-Based Reinforcement learning



- Model-FREE Reinforcement learning

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

Monte Carlo method

Temporal Difference methods

How to  
explore ?

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

- Episodic based

- Single-data-point based

## Monte Carlo Reinforcement Learning

### Key Idea : model-based learning

- Formulate the MDP using the estimated  $T(s, a, s')$  and  $R(s, a, s')$
- Solve the MDP using various solution methods, i.e., DP approach

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

- Data following policy  $\pi$

$$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$$

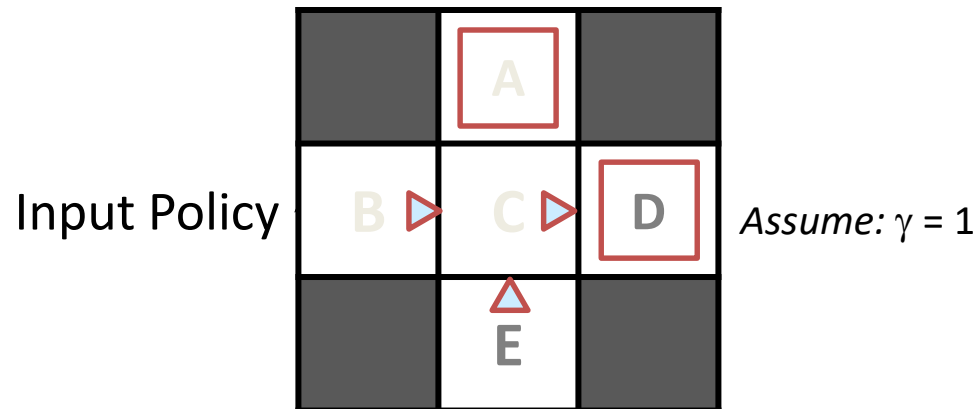
- Transition model

$$\hat{T}(s, a, s') = \frac{\text{\#times } (s, a, s') \text{ occurs}}{\text{\#times } (s, a) \text{ occurs}}$$

- Reward model

$$\hat{R}(s, a, s') = \text{average of } r \text{ in } (s, a, r, s')$$

## Example



- Observed Episodes (Training)

### Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

### Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

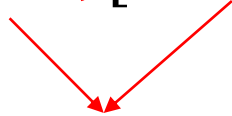
- Learned Model

$$T(s, a, s')$$

$T(B, \text{east}, C) = 1.00$   
 $T(C, \text{east}, D) = 0.75$   
 $T(C, \text{east}, A) = 0.25$

$$R(s, a, s')$$

$R(B, \text{east}, C) = -1$   
 $R(C, \text{east}, D) = -1$   
 $R(D, \text{exit}, x) = +10$

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


Estimation

Why not estimating  $Q^*(s, a)$  directly  
instead of separately estimating  $\hat{T}(s, a, s')$  and  $\hat{R}(s, a, s')$  ?

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

## Model-Free Monte Carlo Based Methods

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

Monte Carlo method

Temporal Difference methods

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

• Episodic based

• Single-data-point based



## Model-Free Monte Carlo Based Methods

- Monte Carlo methods require only experience-sample sequences of states, actions, and rewards from on-line or simulated interaction with an environment
- Monte Carlo methods are ways of solving the reinforcement learning problem based on **averaging sample returns (Monte Carlo)**

$$R(s, a) = \text{average of } r \text{ in } (s, a, r)$$

- Monte Carlo methods are incremental in an **episode-by-episode sense**, but not in a step-by-step sense  
→ after a final state has reached, reward appears, e.g., Go

### Dynamic Programming

- Policy Evaluation
- Policy Iteration
- Policy Improvement

Key ideas

### Monte Carlo Methods

- Policy Evaluation
- Policy Iteration
- Policy Improvement

### Key Idea : Monte Carlo Policy Evaluation

- Learn the state-value function  $V^\pi(s)$  for a given policy  $\pi$
- The value of state is the expected utility - **expected accumulative future reward** starting from  $s$  and following the policy  $\pi$

Data (following policy  $\pi$ ):

$$s_t \in \mathcal{S} = \{s^1, s^2, s^3\}, a_t \in \mathcal{A} = \{a^1, a^2, a^3\}$$

Episode 1:  $s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$

Episode 2:  $s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$

Episode 3:  $s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$

$\vdots$

Utility  $u_t$ :

$$u_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$u_t = r_t + r_{t+2} + \dots + r_T$$

Estimate  $V^\pi(s)$  or  $Q^\pi(s, a)$  :

$$V^\pi(s) = \text{average of } u_t \text{ where } s_t = s \text{ and following } \pi$$

Because the value being computed is dependent on the policy used to generate the data, we call this an **on-policy** algorithm.

### Key Idea : Monte Carlo Policy Evaluation

- Learn the state-value function  $V^\pi(s)$  for a given policy  $\pi$
- The value of state is the expected utility - **expected accumulative future reward** starting from  $s$  and following the policy  $\pi$

$$s_t \in \mathcal{S} = \{s^1, s^2, s^3\}, a_t \in \mathcal{A} = \{a^1, a^2, a^3\}$$

(State, Action, Reward ) pairs generated by policy  $\pi$

Episode 1: ( $s^1, a^2, 1$ ); ( $s^3, a^1, 5$ ); ( $s^2, a^3, 3$ ), ( $s^1, a^3, 10$ ), ( $s^2, a^2, 2$ )

Episode 2: ( $s^3, a^1, 5$ ); ( $s^2, a^2, 2$ ); ( $s^1, a^2, 1$ ); ( $s^2, a^3, 3$ ), ( $s^1, a^3, 10$ )

Episode 3: ( $s^2, a^3, 3$ ); ( $s^1, a^2, 1$ ); ( $s^3, a^1, 5$ ); ( $s^1, a^3, 10$ ), ( $s^2, a^2, 2$ )

( $\gamma = 1$ )

Episode 1:  $V^\pi(s^1) = 1 + 5 + 3 + 10 + 2 = 21$

Episode 2:  $V^\pi(s^1) = 1 + 3 + 10 = 14$

Episode 3:  $V^\pi(s^1) = 1 + 5 + 10 + 2 = 19$

First visit to  $s$

$V^\pi(s^1) =$  average of accumulated reward over all episodes

$$= \frac{21+14+19}{3} = 14.6$$

### Key Idea : Monte Carlo Policy Evaluation

- Learn the state-value function  $V^\pi(s)$  for a given policy  $\pi$
- The value of state is the expected utility - **expected accumulative future reward** starting from  $s$  and following the policy  $\pi$

$$s_t \in \mathcal{S} = \{s^1, s^2, s^3\}, a_t \in \mathcal{A} = \{a^1, a^2, a^3\}$$

(State, Action, Reward ) pairs generated by policy  $\pi$

Episode 1: ( $s^1, a^2, 1$ ); ( $s^3, a^1, 5$ ); ( $s^2, a^3, 3$ ), ( $s^1, a^3, 10$ ), ( $s^2, a^2, 2$ )

Episode 2: ( $s^3, a^1, 5$ ); ( $s^2, a^2, 2$ ); ( $s^1, a^2, 1$ ); ( $s^2, a^3, 3$ ), ( $s^1, a^3, 10$ )

Episode 3: ( $s^2, a^3, 3$ ); ( $s^1, a^2, 1$ ); ( $s^3, a^1, 5$ ); ( $s^1, a^3, 10$ ), ( $s^2, a^2, 2$ )

( $\gamma = 1$ )

Episode 1:  $V^\pi(s^1) = 1 + 5 + 3 + 10 + 2 = 21$ ;  $V^\pi(s^1) = 10 + 2 = 12$

Episode 2:  $V^\pi(s^1) = 1 + 3 + 10 = 14$ ;  $V^\pi(s^1) = 10 = 10$

Episode 3:  $V^\pi(s^1) = 1 + 5 + 10 + 2 = 19$ ;  $V^\pi(s^1) = 10 + 2 = 12$

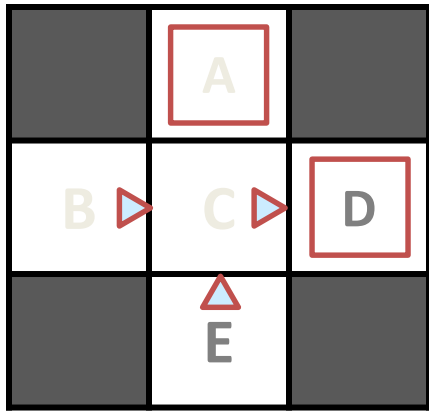
Every visit to  $s$

$V^\pi(s^1) =$  average of accumulated reward over all episodes

$$= \frac{21+12+14+10+19+12}{6} = 15$$

## Example

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

Output Values

	-10	
+8	+4	+10
	-2	

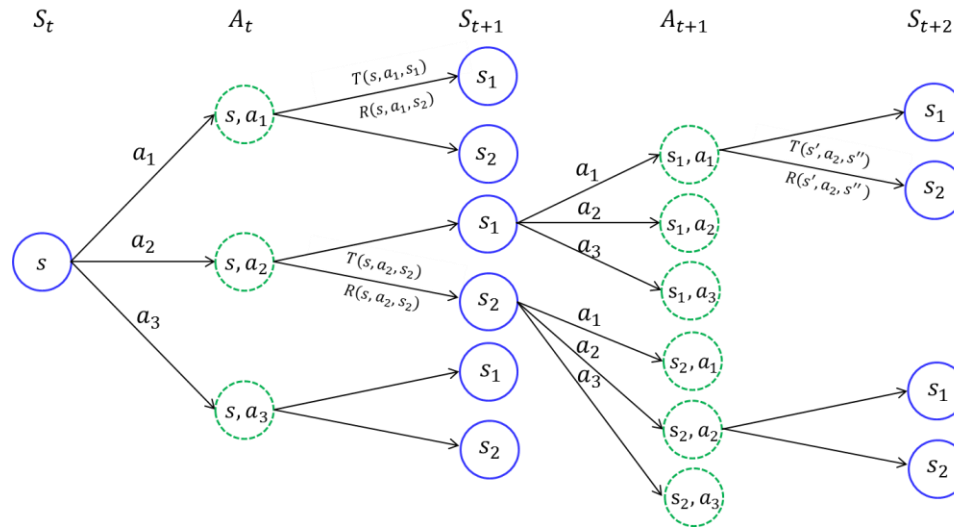
### Example

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of  $T$ ,  $R$
  - It eventually computes the correct average values, using just sample transitions
- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

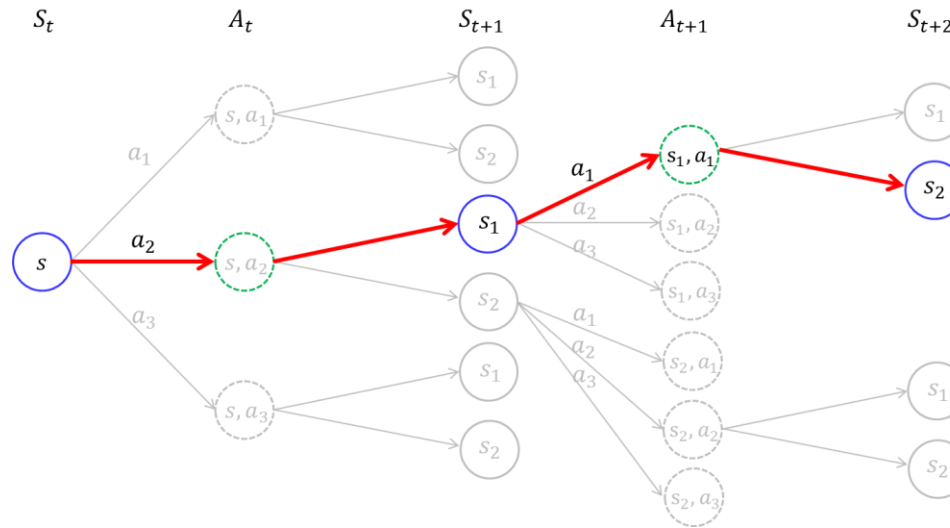
### Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

# Monte Carlo Policy Evaluation



## Monte Carlo Policy Evaluation



- Estimates for each state are independent
  - ✓ The estimate for one state does not build upon the estimate of any other state (No bootstrap)
- The computational expense of estimating the value of a single state is independent of the number of states
  - ✓ Attractive when one requires the value of only a subset of the states



## Monte Carlo Estimation of Action Values

- Without a model, state value function  $V^\pi(s)$  is not enough
  - ✓ We need  $T(s, a, s')$  and  $R(s, a, s')$  to compute the optimum policy:

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

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

- One of primary goals for Monte Carlo method is to estimate  $Q^*(s, a)$

### Key Idea : Monte Carlo Policy Evaluation

- Learn the action-value function  $Q^\pi(s, a)$  for a given policy  $\pi$
- The value of action-state is the expected utility - **expected accumulative future reward** starting from  $s$  and following the policy  $\pi$

Data (following policy  $\pi$ ):

$$s_t \in \mathcal{S} = \{s^1, s^2, s^3\}, a_t \in \mathcal{A} = \{a^1, a^2, a^3\}$$

Episode 1:  $s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$

Episode 2:  $s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$

Episode 3:  $s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$

$\vdots$

Utility  $u_t$ :

$$u_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$u_t = r_t + r_{t+2} + \dots + r_T$$

Estimate  $Q^\pi(s, a)$  :

$$Q^\pi(s, a) = \text{average of } u_t \text{ where } s_t = s, a_t = a \text{ and following } \pi$$

### Key Idea : Monte Carlo Policy Evaluation

- Learn the state-value function  $Q^\pi(s, a)$  for a given policy  $\pi$
- The value of action-state is the expected utility - **expected accumulative future reward** starting from  $s$  and following the policy  $\pi$

$$s_t \in \mathcal{S} = \{s^1, s^2, s^3\}, a_t \in \mathcal{A} = \{a^1, a^2, a^3\}$$

(State, Action, Reward ) pairs generated by policy  $\pi$

Episode 1: ( $s^1, a^2, 1$ ); ( $s^3, a^1, 5$ ); ( $s^2, a^3, 3$ ), ( $s^1, a^3, 10$ ), ( $s^2, a^2, 2$ )

Episode 2: ( $s^1, a^1, 5$ ); ( $s^2, a^2, 2$ ); ( $s^1, a^2, 1$ ); ( $s^2, a^3, 3$ ), ( $s^1, a^3, 10$ )

Episode 3: ( $s^2, a^3, 3$ ); ( $s^1, a^2, 1$ ); ( $s^3, a^1, 5$ ); ( $s^1, a^3, 10$ ), ( $s^2, a^2, 2$ )

( $\gamma = 1$ )

Episode 1:  $Q^\pi(s^1, a^2) = 1 + 5 + 3 + 10 + 2 = 21$

Episode 2:  $Q^\pi(s^1, a^2) = 1 + 3 + 10 = 14$

Episode 3:  $Q^\pi(s^1, a^2) = 1 + 5 + 10 + 2 = 19$

First visit to  $s$

$Q^\pi(s^1, a^2) =$  average of accumulated reward over all episodes

$$= \frac{21+14+19}{3} = 14.6$$

## Incremental Formulation

$s_1, a_1, r_1; s_2 = s, a_2 = a, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$   
 $s_1 = s, a_1 = a, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$   
 $s_1, a_1, r_1; s_2, a_2, r_2; s_3 = s, a_3 = a, r_3; \dots; s_T, a_T, r_T$   
 $\vdots$   
 $s_1, a_1, r_1; s_2 = s, a_2 = a, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T$

Episode 1

Episode 2

Episode 3

Episode  $\infty$

For each episode,

We can get state and action pair

$(s, a)$  and the following accumulated  
reward (utility)  $u$ :  $(s, a) \rightarrow u$

### Incremental formulation (convex combination):

On each  $(s, a, u)$  for a single episode:

$$Q^\pi(s, a) \leftarrow (1 - \eta) \underbrace{Q^\pi(s, a)}_{\text{Current estimate}} + \eta \underbrace{u}_{\text{New data}}$$

where  $\eta = \frac{1}{1 + (\text{\#updates to } (s, a))}$

### Stochastic gradient form :

On each  $(s, a, u)$  for a single episode:

$$Q^\pi(s, a) \leftarrow Q^\pi(s, a) - \eta(Q^\pi(s, a) - u)$$

The incremental formulations can be thought of as a way to solve the following least square estimation in an online manner

$$\min_{Q^\pi} \sum_{(s, a, u)} (Q^\pi(s, a) - u)^2$$

## Issue of computing $Q^\pi(s, a)$

- If  $\pi$  is a deterministic policy, many relevant state –action pairs may never be visited
  - ✓ then in following  $\pi$ , one will observe returns only for one of the actions from each state, i.e., we won't even see  $(s, a)$  if a  $a \neq \pi(s)$
  - ✓ To improve the policy  $\pi$ , we need to **compare** the state-action values of all the possible actions,  $Q^\pi(s, a_1), Q^\pi(s, a_2), \dots$
- To estimate  $Q^\pi(s, a)$  reliably, we need a large number of episodes
  - ✓ *Infinite* number of episodes are required for accurate estimations

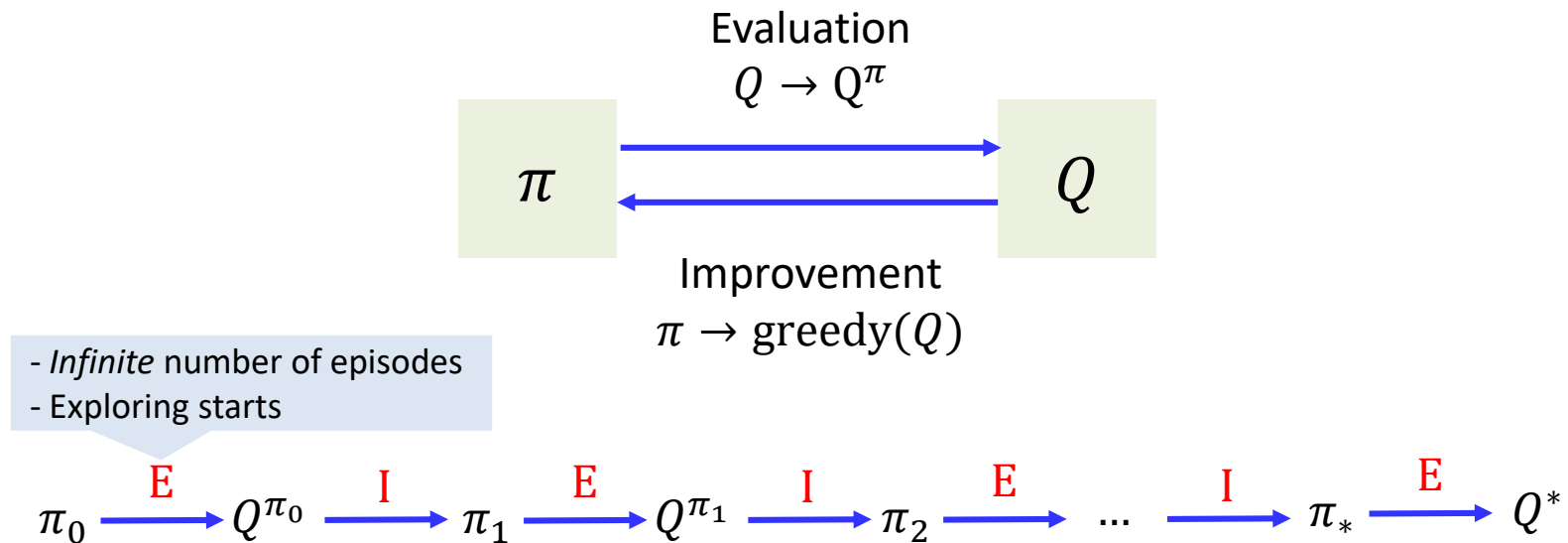
### How to resolve?

- **Exploring starts + infinite number of episodes**
  - ✓ Start from an arbitrary state-action pair  $(s, a)$  with a non-zero probability
  - ✓ Guarantees **that all state-action pair will be visited an infinite number of times in the limit of an infinite number of episodes**
  - ✓ Cannot be used when learning directly from experience
- **Stochastic Policy + infinite number of episodes**
  - ✓ Policy with **a nonzero probability of selecting all actions**
  - ✓ Guarantees that all state-action pair will be visited an infinite number of times in the limit of an infinite number of episodes

We need an efficient exploration strategy!

### Key Idea : Monte Carlo Control

- Idea of generalized policy iteration (GPI)
- Monte Carlo Policy Evaluation + Policy improvement



- **Policy Evaluation**
  - ✓ The value function is repeatedly altered to more closely approximate the value function for the current policy  $\pi$
- **Policy Improvement**
  - ✓ The policy is repeatedly improved with respect to the current action value function  $Q$

# Monte Carlo Control

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} Q^*$$

## E Policy evaluation:

- The value function is repeatedly altered to more closely approximate the value function for the current policy  $\pi$
  - Infinite number of episodes under policy  $\pi$
  - The episodes are generated with exploring starts
- } Converge to exact  $Q^\pi(s, a)$

exploring starts	$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T \rightarrow u$	Episode 1
	$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T \rightarrow u$	Episode 2
	$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T \rightarrow u$	Episode 3
	$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots; s_T, a_T, r_T \rightarrow u$	Episode $\infty$

Average expected reward

## I Policy Improvement:

- Policy improvement can be done by constructing each  $\pi_{k+1}$  as the greedy policy with respect to  $Q^{\pi_k}$

$$\pi_{k+1}(s) = \operatorname{argmax}_a Q^{\pi_k}(s, a)$$

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \operatorname{argmax}_a Q^{\pi_k}(s, a)) \\ &= \max_a Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \\ &= V^{\pi_k}(s) \end{aligned}$$

$$V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s) \text{ for all } s$$

(Policy improvement theorem)

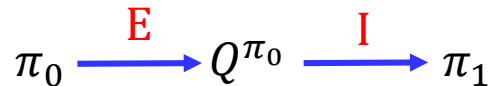
Overall process converges to an optimal policy and the optimal value function

## Make Monte Carlo Control Practical

### Assumptions:

- *Infinite* number of episodes
- Exploring starts

➡ **Relax** these two assumptions



- **Relaxing “Infinite number of episodes”**
  - ✓ Policy improvement with an early stop (Generalized policy improvement concept)
  - ✓ The “in place” version of value iteration
- Relaxing “exploring starts”
  - ✓ Discussed later



### Algorithm : Monte Carlo ES Control

Initialize, for all  $s \in S, a \in A(s)$

$Q(s, a) \leftarrow$  arbitrary

$\pi(s) \leftarrow$  arbitrary

$U(s, a) \leftarrow$  empty list

Repeat forever:

(a) Generate *an episode* using *exploring starts* and  $\pi$

(b) For each pair  $(s, a)$  appearing in the episode:

$U \leftarrow$  utility following the first occurrence of  $s, a$

Append  $U$  to  $U(s, a)$

$Q(s, a) \leftarrow \text{average}(U(s, a))$

(c) For each  $s$  in the episode:

$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} Q(s, a)$

} Policy evaluation

} Policy improvement

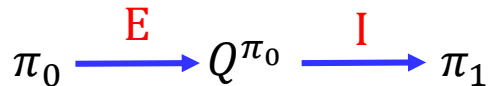
In an single episode, both Policy evaluation and policy improvement proceeds together

## Make Monte Carlo Control Practical

### Assumptions:

- *Infinite* number of episodes
- Exploring starts

➡ **Relax** these two assumptions



- Relaxing “Infinite number of episodes”
  - ✓ Policy improvement with an early stop (Generalized policy improvement concept)
  - ✓ The “in place” version of value iteration
- **Relaxing “exploring starts”**
  - ✓ The only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them
    - ❖ On-policy methods
    - ❖ Off-policy methods

## On policy algorithm



- On-policy methods attempt to evaluate or improve the policy that is used to make decisions (generate data)
  - ✓ Policy is soft:  $\pi(s, a) > 0$  for all  $s \in S$  and  $a \in A(s)$

## Off policy algorithm



- Behavioral policy is used to generate behavior
- Estimation policy is evaluated and improved
  - ✓ Estimation policy may be deterministic, while the behavior policy can continue to sample all possible actions

## On-policy Monte Carlo Control

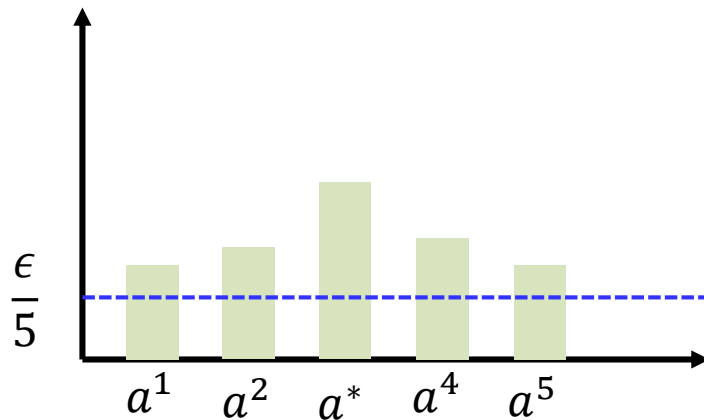
### Key Idea : On-policy methods: Soft policies

- attempt to evaluate or improve the policy that is used to make decisions
- $\pi(s, a) > 0$  for all  $s \in S$  and  $a \in A(s)$

#### $\epsilon$ – soft policy

$$\pi(s, a) \geq \frac{\epsilon}{|A(s)|} \text{ for all } a$$

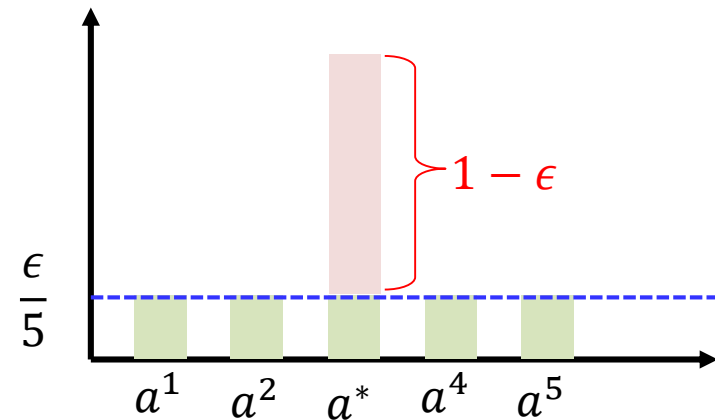
$$\sum_a \pi(s, a) = 1$$



#### $\epsilon$ – greedy policy

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{If } a = a^* \\ \frac{\epsilon}{|A(s)|} & \text{If } a \neq a^* \end{cases}$$

$$\text{Total probability} = \left(1 - \epsilon + \frac{\epsilon}{|A(s)|}\right) 1 + \frac{\epsilon}{|A(s)|} (|A(s)| - 1) = 1$$



## On-policy Monte Carlo Control

Any  $\epsilon$  – *greedy* policy  $\pi'$  respect to  $Q^\pi$  is an improvement over any  $\epsilon$  – *soft* policy  $\pi$

Let  $\pi'$  be the  $\epsilon$  – *greedy* policy,

$$\begin{aligned} Q^\pi(s, \pi'(s, a)) &= \sum_a \pi'(s, a) Q^\pi(s, a) \\ &= \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) + (1 - \epsilon) \max_a Q^\pi(s, a) \\ &\geq \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) + (1 - \epsilon) \sum_a \frac{\pi(s, a) - \frac{\epsilon}{|A(s)|}}{1 - \epsilon} Q^\pi(s, a) \\ &= \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) - \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) + \sum_a \pi(s, a) Q^\pi(s, a) \\ &= V^\pi(s) \end{aligned}$$

### Recall Policy improvement Theorem

Policy improvement must give us a strictly better policy  $\pi'(s)$  than the older policy  $\pi(s)$  except when the original policy is already optimal  $\pi(s, a) = \pi^*(s, a)$

$$Q^\pi(s, \pi'(s, a)) \geq V^\pi(s) \rightarrow V^{\pi'}(s) \geq V^\pi(s)$$

Thus, by the policy improvement theorem,  $\pi' \geq \pi$  (i.e.,  $V^{\pi'}(s) \geq V^\pi(s)$  for all  $s \in S$ )

### Algorithm : $\epsilon$ – *soft* On-Policy Monte Carlo Control

Initialize, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$

$Q(s, a) \leftarrow$  arbitrary

$\pi \leftarrow$  an arbitrary  $\epsilon$  – *soft policy*

$U(s, a) \leftarrow$  empty list

Repeat forever:

(a) Generate *an episode* using  $\pi$

(b) For each pair  $(s, a)$  appearing in the episode:

$U \leftarrow$  utility following the first occurrence of  $s, a$

Append  $U$  to  $U(s, a)$

$Q(s, a) \leftarrow \text{average}(U(s, a))$

} Policy evaluation

(c) For each  $s$  in the episode:

$a^* \leftarrow \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} Q(s, a)$

For all  $a \in \mathcal{A}(s)$

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)| & \text{If } a = a^* \\ \epsilon/|A(s)| & \text{If } a \neq a^* \end{cases}$$

} Policy improvement

## Monte Carlo control algorithm

Black Jack Example

## Summary

- The Monte Carlo methods learn value functions and optimal policies from experience in the form of sample episodes
- Advantages are:
  - ✓ Can be used to learn optimal behavior directly from interaction with the environment with no models of environment dynamics
  - ✓ Can be used with simulation or sample models
  - ✓ It is efficient to focus Monte Carlo methods on a small subset of the states
  - ✓ Less harmed by violations of the Markov property. This is because they do not update their value estimates on the basis of the value estimates of successor states (do not use bootstrap)



## Temporal Difference 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

On line  
Incremental

Model free

TD Generalized Policy iteration for

TD Policy Evaluation + TD Policy Improvement

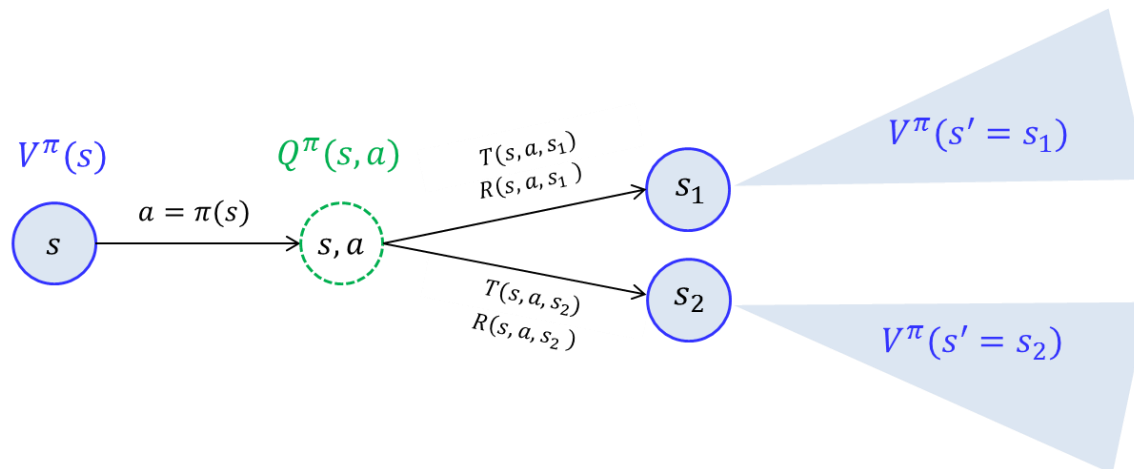
On-Policy TD Control (SARSA)

Off-Policy Q-learning Control

## Recall : Value function

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

Bootstrapping



## Monte Carlo Policy Evaluation

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

### Constant- $\alpha$ 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

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

Bootstrapping

### Temporal Difference Policy Evaluation ; $TD(0)$ :

After visiting  $s_t$  and transiting to  $s_{t+1}$  with a single 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

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

## Temporal Difference Policy Evaluation

### Algorithm : Tabular $TD(0)$ for estimating $V^\pi$

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

**Repeat** (for each episode):

Initialize  $s$

**Repeat** (for **each step** of episode)

$a \leftarrow$  action given by  $\pi$  for  $s$

Take action  $a$ ; observe reward  $r$  and next state  $s'$

$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$

$s \leftarrow s'$

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

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

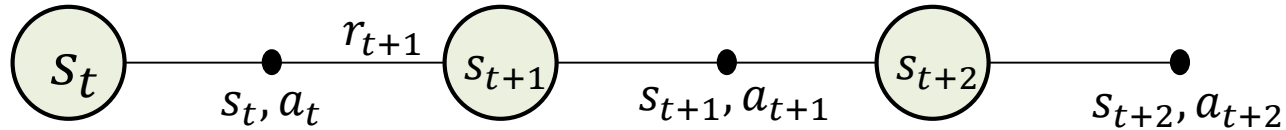
## Advantages of TD Policy Evaluation (prediction)

### 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  $V^\pi$  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 *constant* –  $\alpha$  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

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

### 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



## Classification of RL

		How to estimate $V^*(s)$ and $Q^*(s, a)$	
		Monte Carlo method	Temporal Difference methods
		Non-Bootstrap	Bootstrap
How to explore ?	On-policy	On-policy Monte Carlo Control	SARSA
	Off-policy	Off-policy Monte Carlo Control	Q-Learning (SARSmAxA)

- Episodic based
- Single-data-point based

## SARSA: On-Policy TD Control

### 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):

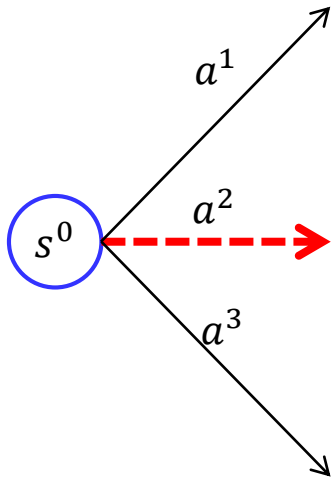
$\left\{ \begin{array}{l} \text{Take action } a \text{ given } s, \text{ observe } r, s' \\ \text{Choose } a' \text{ from } s' \text{ using policy derived from } Q \text{ (e.g., } \epsilon - greedy) \\ Q_{\pi}(s, a) \leftarrow Q_{\pi}(s, a) + \eta(r + \gamma Q_{\pi}(s', a') - Q_{\pi}(s, a)) \\ s \leftarrow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

Behavioral policy  
||  
Estimation policy

Until  $s$  is terminal

- As in all on-policy methods, we continually estimate  $Q^{\pi}$  for the behavioral policy, and the same time change  $\pi$  toward greediness with respect to  $Q^{\pi}$
- 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

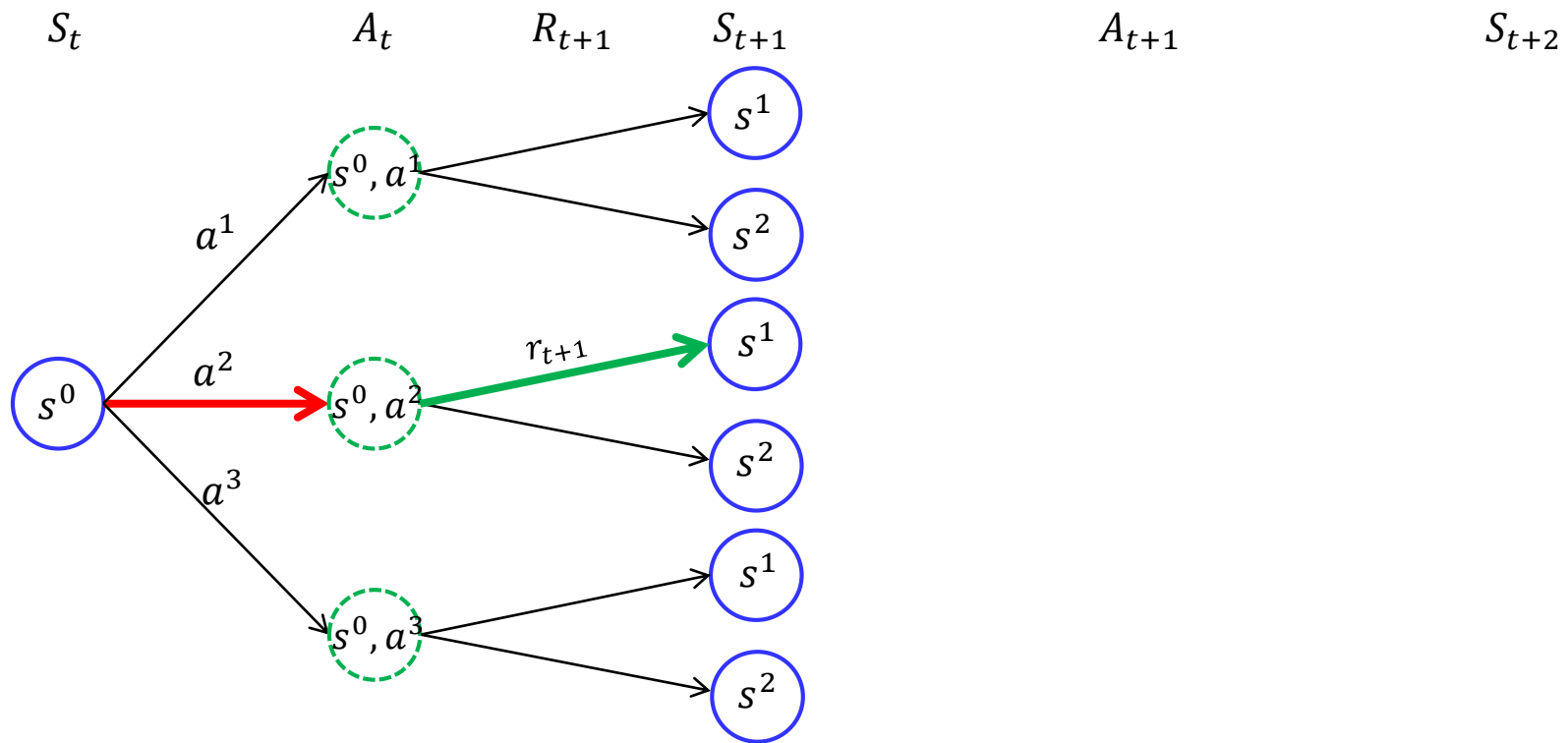
 $s_t$  $A_t$  $R_{t+1}$  $s_{t+1}$  $A_{t+1}$  $s_{t+2}$ 

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

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

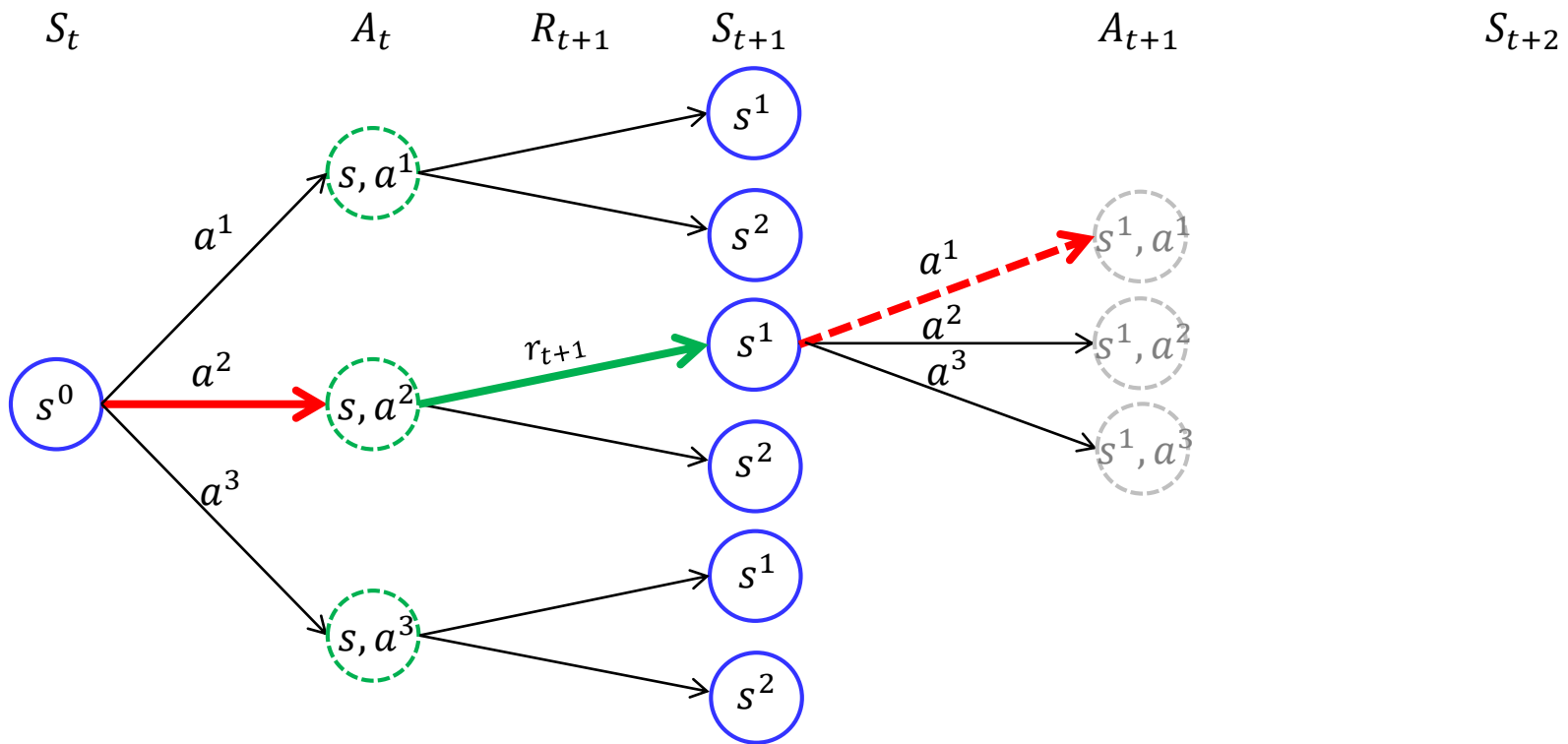
Assume  $a^2$  is chosen

## 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

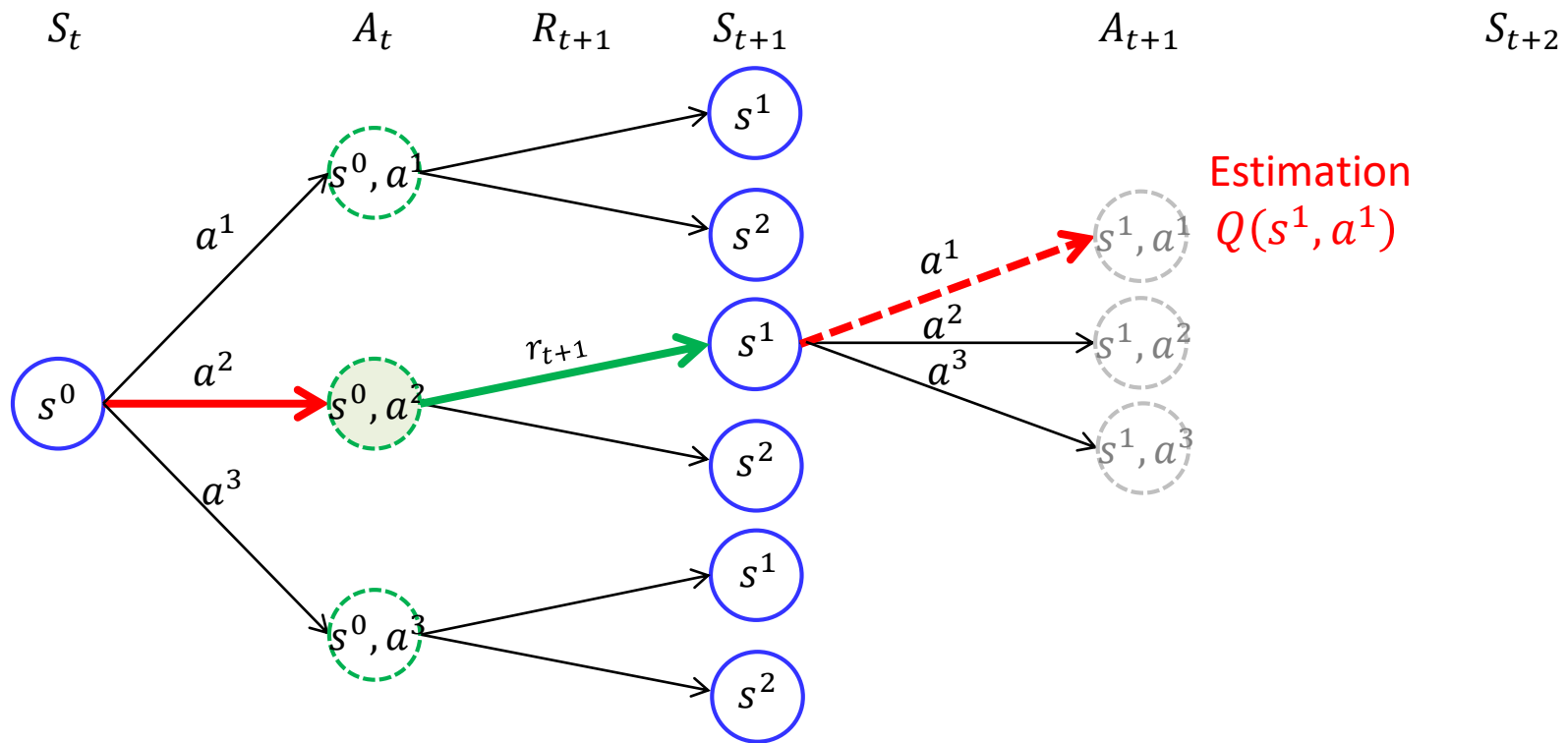


Choose  $a_{t+1}$  from  $s_{t+1} = s^1$  using current  $Q$

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

Assume  $a^1$  is chosen

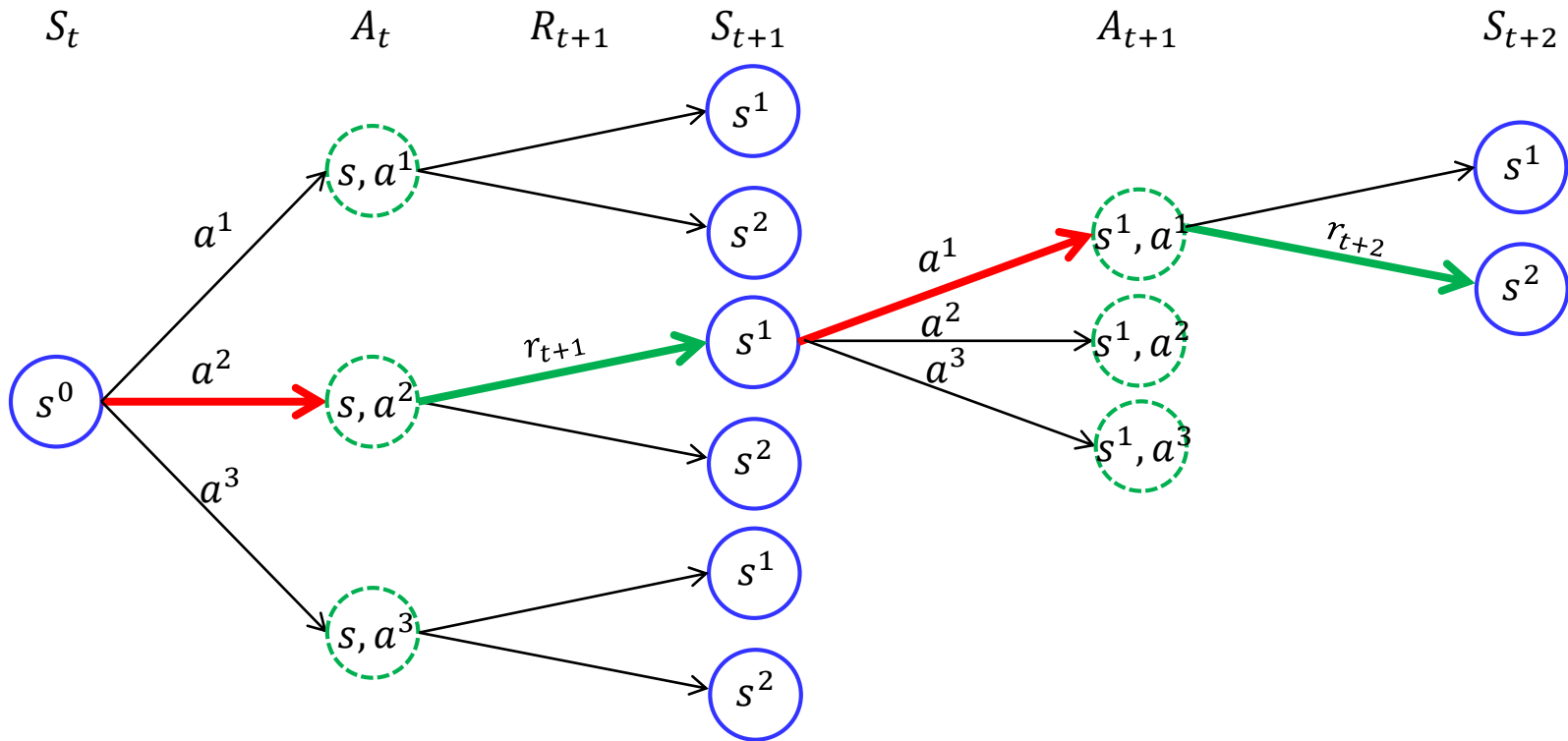
## Sarsa: On-Policy TD Control



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

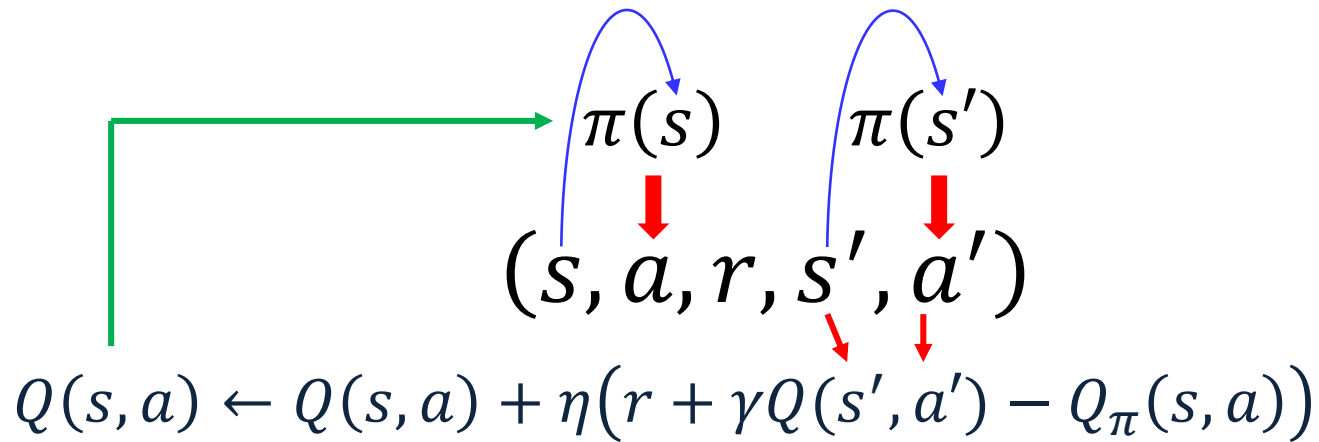
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$
$$\rightarrow Q(s^0, a^2) \leftarrow Q(s^0, a^2) + \alpha[r_{t+1} + \gamma Q(s^1, a^1) - Q(s^0, a^2)]$$

## Sarsa: On-Policy TD Control



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

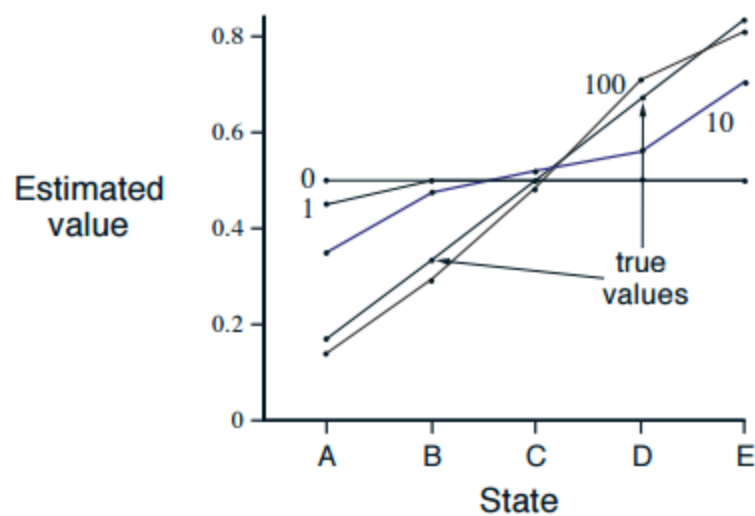
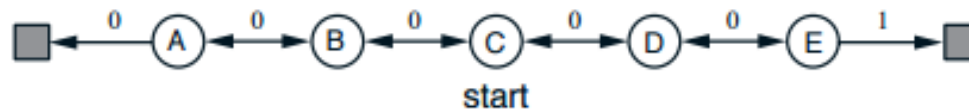
## Why Q-learning is considered as Off-Policy method



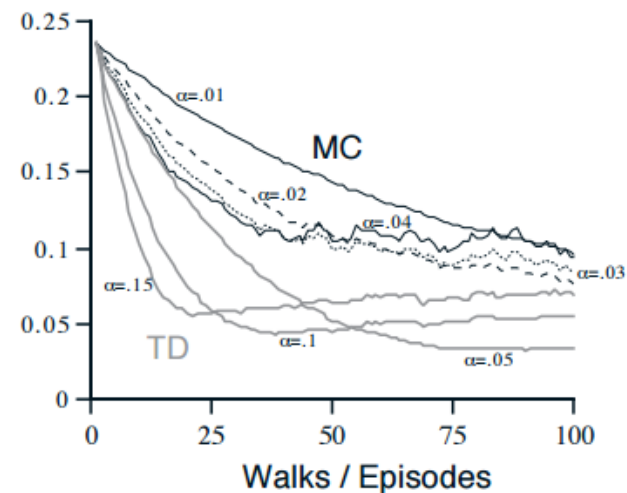


## Sarsa: On-Policy TD Control : Windy Grid world Example

A small Markov process for generating random walk

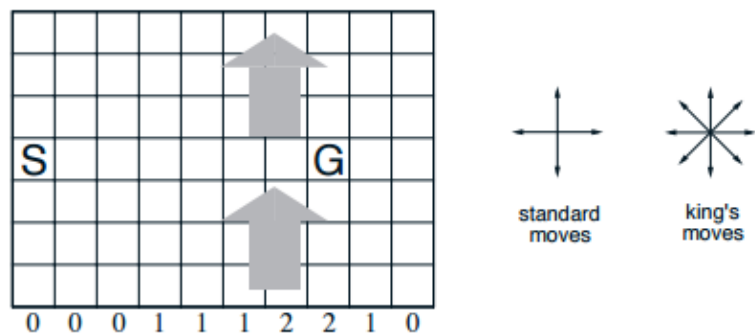


RMS error,  
averaged  
over states

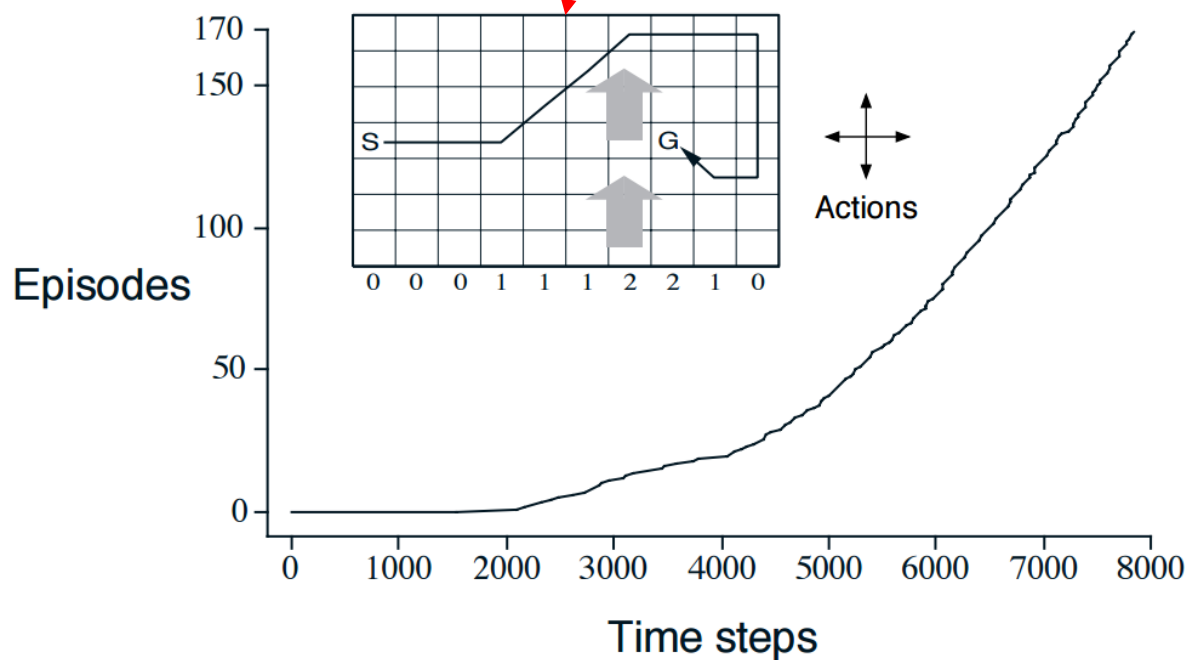


## Sarsa: On-Policy TD Control : Windy Grid world Example

**State transition is encoded by this figure**



-Optimal policy



## 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.,  $\epsilon - greedy$ )

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


### 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**

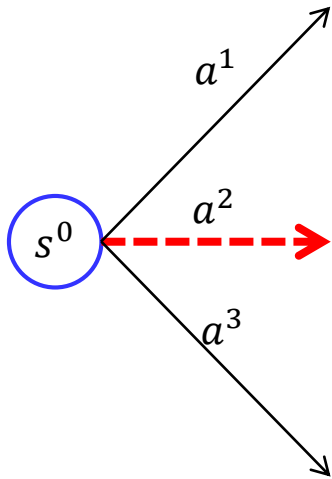
(Always try to estimate the optimal policy)

-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

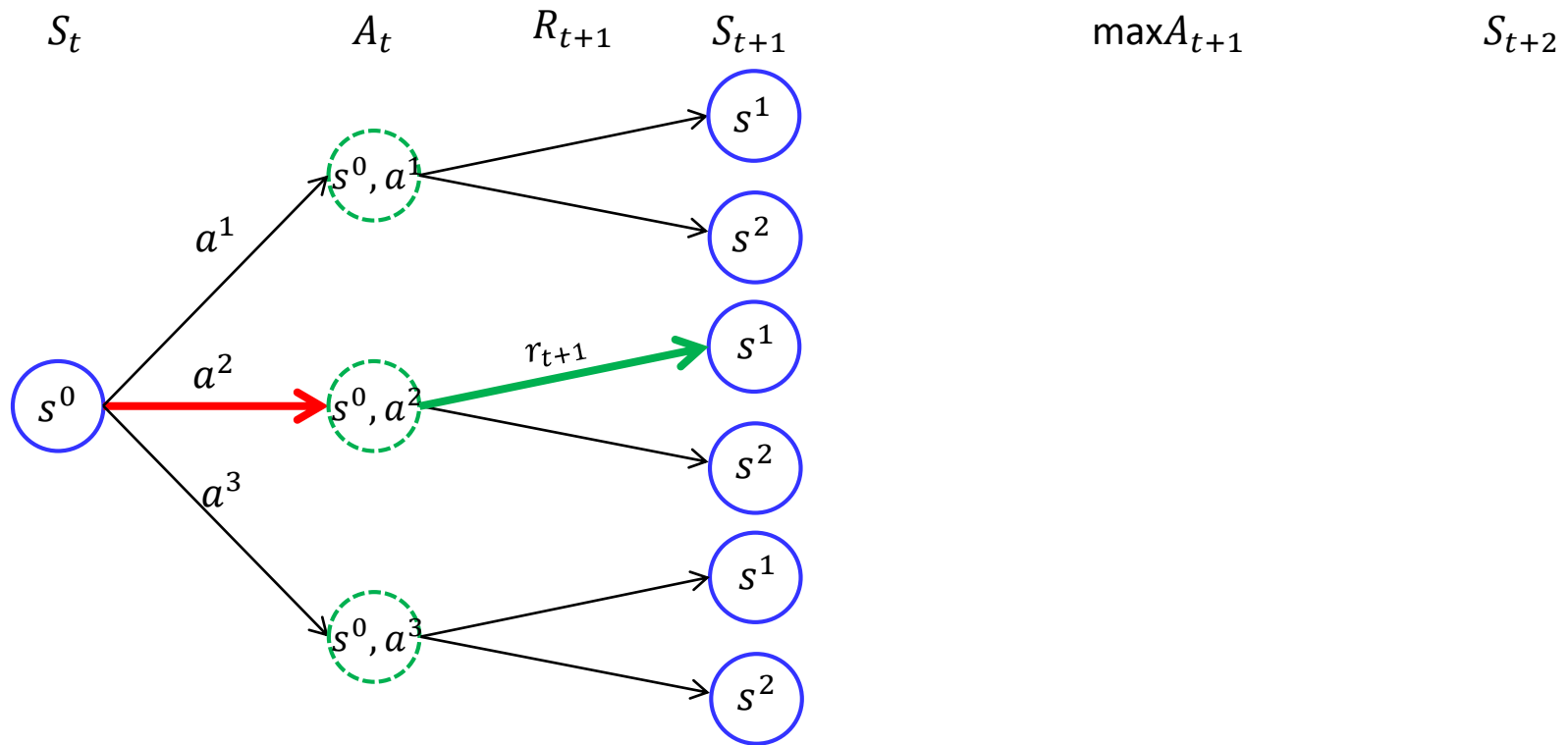
 $s_t$  $A_t$  $R_{t+1}$  $s_{t+1}$  $\max A_{t+1}$  $s_{t+2}$ 

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

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

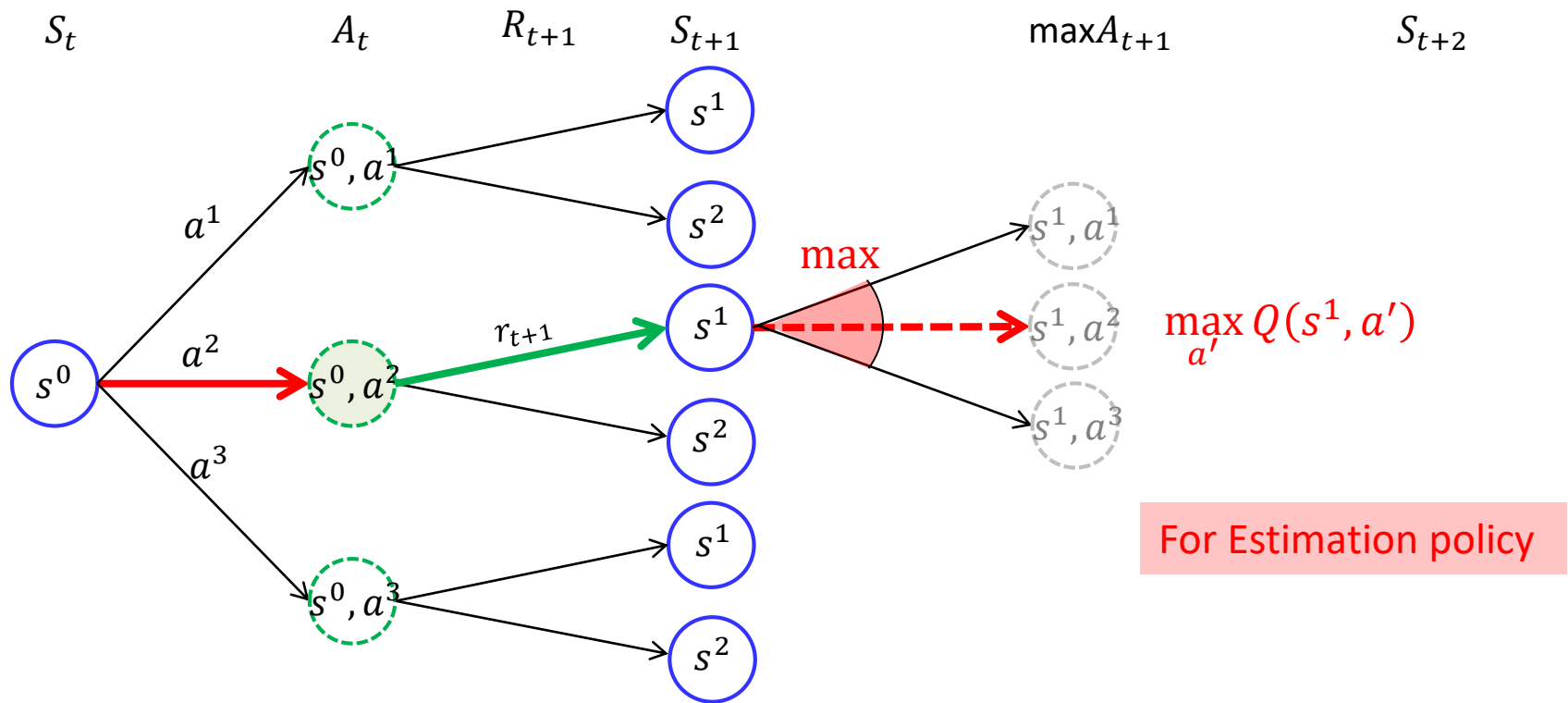
Assume  $a^2$  is chosen

## 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



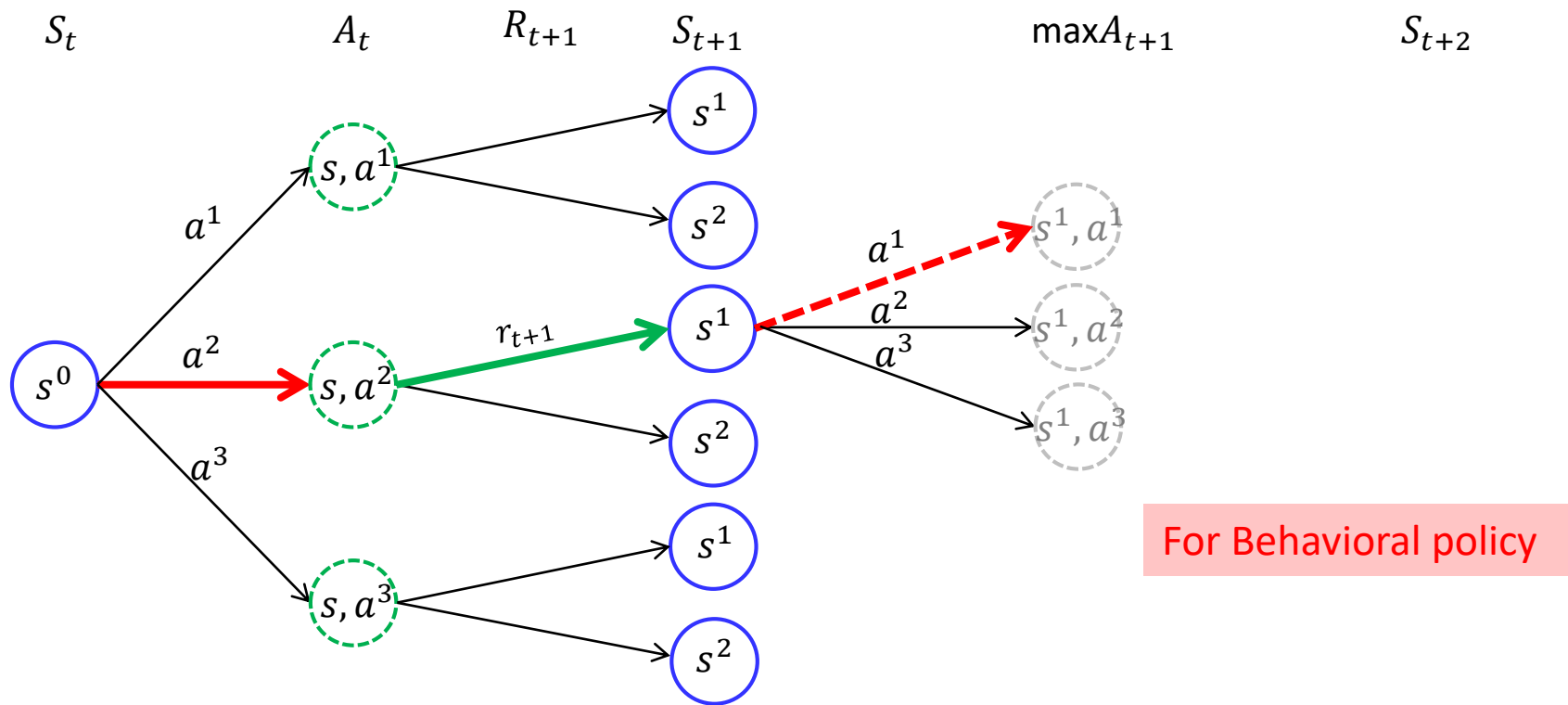
Update  $Q$  function with the  $\max_{a'} Q(s^1, a')$

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

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



## Q-Learning: Off-Policy TD Control

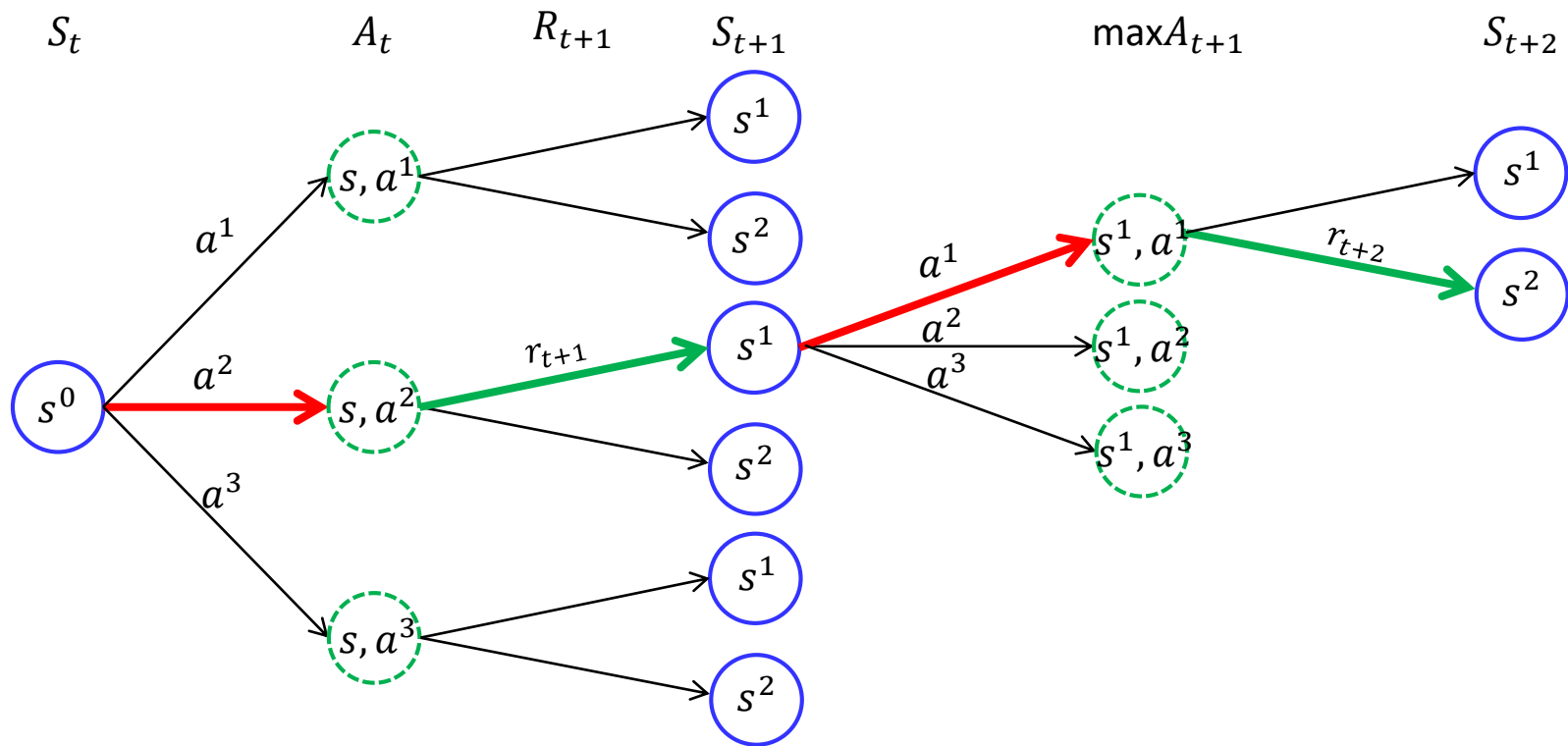


Choose  $a_{t+1}$  from  $s_{t+1} = s^1$  using current  $Q$

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

Assume  $a^1$  is chosen

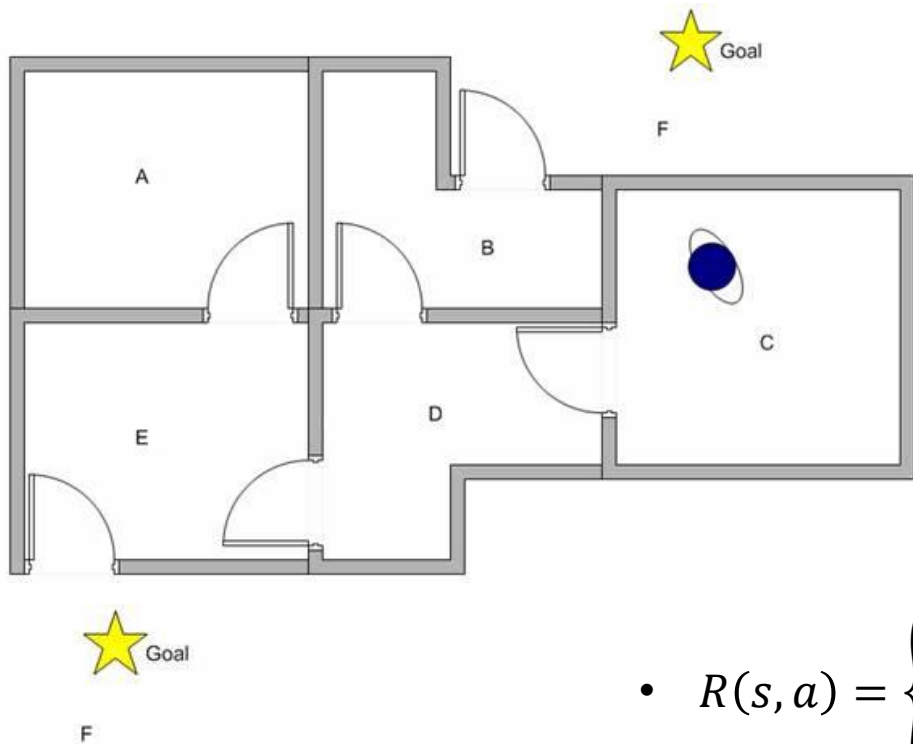
## Q-Learning: Off-Policy TD Control



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)

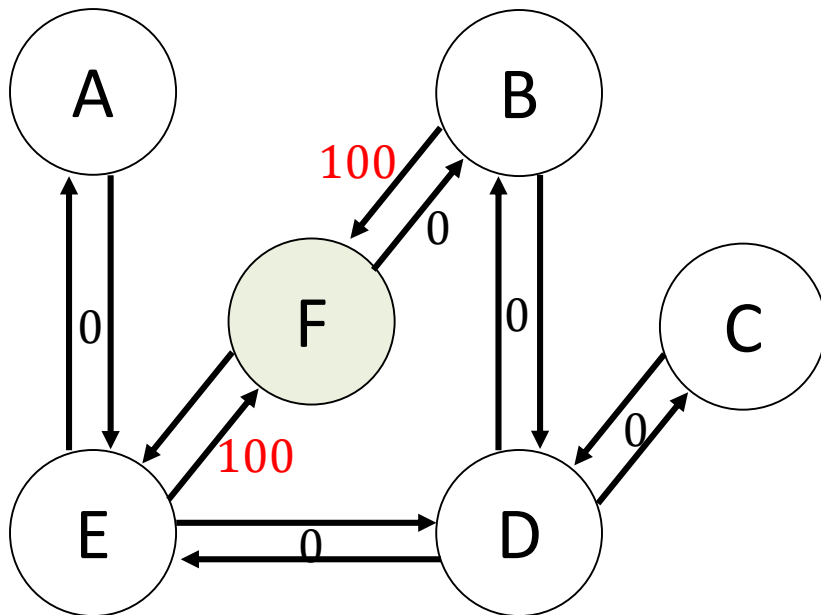


$$MDP = \{\mathcal{S}, \mathcal{A}, T, R, \gamma\}$$

- $s \in \mathcal{S} = \{A, B, C, D, E, F\}$
- $a \in \mathcal{A} = \{A, B, C, D, E, F\}$   
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.,  $T(C, D) = 1$

- $R(s, a) = \begin{cases} 0 & \text{if move to } a \text{ is allowed and } a \neq F \\ 100 & \text{if move to } a \text{ is allowed and } a = F \end{cases}$
- $\gamma = 0.8$

## Q-learning Step-by-Step Example



$R(s, a) =$

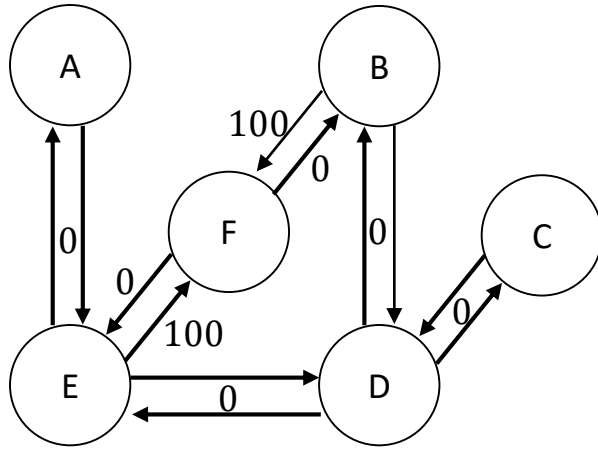
$s \backslash a$	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

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

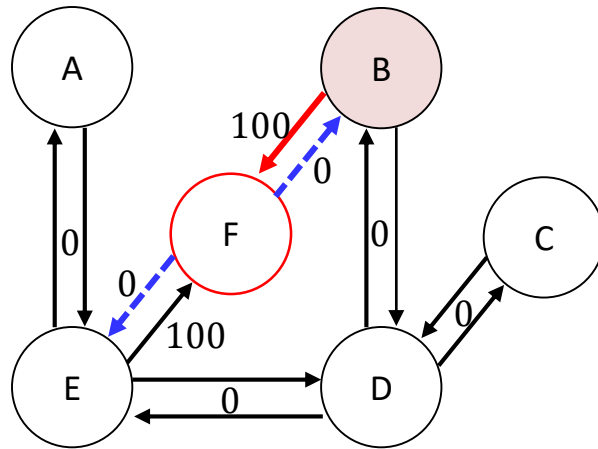
$Q$  Learning update rule:  $Q(s, a) \leftarrow Q(s, a) + \eta \left( r + \gamma \max_a Q(s', a) - Q(s, a) \right)$



[illegible]

## Q-learning Step-by-Step Example

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



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

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

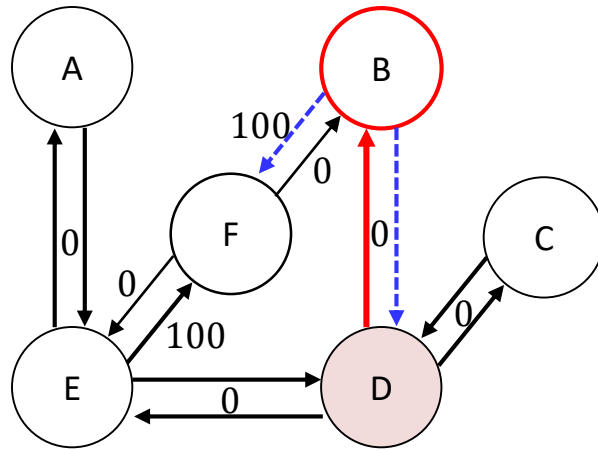
$$Q(B, F) \leftarrow Q(B, F) + 0.5 \left( R(B, F) + 0.8 \max_a \{Q(F, B), Q(F, E)\} - Q(B, F) \right)$$

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

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

## Q-learning Step-by-Step Example

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



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

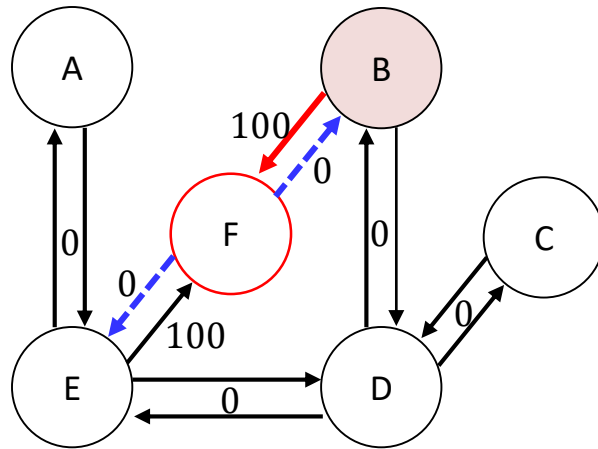
1. Assume the initial state is  $D$  and take action  $B$  randomly (stochastic policy):

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

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

## Q-learning Step-by-Step Example

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



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

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

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

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

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

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

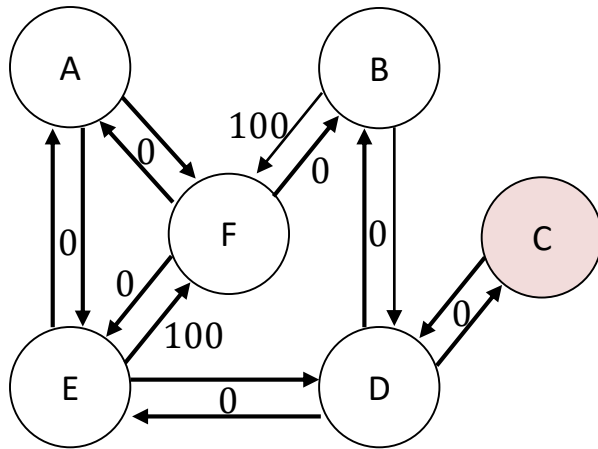
$$Q(B, F) \leftarrow 50 + 0.5(100 + 0.8 \times 0 - 50) = 75$$

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



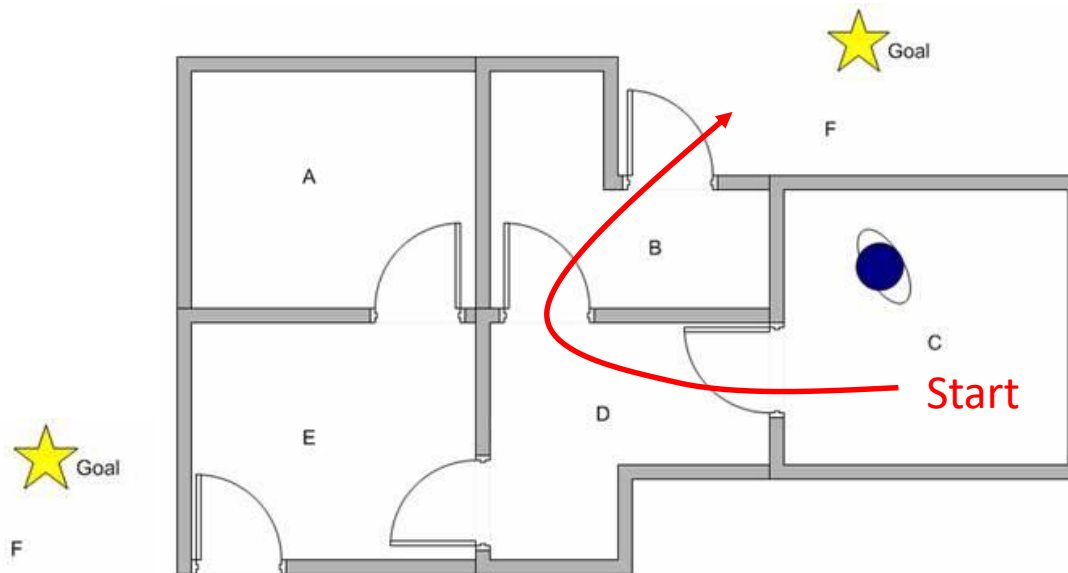
## Q-learning Step-by-Step Example

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

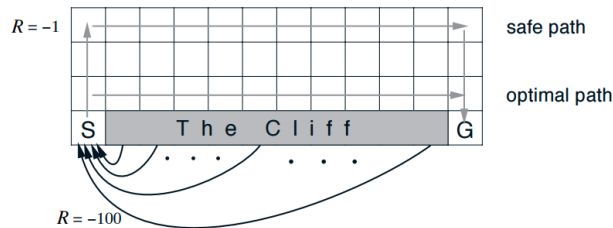


$$Q^*(s, a) = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\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

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  $\pi_B$

Estimation Policy  $\pi_E$

$$a'_B = \pi_B(s)$$

$\neq$

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

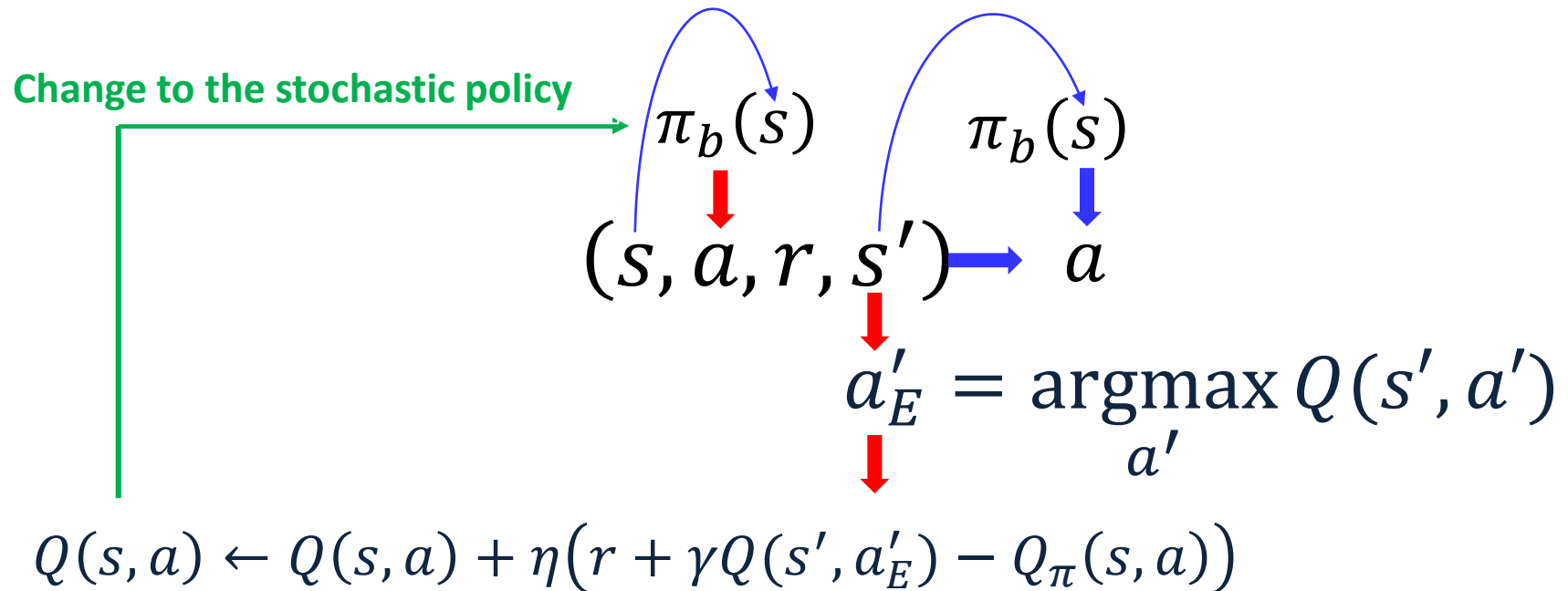
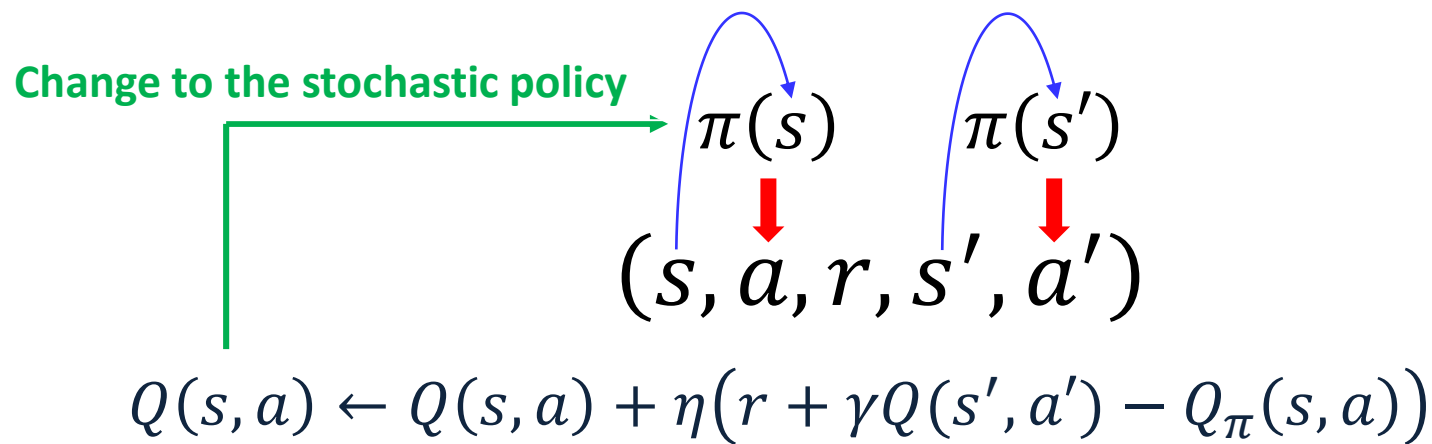
Used to generated data

Used to estimate  $Q(s, a)$

Take action  $a'_B$  and transit to the next state

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

## Why Q-learning is considered as Off-Policy method



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.

### Q-learning (stochastic gradient update)

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[ \left( r + \gamma \max_a Q(s', a) \right) - Q(s, a) \right]$$

#### Problem:

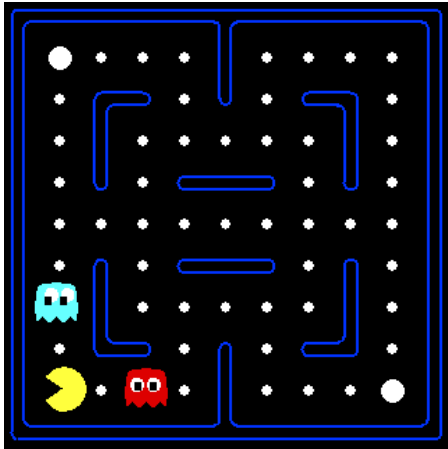
- Basic Q-Learning keeps a table of all q-values
- We cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Doesn't generalize to unseen states/actions

#### Solution:

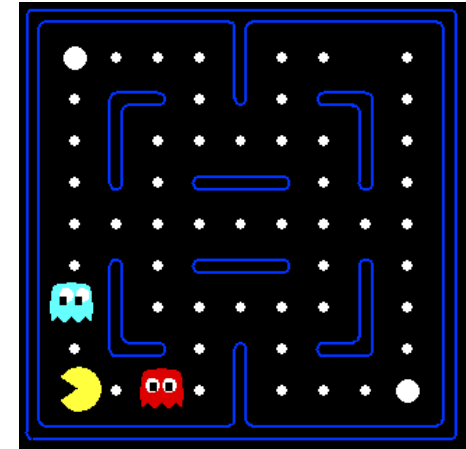
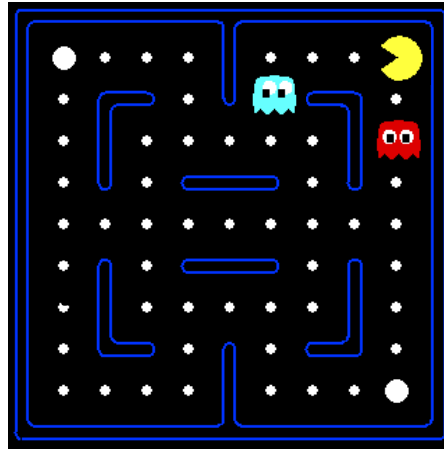
- Learn about some small number of training states from experience
- Generalize that experience to new, similar situations
  - fundamental idea in machine learning

## Q-learning with function approximation

Obtained experience:  
This state is bad!



Complete new states



- Are these states are good or bad?
- For table representation of  $Q(s, a)$ , **we cannot answer**
- **We can represent  $Q(s, a)$  using the properties of the current state  $s$ :**

$$\begin{aligned} Q(s, a; w) &= w_1 \phi_1(s, a) + w_2 \phi_2(s, a) + \dots + w_n \phi_n(s, a) \\ &= w^T \phi(s, a) \end{aligned}$$

$\phi_i(s, a)$ : distance to closet ghost, number of ghost, distance to foods

Approximate  $Q(s, a)$  as a function:

$$\begin{aligned} Q(s, a; w) &= w_1 \phi_1(s, a) + w_2 \phi_2(s, a) + \dots + w_n \phi_n(s, a) \\ &= w^T \phi(s, a) \end{aligned}$$

$w$  : weight vector  $\phi(s, a)$  : features vector

- Features,  $\phi(s, a) = \{\phi_1(s, a), \dots, \phi_n(s, a)\}$ , are supposed to be properties of the state-action  $(s, a)$  pair that are indicative of the quality of taking action  $a$  and state  $s$



## Q-learning with function approximation

### Algorithm : Q-learning with function approximation

On each  $(s, a, r, s')$ :

$$w \leftarrow w + \eta \left[ \underbrace{\left( r + \gamma \max_a \hat{Q}(s', a; w) \right)}_{\text{Target}} - \underbrace{\hat{Q}(s, a; w)}_{\text{Estimate}} \right] \phi(s, a)$$

This is equivalent to find the weight  $w$  that maximizes the following objective function

$$\min_{\hat{Q}_\pi} \frac{1}{2} \sum_{(s,a,r,s')} \left( \underbrace{\left( r + \gamma \max_a \hat{Q}(s', a; w) \right)}_{\text{Target}} - \underbrace{\hat{Q}(s, a; w)}_{\text{Estimate}} \right)^2$$

For a single transition  $(s, a, r, s')$ , the error can be expressed as:

$$\begin{aligned} \text{Error}(w) &= \frac{1}{2} \left( \left( r + \gamma \max_a \hat{Q}(s', a; w) \right) - \hat{Q}(s, a; w) \right)^2 \\ &= \frac{1}{2} \left( \left( r + \gamma \max_a \hat{Q}(s', a; w) \right) - \sum_{k=1}^n w_k \phi_k(s, a) \right)^2 \quad \because \hat{Q}(s, a; w) = \sum_{k=1}^n w_k \phi_k(s, a) \\ \frac{\partial \text{Error}(w)}{\partial w_k} &= - \left( \left( r + \gamma \max_a \hat{Q}(s', a; w) \right) - \sum_{k=1}^n w_k \phi_k(s, a) \right) \phi_k(s, a) \end{aligned}$$

$$w_k \leftarrow w_k + \eta \left[ \left( r + \gamma \max_a \hat{Q}(s', a; w) \right) - \hat{Q}(s, a; w) \right] \phi(s, a)$$

## Deep Reinforcement Learning

Use a neural network for estimating  $Q(s, a)$

Playing Atari [Google DeepMind, 2013]:

$$a^* = \pi(s =$$

move left  
move right  
stroke



Figure 1: Atari Breakout game. Image credit: DeepMind.

- last 4 frames (images)  $\Rightarrow$  3-layer NN  $\Rightarrow$  keystroke (move left, right, stroke)
- $\epsilon$ -greedy, train over 10M frames with 1M replay memory
- Human-level performance on some games (breakout)

## Q-Learning with Neural Network

Screen size :  $84 \times 84$  and convert to grayscale with 256 gray levels

State size =  $256^{84 \times 84 \times 4} \approx 10^{67970}$  more than the number of atoms in the known universe

$$Q(s, a) = \left. \begin{array}{c} \text{Red box} \end{array} \right\} 10^{67970}$$

- Need to represent this large Q table using a function approximation (Deep learning)
- Need to estimate Q-values for states that have never been seen before (generalization)

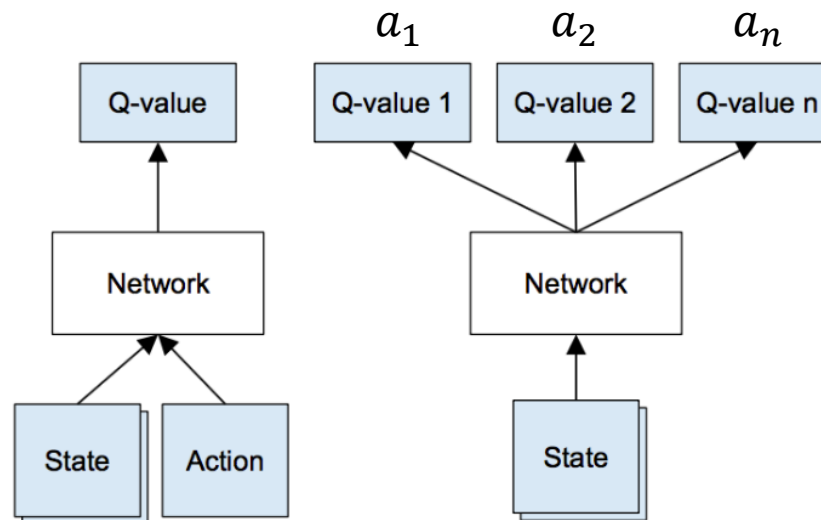


Figure 3: Left: Naive formulation of deep Q-network. Right: More optimized architecture of deep Q-network, used in DeepMind paper.

### 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

## Actor Critic Algorithm

## Motivation

- One of the primary goals of the field of artificial intelligence is to solve complex tasks from unprocessed, high-dimensional, sensory input.
- Recent progresses in RL have shown the possibilities
  - Deep Q Network for Atari games and
  - AlphaGo
- Although DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces
- Discretization does not solve the issues
  - Assume  $a_t = (a_{t,1}, a_{t,2}, \dots, a_{t,M})$  and  $a_{t,i} \in \{-k, 0, k\}$ , then it requires  $3^M$  possible actions
  - Such large action spaces are difficult to explore efficiently
  - Successful training DQN-like networks is intractable

## Motivation

- Assume the action at time is a profile of the actions of  $N$  agents:

$$a^t = \begin{bmatrix} a_1^t \\ a_2^t \\ \vdots \\ a_N^t \end{bmatrix} \quad \text{If } a_i^t \in \{1, \dots, K\} \quad |a^t| = K^N$$

- Treat each action is continuous value to relax cardinality issue

$$a^t = \begin{bmatrix} a_1^t \\ a_2^t \\ \vdots \\ a_N^t \end{bmatrix} \quad \text{If } a_i^t \in [0, K]$$

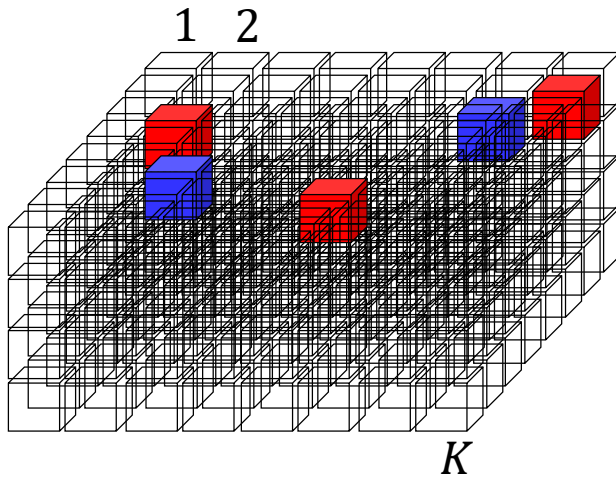
- Now, our action is a  $N$ -dimensional vector  $a^t \in \mathbb{R}^N$
- $\pi(s) = \operatorname{argmax}_a Q(s, a)$  is difficult to solve
  - Policy gradient method
  - Directly optimize the best action in the given state

## Motivation

$$a^t = \begin{bmatrix} a_1^t \\ a_2^t \\ \vdots \\ a_N^t \end{bmatrix}$$

If  $a_i^t \in [0, K]$

$$a^t = \begin{bmatrix} a_1^t \\ a_2^t \\ a_3^t \\ a_4^t \\ a_5^t \end{bmatrix}$$



$a_1^t$ : The cell that digger 1 need to excavate

$a_4^t$ : The soil that mover 4 need to dump away

- Discrete  $\rightarrow$  continuous  $\rightarrow$  discrete

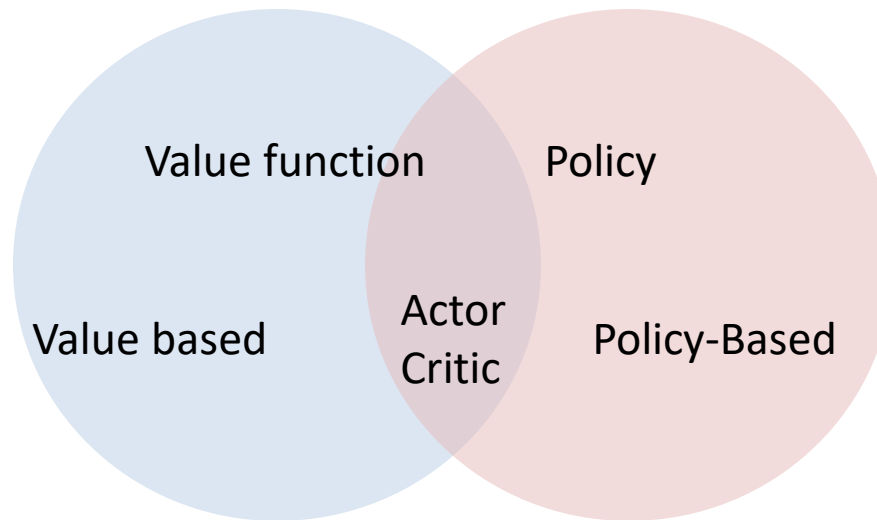


### Continuous Control with Deep Reinforcement Learning

Lillicrap et al., (2016) [google.com](https://arxiv.org/abs/1612.07982)

- Develop a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces
- It combines three components:
  - Actor-critic algorithm (Sutton et al., 1999)
  - Deep Q Network (Mnih et al, 2015)
  - Deterministic Policy Gradient (Silver 2015)

## Actor-Critic Method



## Policy Objective Function

- Stochastic policy can be parameterized as

$$\pi_{\theta}(s, a) = P(a|s, \theta)$$

- We can measure the quality of the policy  $\pi_{\theta}$  using the **policy objective functions**  $J(\theta)$ :
  - In episodic environments we can use the start value:

$$J_1(\theta) = V^{\pi_{\theta}}(s_1) = \mathbb{E}_{\pi_{\theta}}[v_1]$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_{\theta}}(s) V^{\pi_{\theta}}(s)$$

- Average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_{\theta}}(s) \sum_a \pi_{\theta}(s, a) R(s, a)$$

✓  $d^{\pi_{\theta}}(s)$  is stationary distribution of Markov chain for  $\pi_{\theta}$

## Policy Gradient

- The policy gradient  $\nabla_{\theta} J(\theta)$  is given as

$$\nabla_{\theta} J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- The parameters for the policy  $\pi_{\theta}(s, a) = P(a|s, \theta)$  can be updated using gradient ascent

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

where  $\alpha$  is a step-size parameter

## Policy Gradient Theorem (Stochastic policy)

- Consider a simple class of one-step MDPs
  - Starting in state  $s \sim d(s)$
  - Terminating after one time-step with reward  $r = R(s, a)$
- Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_{\theta}}[r] = \sum_s d(s) \sum_a \pi_{\theta}(s, a) R(s, a)$$

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \sum_s d(s) \sum_a \nabla \pi_{\theta}(s, a) R(s, a) \\ &= \sum_s d(s) \sum_a \pi_{\theta}(s, a) \nabla \log \pi_{\theta}(s, a) R(s, a) \\ &= \mathbb{E}_{\pi_{\theta}}[\nabla \log \pi_{\theta}(s, a) R(s, a)] \end{aligned}$$

$$\nabla \pi_{\theta}(s, a) = \pi_{\theta}(s, a) \frac{\nabla \pi_{\theta}(s, a)}{\pi_{\theta}(s, a)} = \pi_{\theta}(s, a) \nabla \log \pi_{\theta}(s, a)$$

## Policy Gradient Theorem (Stochastic policy)

### Theorem

For any differentiable policy  $\pi_{\theta}(s, a)$ ,

For any of the policy objective functions  $J = J_1, J_{avR}$ , or  $\frac{1}{1-\gamma} J_{avV}$

The (stochastic) policy gradient is

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) Q^{\pi_{\theta}}(s, a)]$$

- Expectation over (state and action)
- Policy gradient is (score function)  $\times$  (action-value function)

## Monte-Carlo Policy Gradient (Reinforce)

- Using policy gradient theorem
- Using return  $v_t$  as an unbiased sample of  $Q^{\pi_\theta}(s, a)$ , Update parameters by stochastic gradient ascent:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)] \\ \sim \nabla \log \pi_\theta(s, a) v_t$$

$$\Delta \theta_t = \alpha \nabla \log \pi_\theta(s, a) v_t$$

### **function REINFORCE**

Initialise  $\theta$  arbitrarily

**for** each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  **do**

**for**  $t = 1$  to  $T - 1$  **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

**end for**

**end for**

**return**  $\theta$

**end function**

## Reducing Variance Using a Critic

- Monte-Carlo policy gradient has high variance, especially when the episode is long
- Use a critic to estimate the action-value function (Bootstrap)

$$Q^W(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms follow an approximate policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

$$\sim \nabla \log \pi_\theta(s, a) v_t \quad \text{(Sample trajectory)}$$

$$\sim \nabla \log \pi_\theta(s, a) Q^W(s, a) \quad \text{(Bootstrap)}$$

- Actor-critic algorithms maintain two sets of parameters
  - **Critic:** Update action-value function parameters
  - **Actor:** Update policy parameters  $\theta$ , in direction suggested by critic



## Actor-Critic Algorithm

**function** QAC

    Initialise  $s, \theta$

    Sample  $a \sim \pi_\theta$

**for** each step **do**

        Sample reward  $r = \mathcal{R}_s^a$ ; sample transition  $s' \sim \mathcal{P}_{s,\cdot}^a$ .

        Sample action  $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a', s \leftarrow s'$

**end for**

**end function**

- Actor: Updates  $\theta$  by policy gradient
- Critic: Updates  $w$  by linear TD(0)

## Deterministic Policy Gradient

### Motivation

- The majority of model-free reinforcement learning algorithm are based on generalized policy iteration:
  - Policy evaluation : estimate the action value function  $Q^{\mu^k}(s, a)$
  - Policy improvement: update the policy with respect to the estimated action value function:

$$\mu^{k+1}(s) = \underset{a}{\operatorname{argmax}} Q^{\mu^k}(s, a)$$

- In continuous action spaces, greedy policy improvement become problematic, requiring a global maximization at every step.
- Instead, a simple and computationally attractive alternative is to move the policy in the direction of the gradient of  $Q$ , rather than globally maximizing  $Q$

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s \sim \rho^{\mu^k}} \left[ \nabla Q^{\mu^k}(s, \mu_{\theta}(s)) \right]$$

- By applying **the chain rule**,

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s \sim \rho^{\mu^k}} \left[ \nabla \mu_{\theta}(s) \nabla_a Q^{\mu^k}(s, a) \Big|_{a=\mu_{\theta}(s)} \right]$$

## Deterministic Policy Gradient

### Theorem (Deterministic policy gradient)

Given that  $\nabla_{\theta} \mu_{\theta}(s, a)$  and  $\nabla_a Q^{\mu_{\theta}}(s, a)$  exist, the deterministic policy gradient is defined as

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \int_{\mathcal{S}} \rho^{\mu}(s) \nabla \mu_{\theta}(s, a) \nabla_a Q^{\mu_{\theta}}(s, a) \Big|_{a=\mu_{\theta}} ds \\ &= \mathbb{E}_{s \sim \rho^{\mu}} \left[ \nabla \mu_{\theta}(s, a) \nabla_a Q^{\mu_{\theta}}(s, a) \Big|_{a=\mu_{\theta}} \right]\end{aligned}$$

$$\rho^{\mu}(s') := \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \rightarrow s', t, \mu) ds$$

## Deterministic Actor-Critic Algorithms

- On-policy Deterministic Actor-Critic Method (SARSA)

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t)$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \delta_t \nabla_\theta J(\theta)$$

$$= \theta_t + \alpha_\theta \delta_t \mathbb{E}_{s \sim \rho^\mu} \left[ \nabla \mu_\theta(s_t, a_t) \nabla_a Q^{\mu_\theta}(s_t, a_t) \Big|_{a=\mu_\theta(s)} \right]$$

$$\sim \theta_t + \alpha_\theta \delta_t \nabla \mu_\theta(s_t, a_t) \nabla_a Q^{\mu_\theta}(s_t, a_t) \Big|_{a=\mu_\theta(s)}$$

Stochastic ascent

$$\sim \theta_t + \alpha_\theta \delta_t \nabla \mu_\theta(s_t, a_t) \nabla_a Q^w(s_t, a_t) \Big|_{a=\mu_\theta(s)}$$

Use Critic

## Deterministic Actor-Critic Algorithms

- Off-policy Deterministic Actor-Critic Method (SARSA)

Learn a deterministic target policy  $\mu_\theta(s)$  from trajectories generated by an arbitrary stochastic behavior policy

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t)$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \delta_t \nabla_\theta J(\theta)$$

$$= \theta_t + \alpha_\theta \delta_t \mathbb{E}_{s \sim \rho^\mu} \left[ \nabla \mu_\theta(s_t, a_t) \nabla_a Q^{\mu_\theta}(s_t, a_t) \Big|_{a=\mu_\theta(s)} \right]$$

$$\sim \theta_t + \alpha_\theta \delta_t \nabla \mu_\theta(s_t, a_t) \nabla_a Q^{\mu_\theta}(s_t, a_t) \Big|_{a=\mu_\theta(s)}$$

Stochastic ascent

$$\sim \theta_t + \alpha_\theta \delta_t \nabla \mu_\theta(s_t, a_t) \nabla_a Q^w(s_t, a_t) \Big|_{a=\mu_\theta(s)}$$

Use Critic

### Continuous Control with Deep Reinforcement Learning

Lillicrap et al., (2016) [google.com](https://arxiv.org/abs/1509.02931)

- Develop a model-free, off-policy **actor-critic algorithm** using **deep function approximator** that can learn policies in **high-dimensional, continuous action spaces**
- It combines three components:
  - **Actor-critic algorithm** (Sutton et al., 1999)
  - **Deep Q Network** (Mnih et al, 2015)
  - **Deterministic Policy Gradient** (Silver 2015)

# Deep-Deterministic Policy Gradient (DDPG) Algorithms

---

**Algorithm 1** DDPG algorithm

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1, M **do**

    Initialize a random process  $\mathcal{N}$  for action exploration

    Receive initial observation state  $s_1$

**for** t = 1, T **do**

        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}))|\theta^{Q'}$

        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Critic:  $\theta_{t+1}^Q = \theta_t^Q + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$

Actor:  $\theta_{t+1}^\mu = \theta_t^\mu + \alpha_\theta \delta_t \nabla_\theta J(\theta)$   
 $= \theta_t^\mu + \alpha_\theta \delta_t \mathbb{E}_{s \sim \rho^\mu} \left[ \nabla_{\theta^\mu} \mu(s_t) \nabla_a Q^{\mu\theta}(s_t, a_t) \Big|_{a=\mu_\theta(s_t)} \right]$

→ Off-policy RL

→ Reply buffer

→ TD(0) value evaluation

## Deep-Deterministic Policy Gradient (DDPG) Algorithms

---

### Algorithm 1 DDPG algorithm

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1, M **do**

    Initialize a random process  $\mathcal{N}$  for action exploration

    Receive initial observation state  $s_1$

**for** t = 1, T **do**

        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

**end for**

**end for**

---



## Deep-Deterministic Policy Gradient (DDPG) Algorithms

---

### Algorithm 1 DDPG algorithm

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1, M **do**

    Initialize a random process  $\mathcal{N}$  for action exploration

    Receive initial observation state  $s_1$

**for** t = 1, T **do**

        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

$$\begin{aligned} \text{Actor: } \theta_{t+1}^\mu &= \theta_t^\mu + \alpha_\theta \delta_t \nabla_{\theta} J(\theta) \\ &= \theta_t^\mu + \alpha_\theta \delta_t \mathbb{E}_{s \sim \rho^\mu} \left[ \nabla_{\theta^\mu} \mu(s) \nabla_a Q^{\mu\theta}(s, a_t) \Big|_{a=\mu_\theta(s_t)} \right] \end{aligned}$$