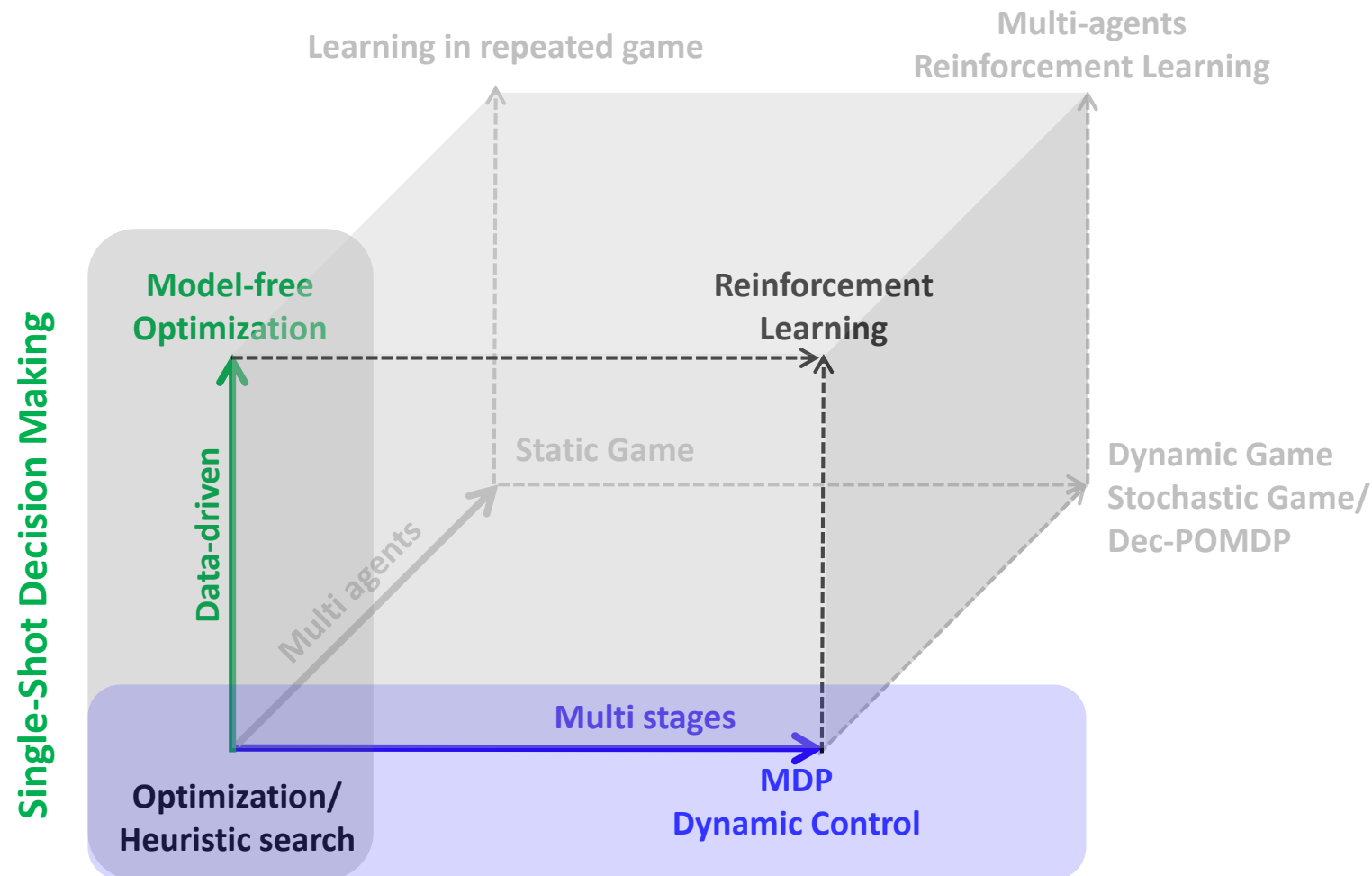


6. Markov Decision Process and Value Based Reinforcement Learning

Markov Decision Process and Dynamic Programming

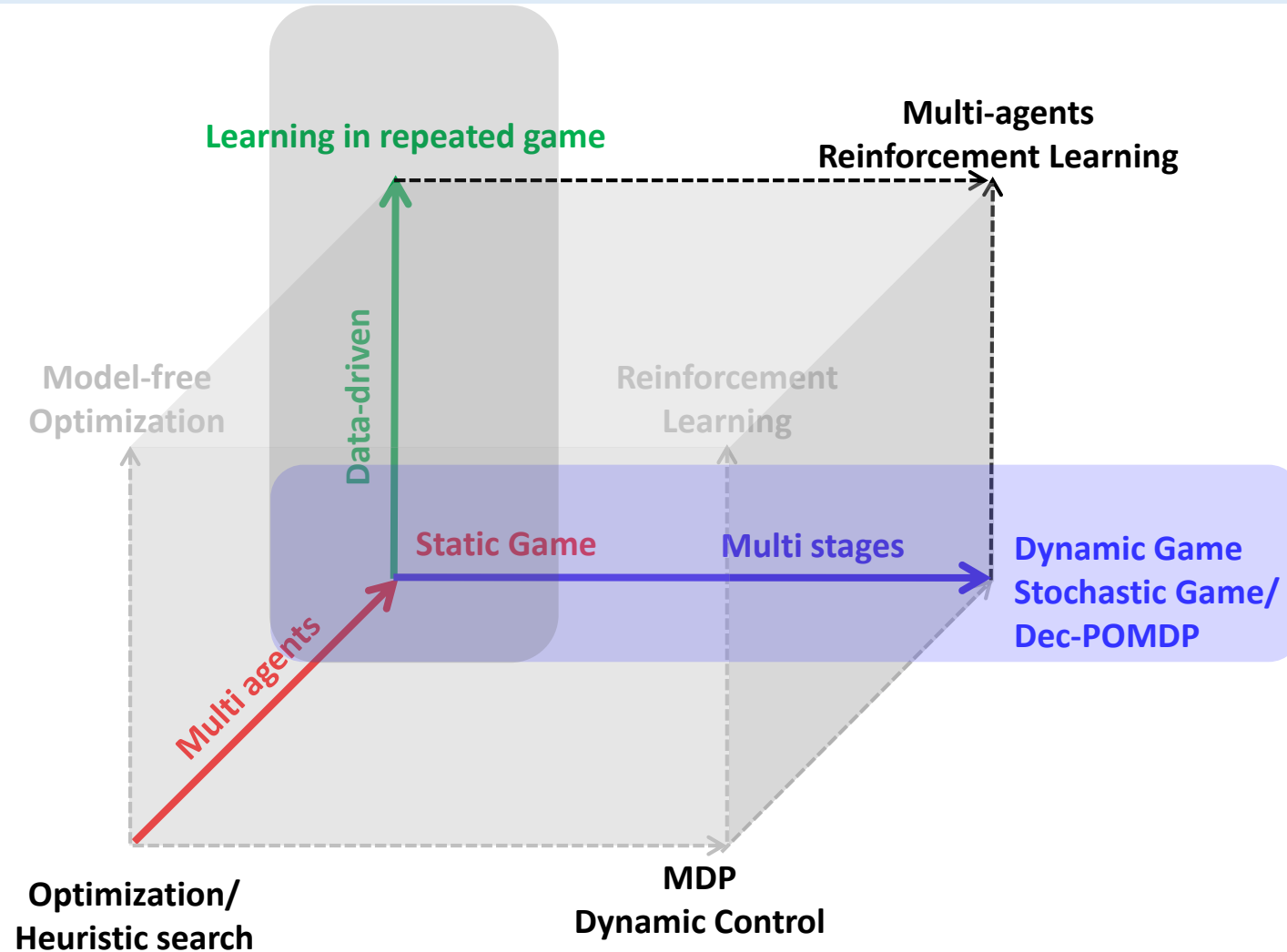
1. Definition of Markov Decision Process
2. Dynamic programming approach
 - Policy Evaluation
 - Policy Improvement
 - Policy iteration
 - Value Iteration

Problem Solving



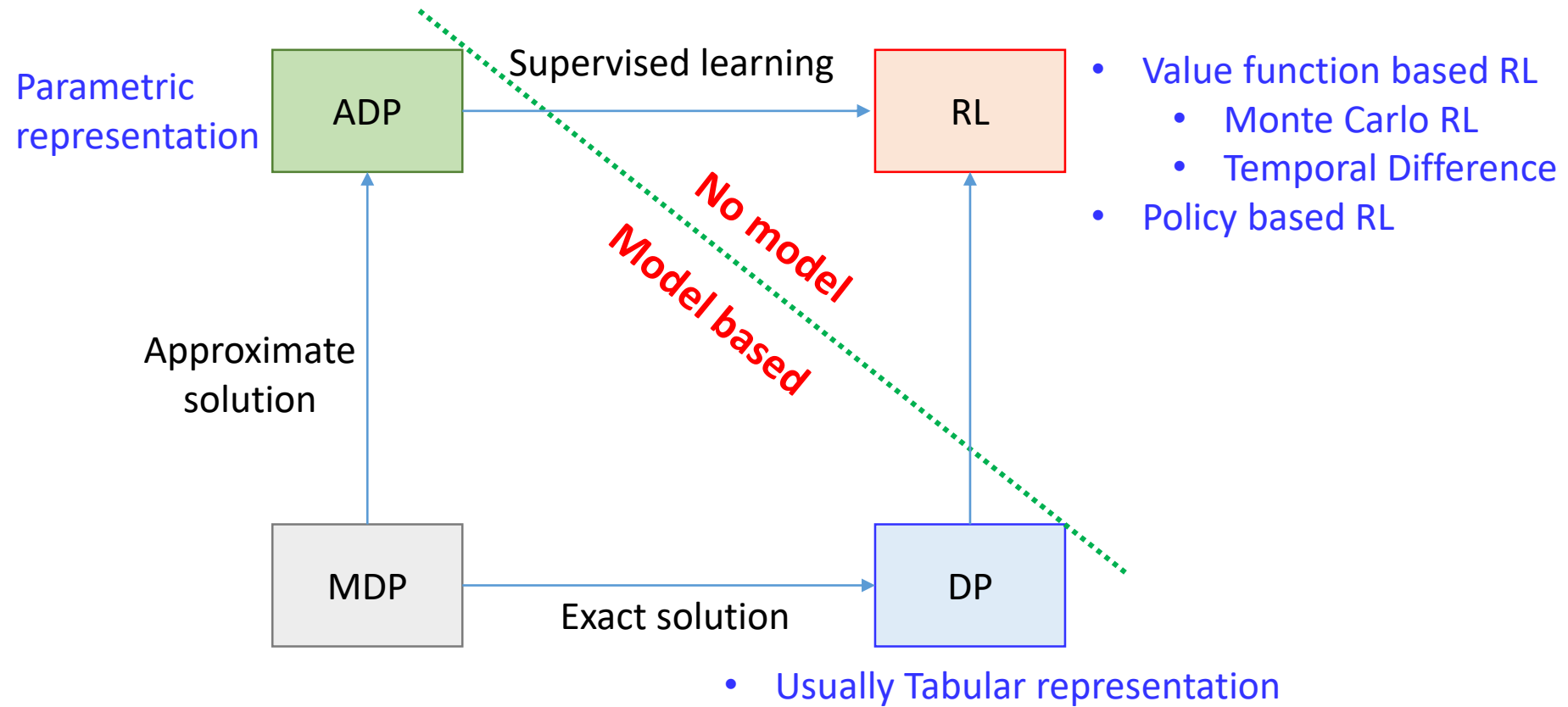
- Requires resonating about future *sequences* of actions and observations
- Requires resonating about actions of other players

Problem Solving



- Requires resonating about future *sequences* of actions and observations
- Requires resonating about actions of other players

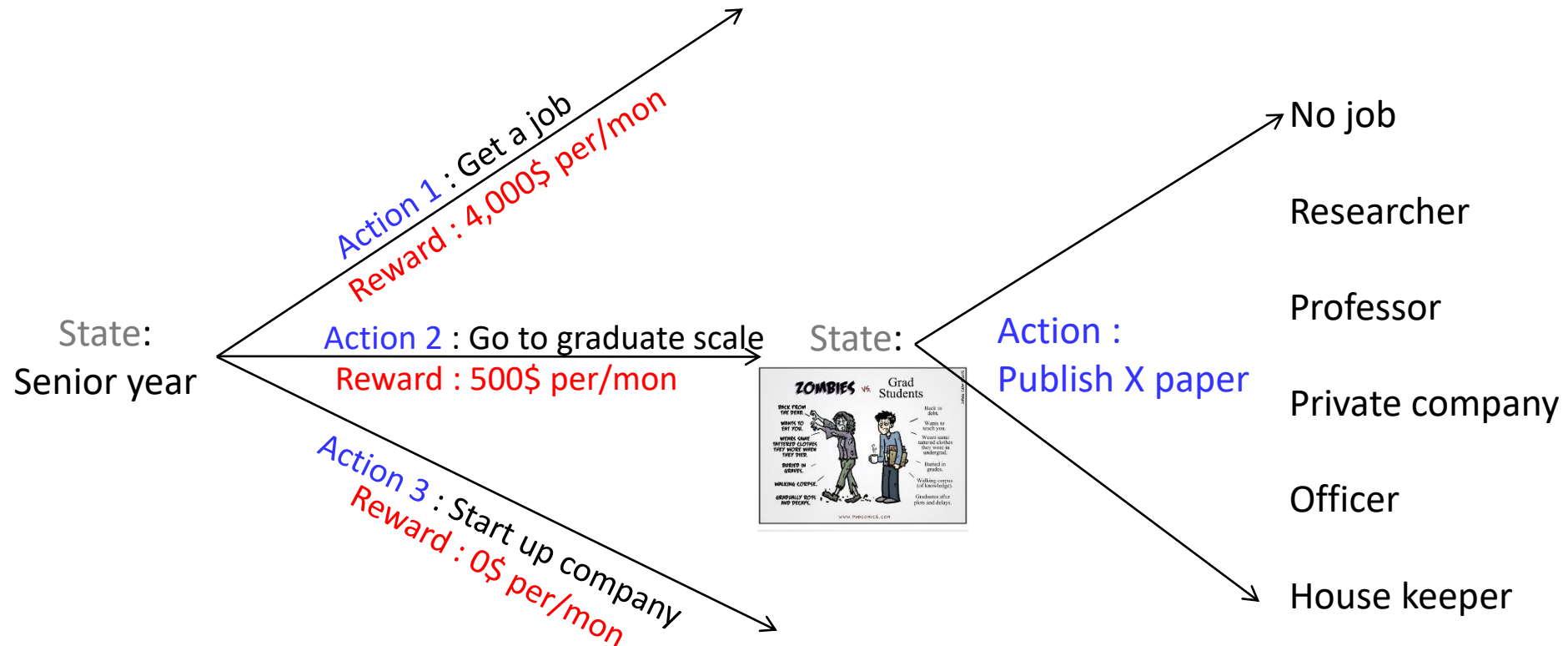
Contents



Introduction

Sequential Decision Making in Uncertainties

- Many important problems require the decision maker to make a series of decisions.
- It requires resonating about future *sequences* of actions and observations

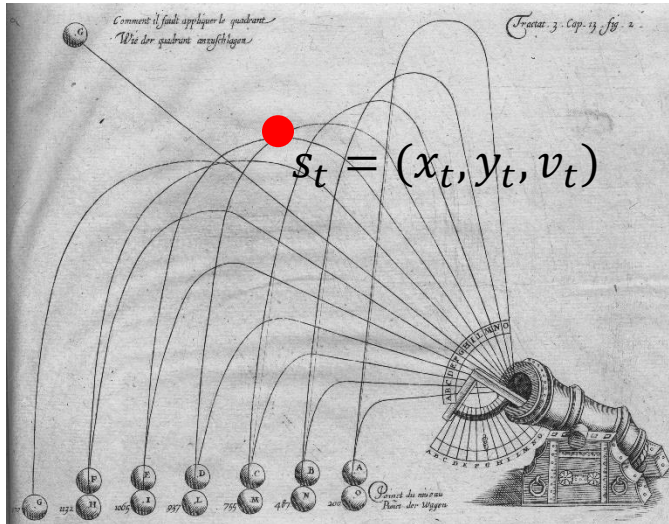


- taking an action might lead to any one of many possible states
- how we can even hope to act optimally in the face of randomness?

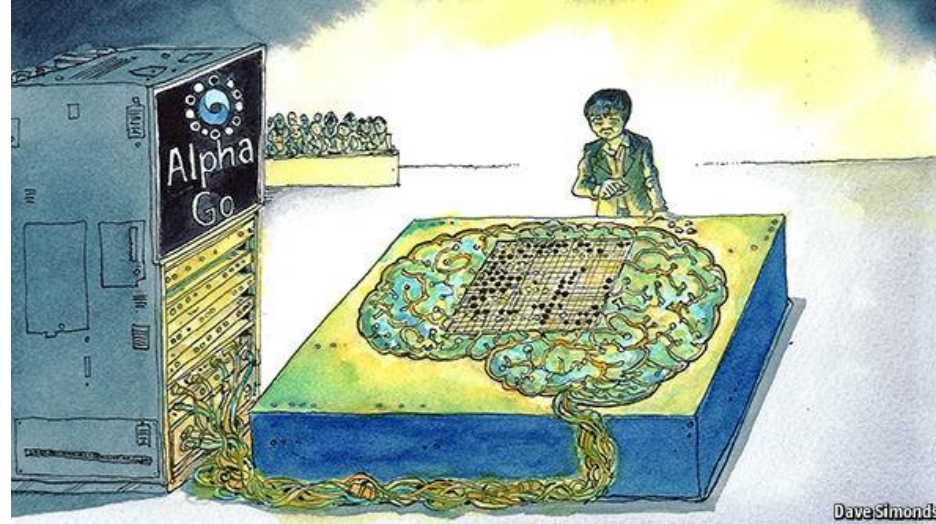
Definition of Markov Decision Process

State and Markov Property

- The state is constructed and maintained on the basis of **immediate sensations** together with **the previous state or some other memory of past sensations**
- Ideal state signal summarizes past sensations compactly, yet in such a way that all relevant information is retained
- A state signal that succeeds in retaining all relevant information is said to be **Markov**



s_t = the location and velocity of the flying canon



s_t = the configuration of the black and white stones

The Markov Property

- “Markov” generally means that given the present state, the future and the past are independent



Andrey Markov (1856-1922)

- For Markov decision processes, “Markov” means action outcomes depend only on the current state

$$\begin{aligned} P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \dots, S_0 = s_0, A_0 = a_0) \\ = P(S_{t+1} = s' | S_t = s_t, A_t = a_t) \end{aligned}$$

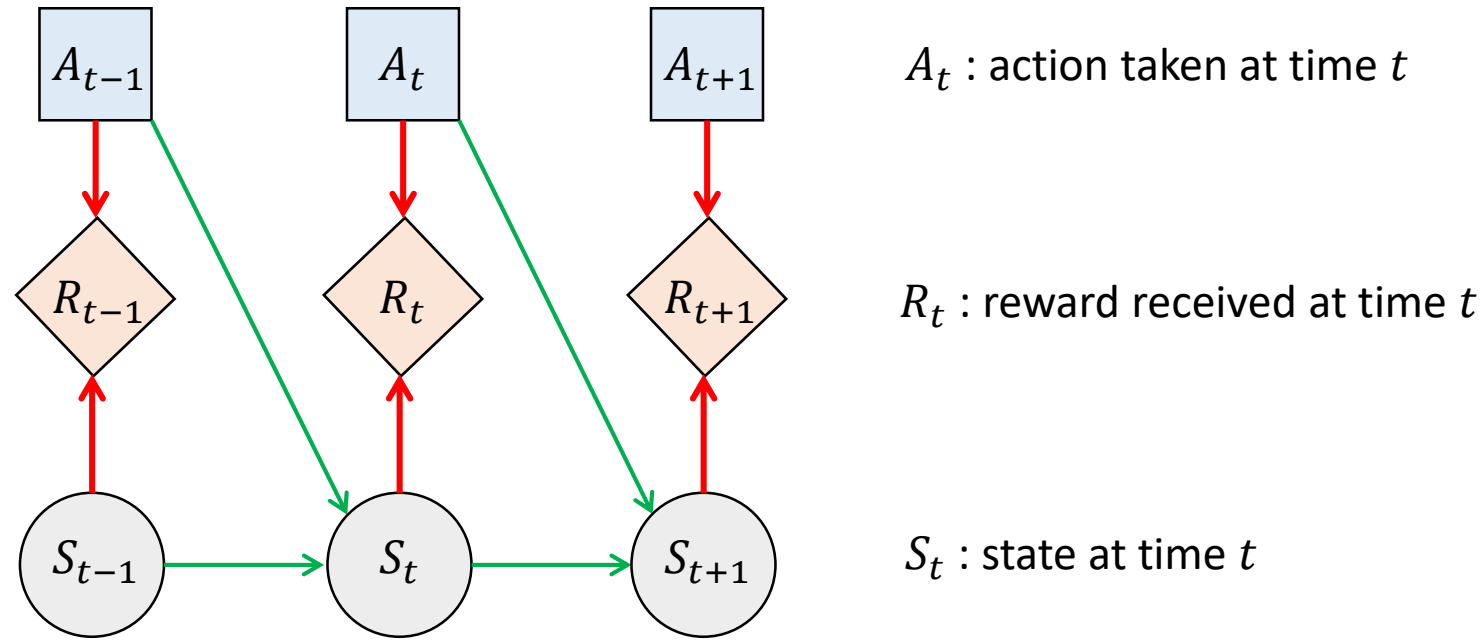
- The best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete history

$$\pi^*(s_t, s_{t-1}, \dots, s_0) = \pi^*(s_t)$$

Note:

- As states become more Markovian, the better performance in MDP solution
- It is useful to think of the state at each time step as an approximation to a Markov value

Markov Decision Processes



- **Transition probability** $T_t(s_t, a_t, s_{t+1}) = P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t) = P(s_{t+1} | s_t, a_t)$
 - ✓ represents the probability of transitioning from state s_t to s_{t+1} after executing action s_t at time t (**Markov assumption**)
- **Reward function:** $r_t = R_t(s_t, a_t)$: represents the probability of receiving reward r_t when executing action a_t from state s_t at time t

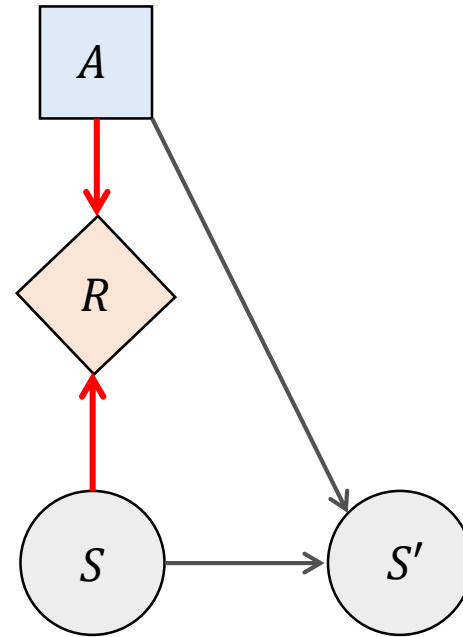
The Agent-Environment Interface

- The time steps $t = 0, 1, \dots$ need not refer to fixed interval of real time:
 - ✓ they can refer to arbitrary successive stages of decision-making and action
- The **actions** can be
 - ✓ Low-level controls, such as the voltages applied to motors of a robot arm
 - ✓ High-level decisions, such as whether or not to go graduate school
- The **states** can take a wide variety of forms
 - ✓ Can be completely determined by low-level sensations, such as sensor readings
 - ✓ Can be high-level and abstract, such as image or mental status
- The **reward** is a consequence of taking an action given a state
 - ✓ Can be specified according to the target tasks
 - ✓ Maximizing reward should result in achieving the goals of a task

In summary,

Actions can be any decisions we want to learn how to make to affect rewards, and the states can be anything we can know that might be useful in making them

(Stationary) Markov Decision Processes

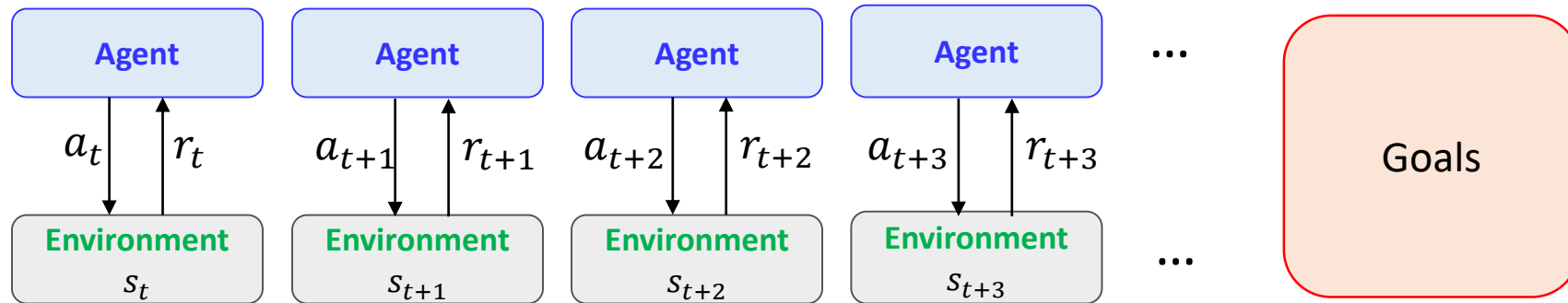


Stationary Markov Decision Process (MDP)

→ transition and reward models are **stationary**

- Transition probability $T(s_t, a_t, s_{t+1})$ are the same for all time step t
- Reward function: $r_t = R(s_t, a_t)$ are the same for all time step t

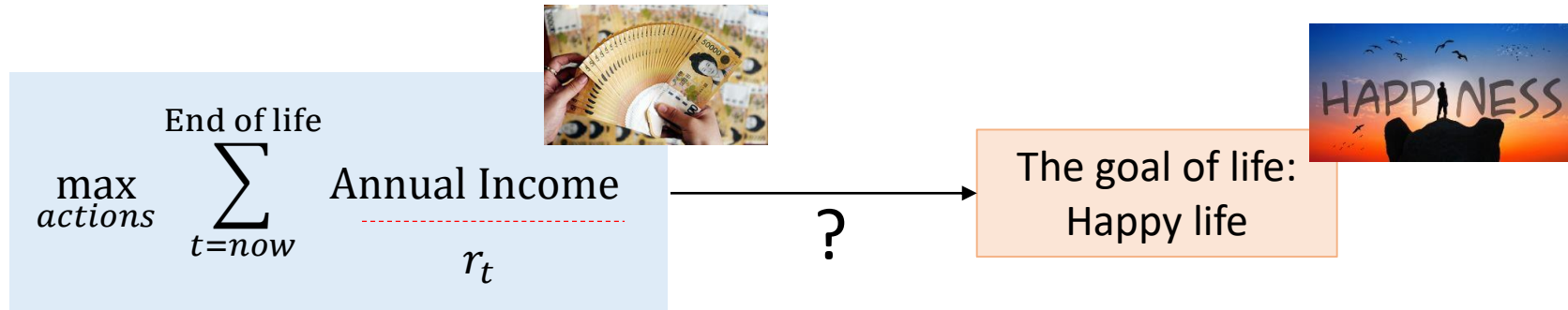
Goals and Rewards



- The agent's goal is to maximize the total amount of reward (cumulative reward) it receives.

$$\max_{a_t, a_{t+1}, \dots} \sum_k r_{t+k}$$

- Rewards we setup truly indicate what we want accomplished
- The reward signal is your way of communicating to the agent **what you want it to achieve**, not how you want it achieved



Utility (returns)

How to formally define the **accumulated reward** in the long run?

- Denote reward : $r_t = R(S_t, A_t)$

- In the episodic tasks, the simplest utility is defined as

$$U_t = r_t + r_{t+1} + r_{t+2} + \dots + r_T = \sum_{k=0}^T r_{t+k}$$

- ✓ T is a final time step associated with the terminal time step T
- ✓ Examples include, maze, go, chess, etc.

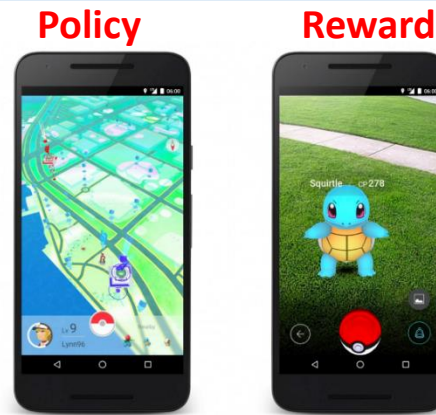
- In the continuing tasks, the discounted utility is defined as

$$U_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

- ✓ γ is a parameter, $0 \leq \gamma \leq 1$, called discount rate
- ✓ Examples include, maze game, Go, Chess, etc.
- ✓ $\gamma = 0$ (myopic): concern only with maximizing immediate rewards
- ✓ $\gamma = 1$ (farsighted): the objective takes future rewards take into account more strongly

- A policy π gives an action $a \in \mathcal{A}$ for each state $s \in \mathcal{S}$:

$$\pi: \mathcal{S} \rightarrow \mathcal{A}$$



- An optimal policy** π^* is one that maximizes expected utility if followed:

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}[U^{\pi}(s)] \text{ for all } s$$

$$\text{where } \mathbb{E}[U^{\pi}(s)] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

- An optimal policy** can be either deterministic or stochastic

deterministic

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

Stochastic

$$p(a|s) = \pi^*(s, a)$$

Take action a^* with a probability one

Take action a with a probability $p(a|s)$

We will exclusively consider deterministic policy

Summary: Markov Decision Processes

Finite Markov Decision Process (MDP) :The state and action space are finite

An MDP is defined by:

- A set of states $s \in \mathcal{S}$
- A set of actions $a \in \mathcal{A}$
- A transition function $T(s, a, s') = P(S_{t+1} = s' | S_t = s, A_t = a) = P(s' | s, a)$
 - ✓ Probability that a from s leads to s' when taking action a
 - ✓ Also called the model or the dynamics
- A reward function $R(s, a, s')$
 - ✓ $r_t = R(s_t, a_t, s_{t+1})$ or $r_{t+1} = R(s_t, a_t)$
 - ✓ If stochastic, $R(s, a, s') = \mathbb{E}[r_t + r_{t+1}, \dots | S_t = s, A_t = a, S_{t+1} = s']$
- A start state $s_0 \in \mathcal{S}$
- A terminal state $s_T \in \mathcal{S}^+$ (for episodic tasks)

Goal of MDP is to find the optimal policy π^* that maps the current state to the best action

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

Dynamic Programming Approach

Value Function & Q function

Value function (**state value function for π**)

“How good it is for the agent to be in a given state”

$V^\pi(s)$: The expected utility received by following policy π from state s

$$V^\pi(s) = \mathbb{E}_\pi(U_t | S_t = s) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s\right)$$

\mathbb{E}_π : not expectation over policy π but all stochastic state transitions associated with π

Q-function (**action-value function for π**)

“How good it is for the agent to perform a given action in a given state”

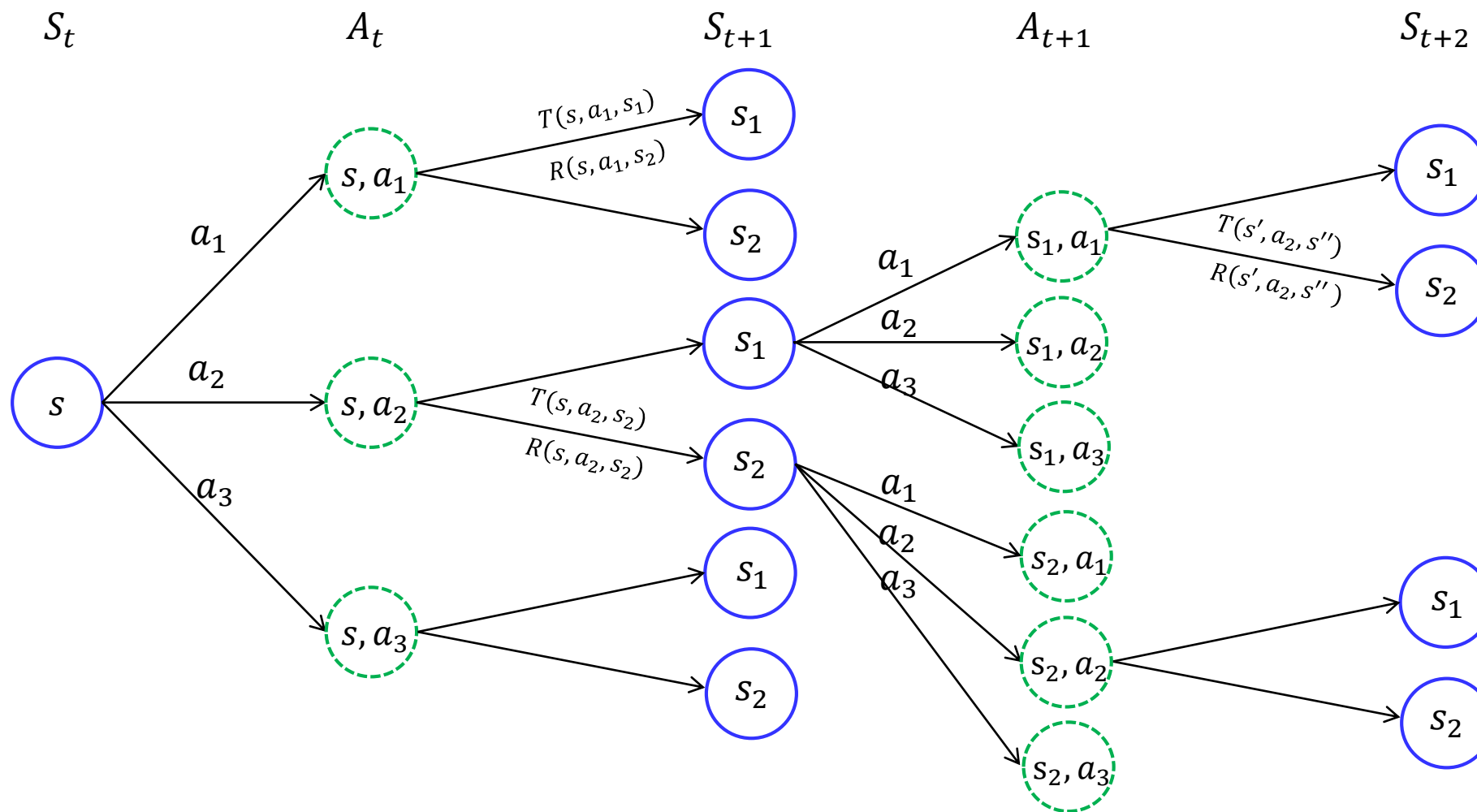
$Q^\pi(s, a)$: The expected utility of taking action a from state s , and then following policy π

$$Q^\pi(s, a) = \mathbb{E}_\pi(U_t | S_t = s, A_t = a) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s, A_t = a\right)$$

Because the agent can expect to receive in the future depend on what actions it will take
→ Value and Q functions are defined with respect to a particular policy mapping state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$

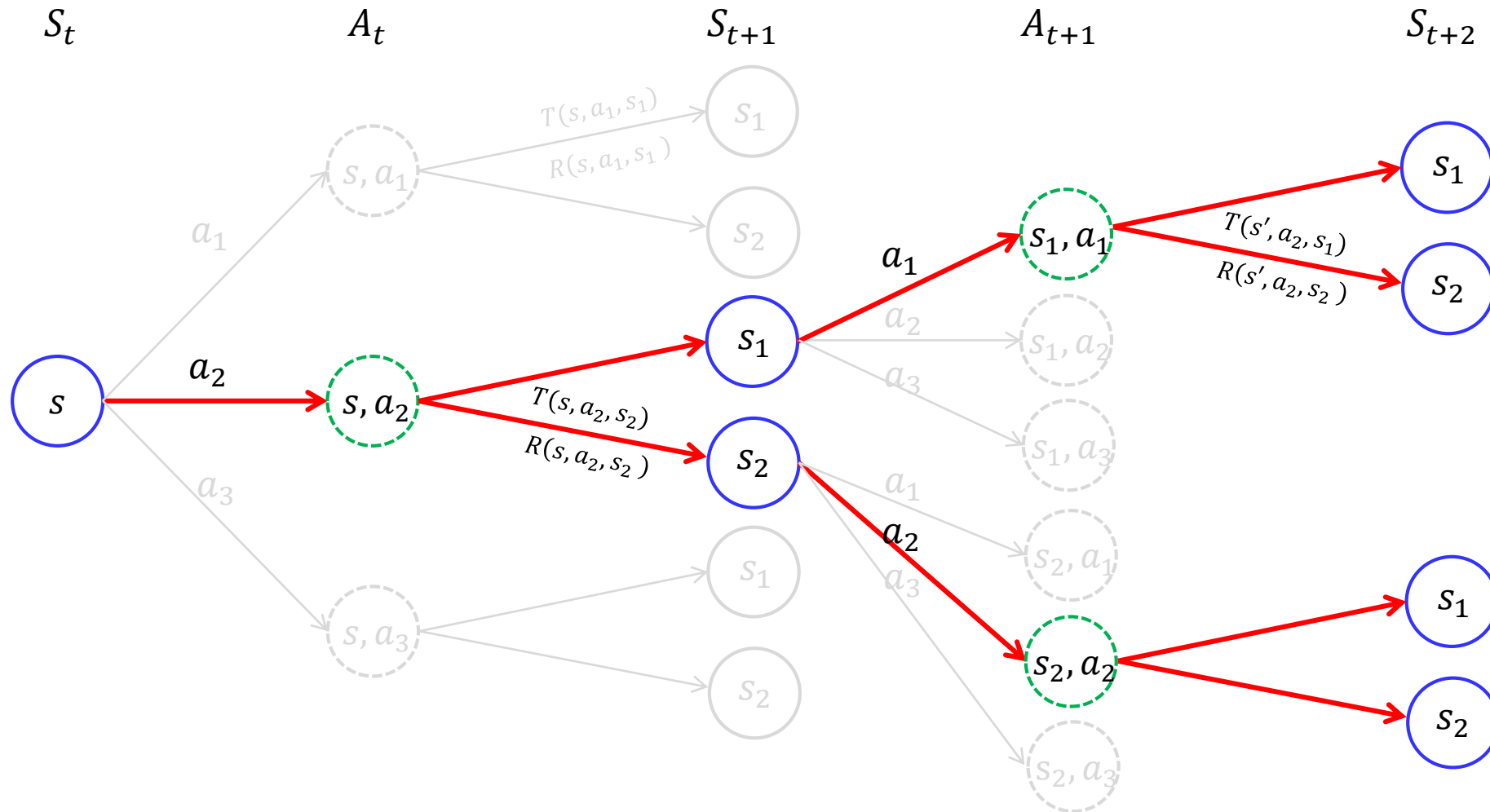
Value Function

All possible trajectories



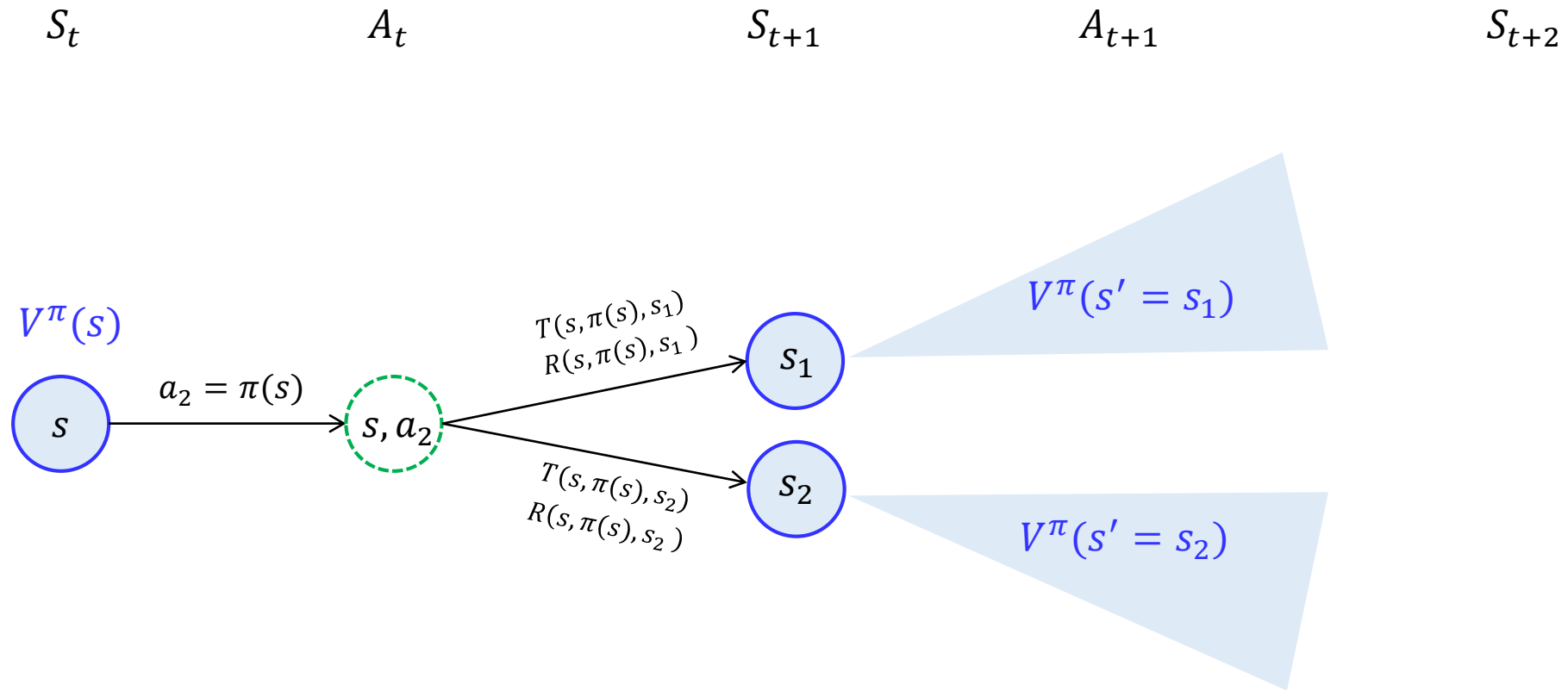
Value Function

A policy π is given as: $\pi(s) = a_2$; $\pi(s_1) = a_1$; $\pi(s_2) = a_2$



Value Function

A policy π is given as: $\pi(s) = a_2$; $\pi(s_1) = a_1$; $\pi(s_2) = a_2$



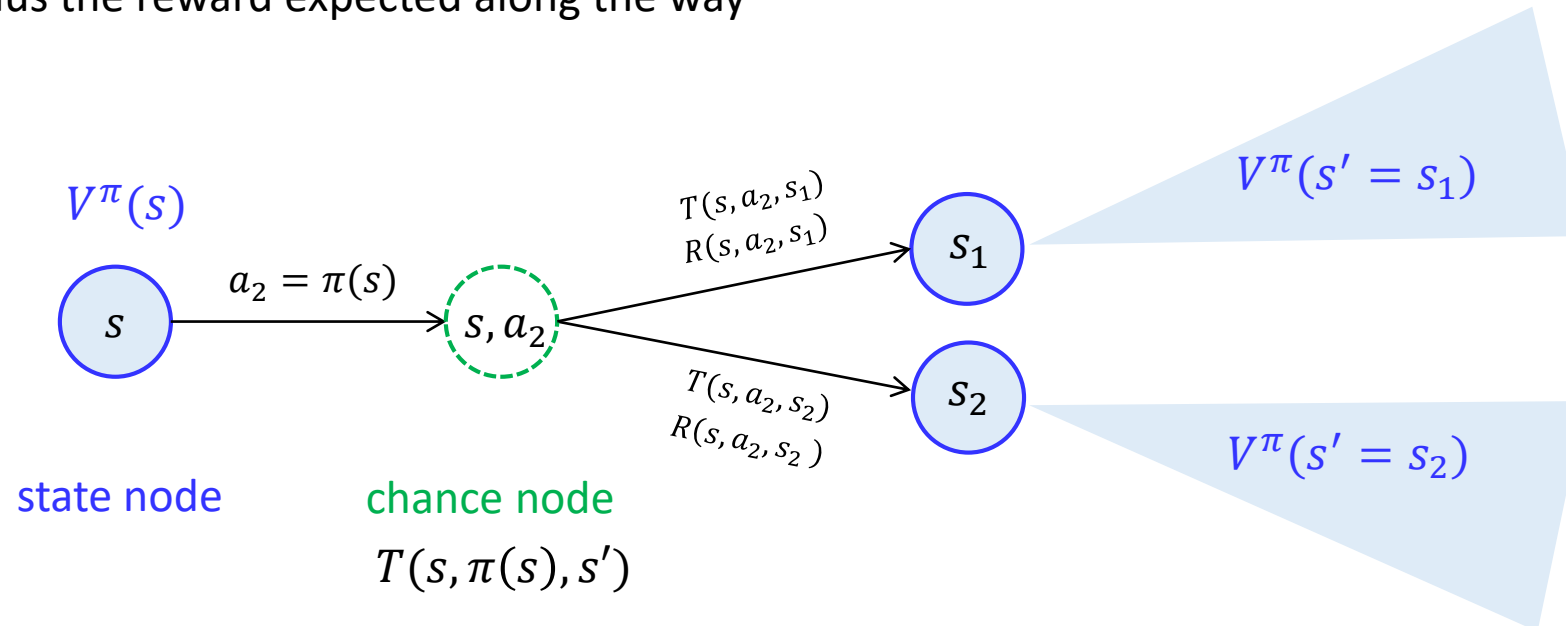
$$V^\pi(s) = T(s, \pi(s), \mathbf{s}_1) \{ R(s, \pi(s), s_1) + \gamma V^\pi(s_1) \} + \\ T(s, \pi(s), s_2) \{ R(s, \pi(s), s_2) + \gamma V^\pi(\mathbf{s}_2) \}$$

The Bellman Equation for Value Function

Recursive Formulation (The Bellman equation)

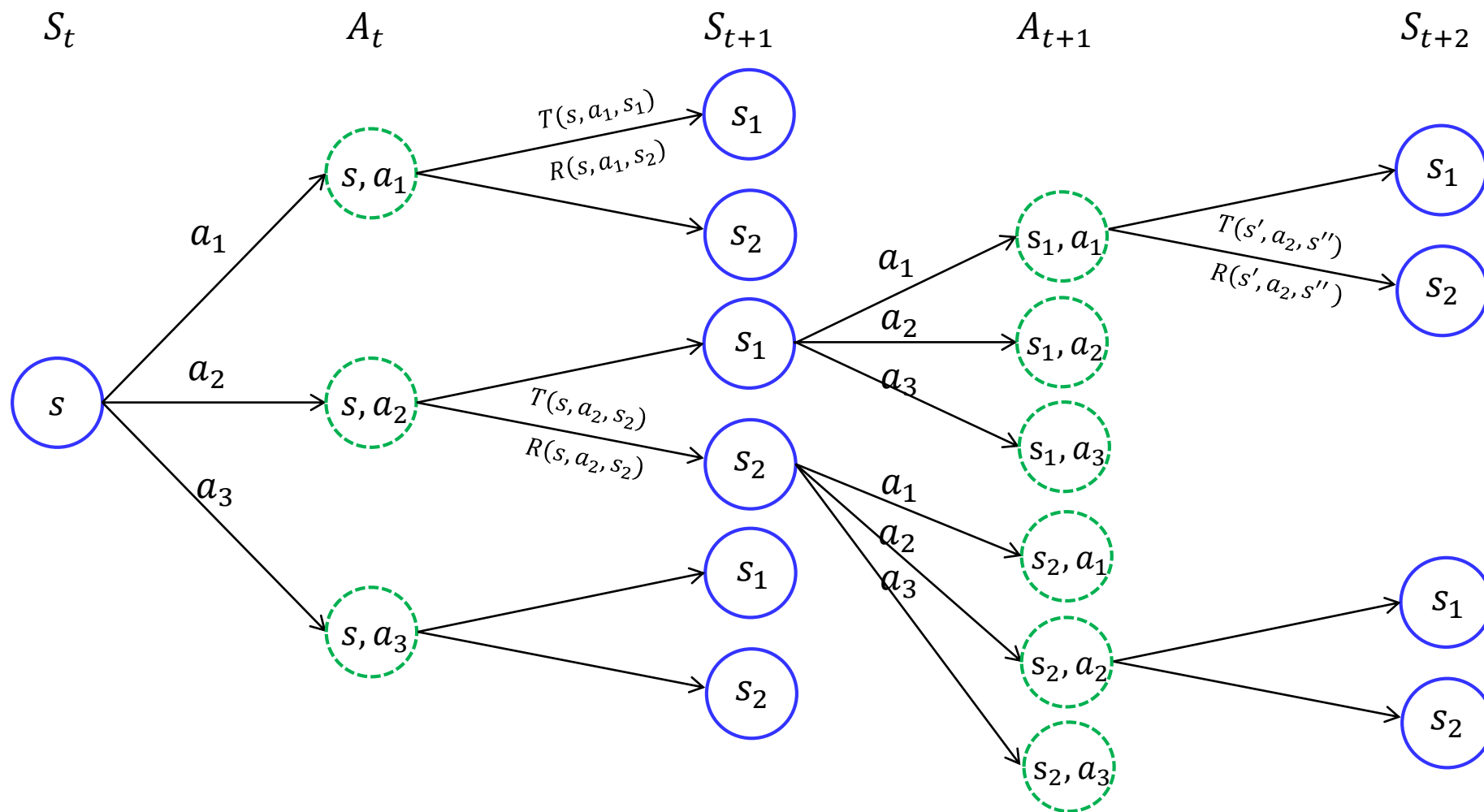
$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s \right) \\ &= \mathbb{E}_\pi \left(r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid S_t = s \right) \\ &= \sum_{s'} T(s, \pi(s), s') \{ R(s, \pi(s), s') + \gamma \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid S_{t+1} = s' \right) \} \\ &= \sum_{s'} T(s, \pi(s), s') \{ R(s, \pi(s), s') + \gamma V^\pi(s') \} \end{aligned}$$

The value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way



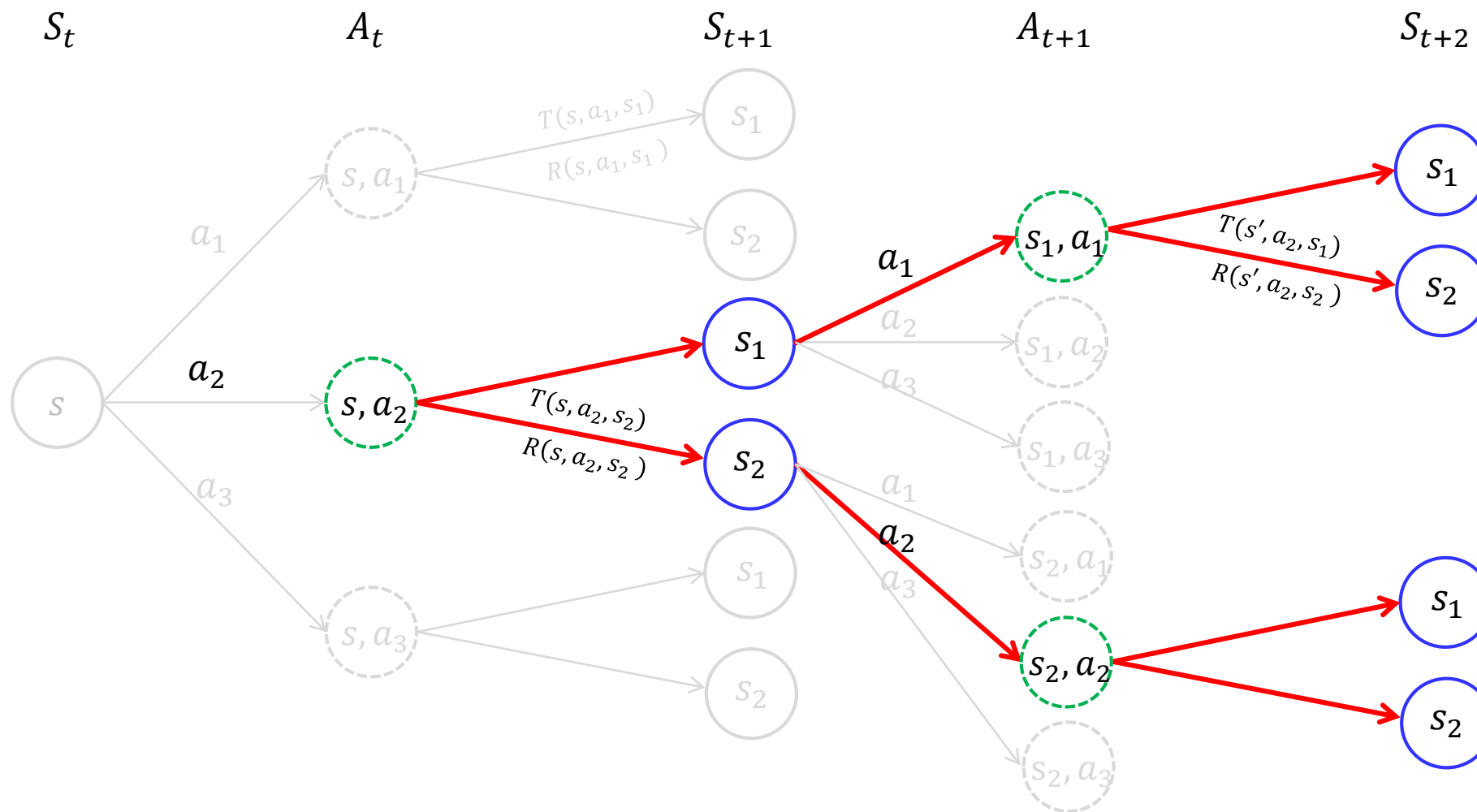
Q function

All possible trajectories



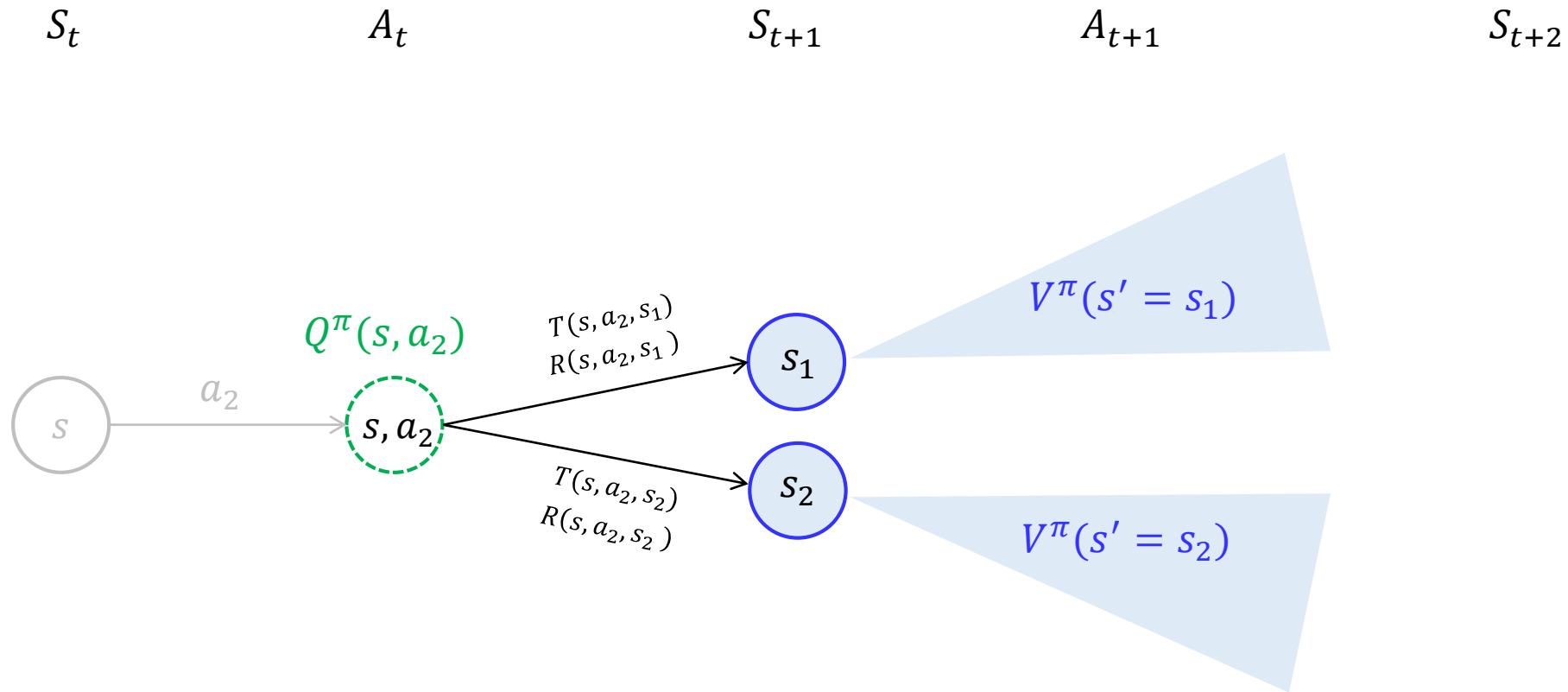
Q function

A policy π is given as: $\pi(s) = a_2$; $\pi(s_1) = a_1$; $\pi(s_2) = a_2$



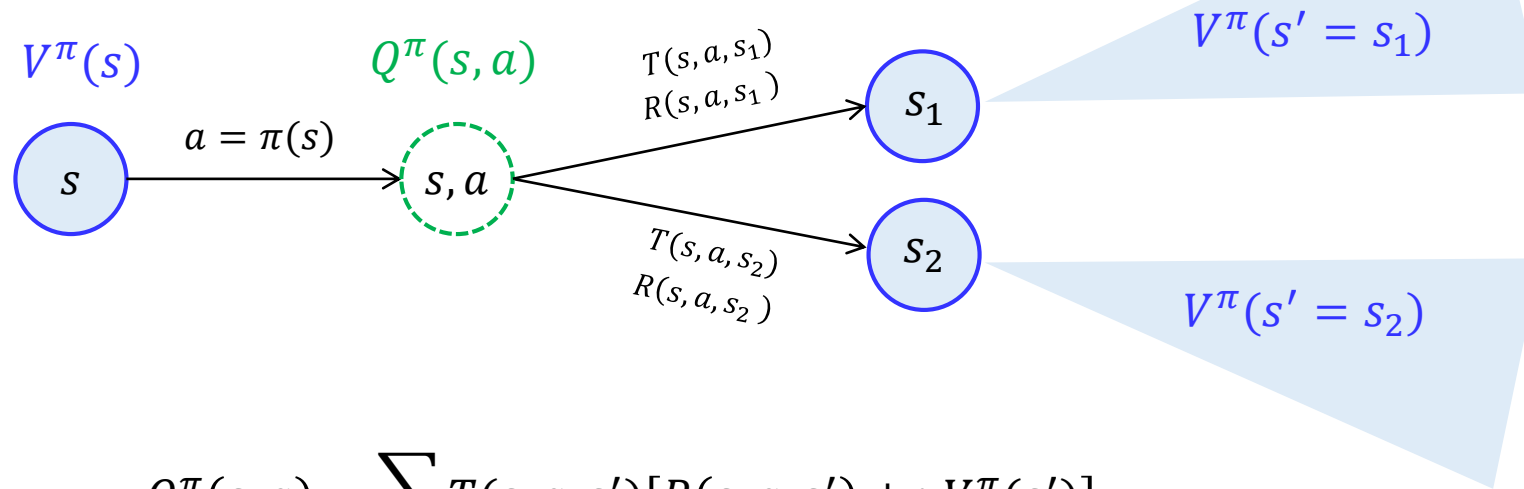
Q Function

A policy π is given as: $\pi(s) = a_2$; $\pi(s') = a_1$; $\pi(s'') = a_2$



$$Q^\pi(s, a_2) = T(s, a_2, s_1)\{R(s, a_2, s_1) + \gamma V^\pi(s_1)\} + T(s, a_2, s_2)\{R(s, a_2, s_2) + \gamma V^\pi(s_2)\}$$

Q Function



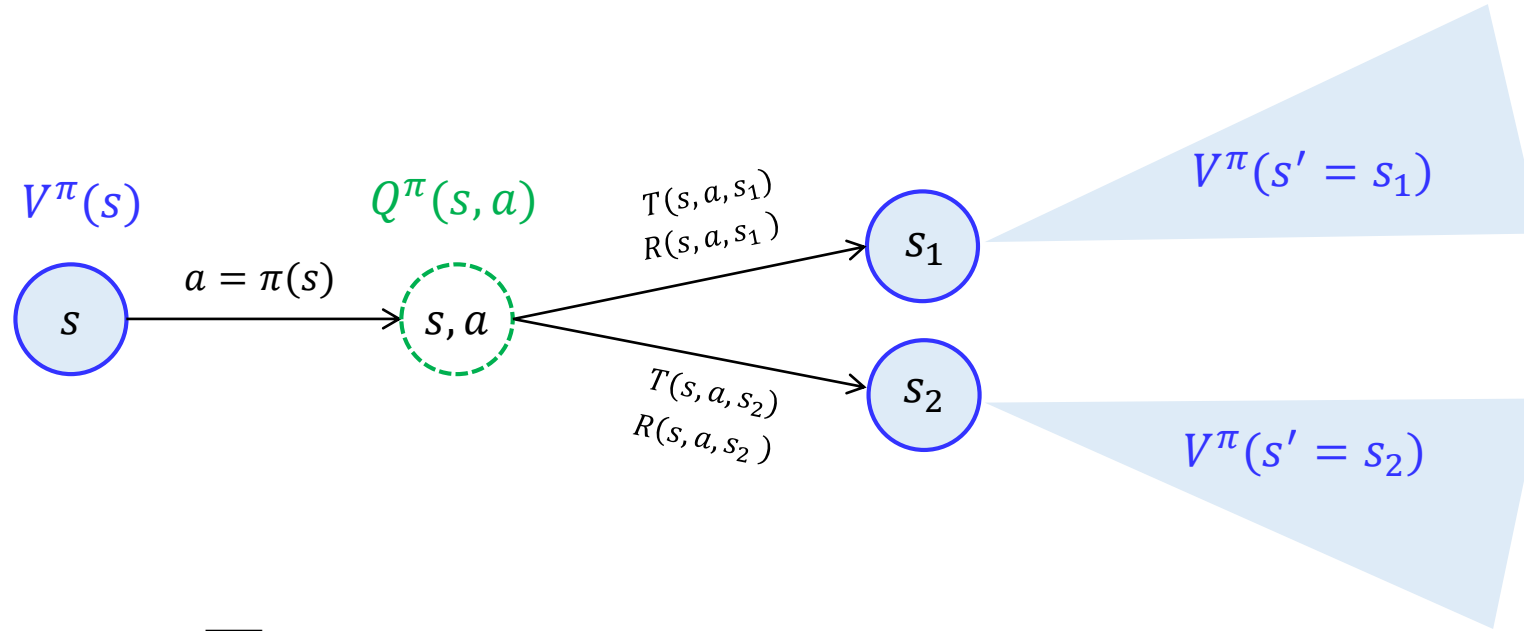
$$Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$$

Note that:

$$\begin{aligned} V^\pi(s) &= \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V^\pi(s')\} \\ &= Q^\pi(s, \pi(s)) \end{aligned}$$

- $Q^\pi(s, a)$ is more general since it has the option to select an action a given state s
- If the action is enforced to select $a = \pi(s)$ according to the policy π , $V^\pi(s) = Q^\pi(s, \pi(s))$

Summary for Value function and Q function



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

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

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

Optimal Value Function & Q function

Optimal policy

$$\pi^* \geq \pi \text{ if and only if } V^{\pi^*}(s) \geq V^\pi(s) \text{ for all } s$$

Optimal state- value function

$$V^*(s) = \max_{\pi} V^\pi(s) \text{ for all } s$$

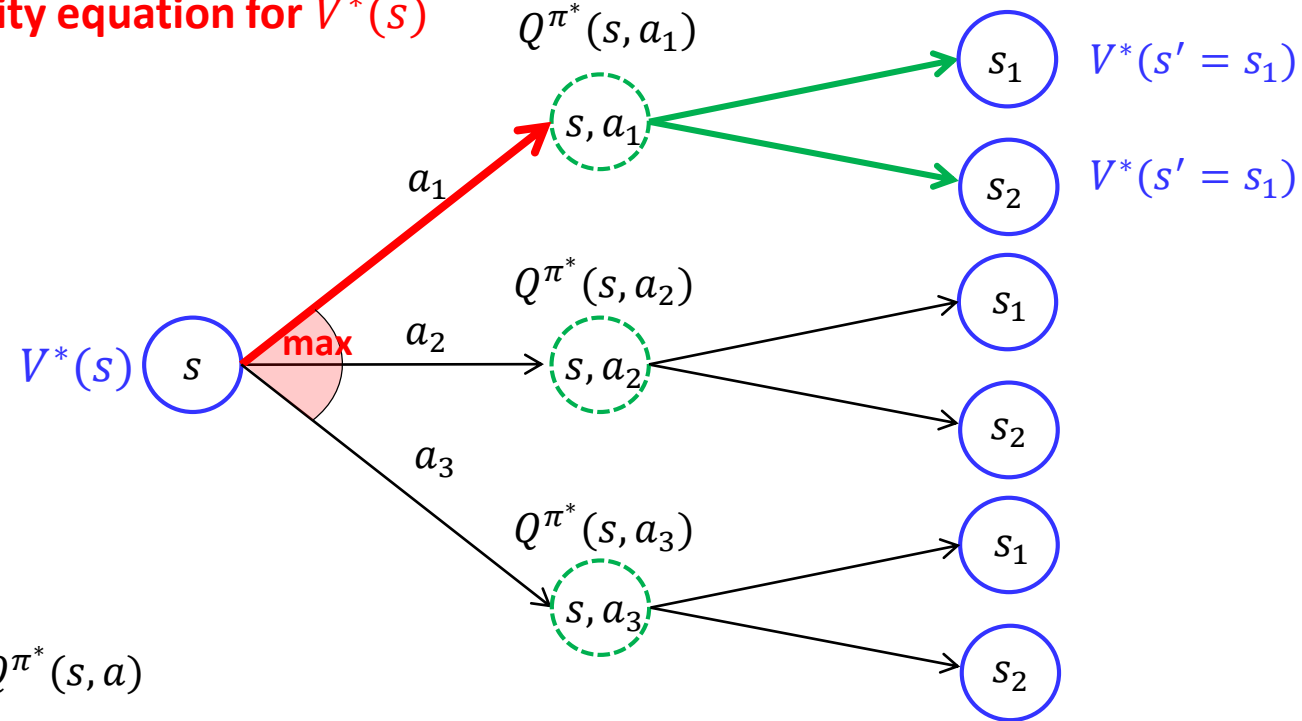
Optimal action-value function

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s)$$

$$\begin{aligned} Q^*(s, a) &= \max_{\pi} Q^\pi(s, a) && \because Q^\pi(s, a) = \mathbb{E}[R(s, a, s') + \gamma V^\pi(s') | s_t = s, a_t = a] \\ &= \max_{\pi} \mathbb{E}[R(s, a, s') + \gamma V^\pi(s') | s_t = s, a_t = a] && \mathbb{E} \text{ is over the next state } s' \\ &= \mathbb{E} \left[R(s, a, s') + \gamma \max_{\pi} V^\pi(s') \mid s_t = s, a_t = a \right] \\ &= \mathbb{E}[R(s, a, s') + \gamma V^*(s') | s_t = s, a_t = a] \end{aligned}$$

Bellman Optimality Equation for State-Value Function

Bellman optimality equation for $V^*(s)$



$$V^*(s) = \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s, a)$$

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi^*} \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s, A_t = a \right)$$

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi^*} \left(r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s, A_t = a \right) \quad \mathbb{E} \text{ is over transitions associated with } \pi^*$$

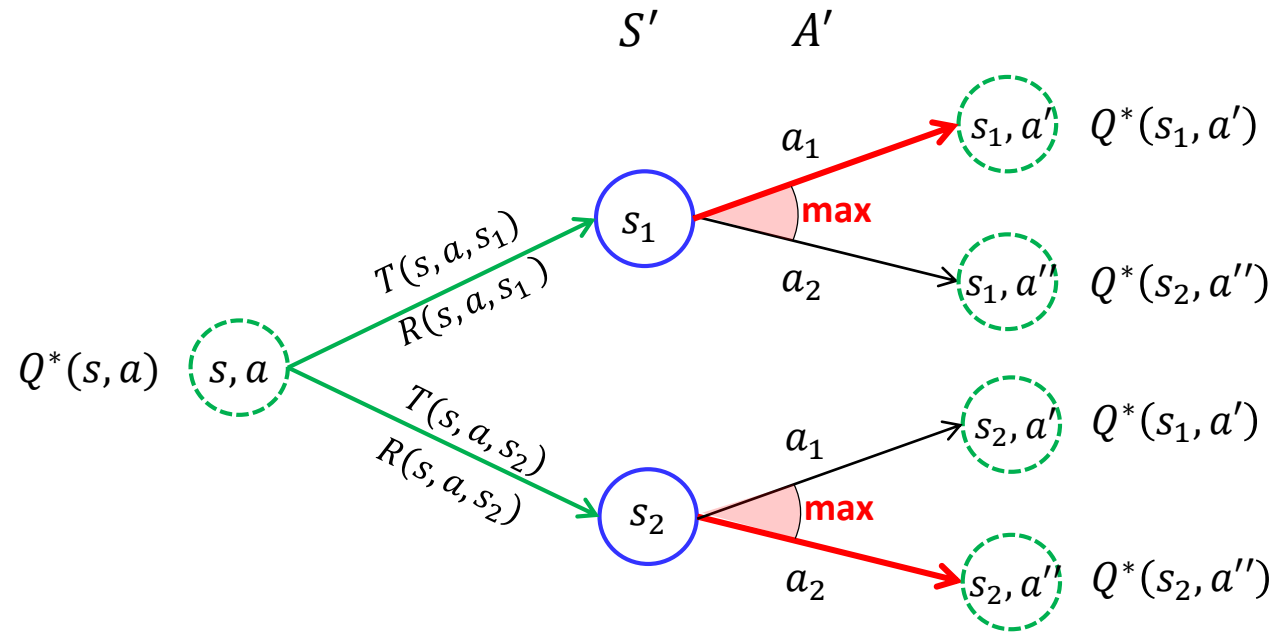
$$= \max_{a \in \mathcal{A}(s)} \mathbb{E} (r_t + \gamma V^*(s_{t+1}) \mid S_t = s, A_t = a) \quad \mathbb{E} \text{ is over transitions}$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

First take optimum action and follow the optimum policy

Bellman Optimality Equation for State-Action Value Function

Bellman optimality equation for $Q^*(s, a)$



$$\begin{aligned} Q^*(s, a) &= \mathbb{E} \left\{ r_t + \gamma \max_{a'} Q^*(s', a') \mid s_t = s, a_t = a \right\} \\ &= \sum_{s'} T(s, a, s') \left\{ R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right\} \end{aligned}$$

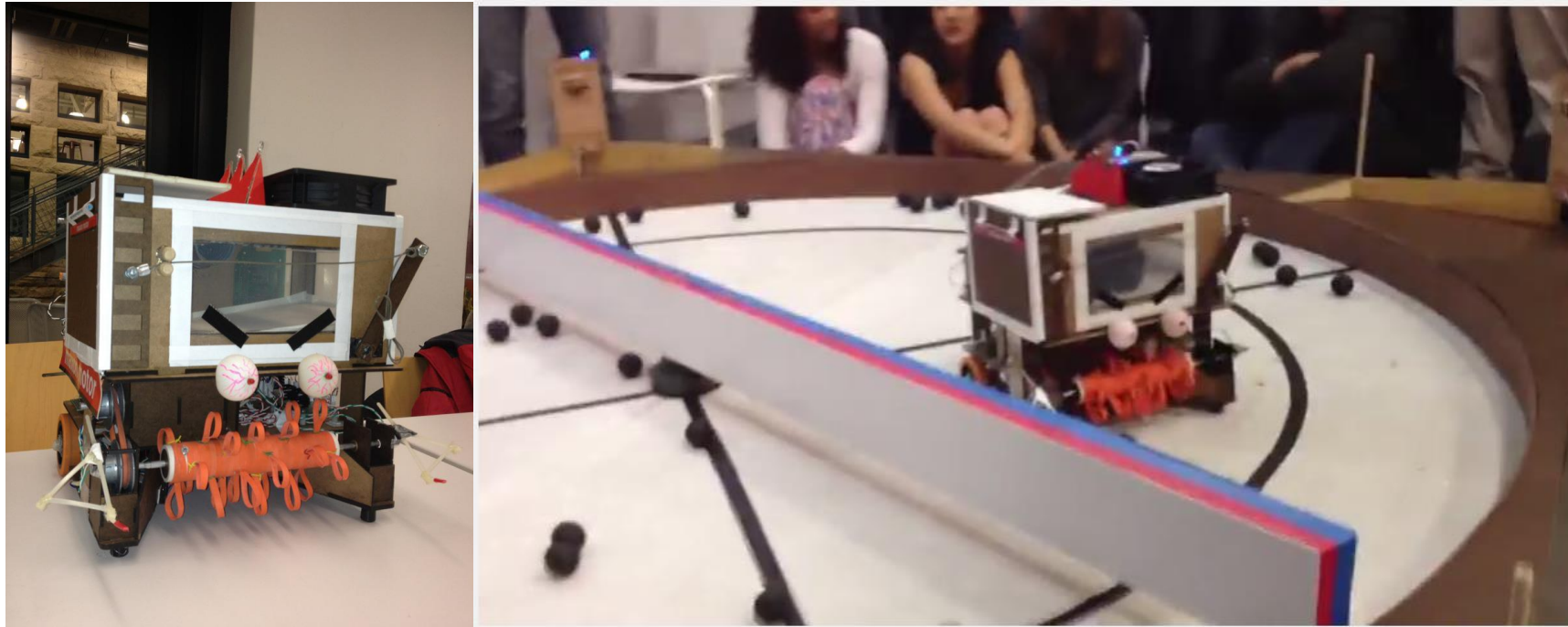
\mathbb{E} is over transitions $s' \rightarrow s'$

First transits by transition probability and take the optimum action for each consequent states

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

- The Bellman optimality equation is a system of equations, one for each state
 - ✓ For N states, N unknown and N equations
 - ✓ With known $T(s, a, s')$ and $R(s, a, s')$, the equations can be solved using various mathematical programming (i.e., Linear programming, Quadratic programming)

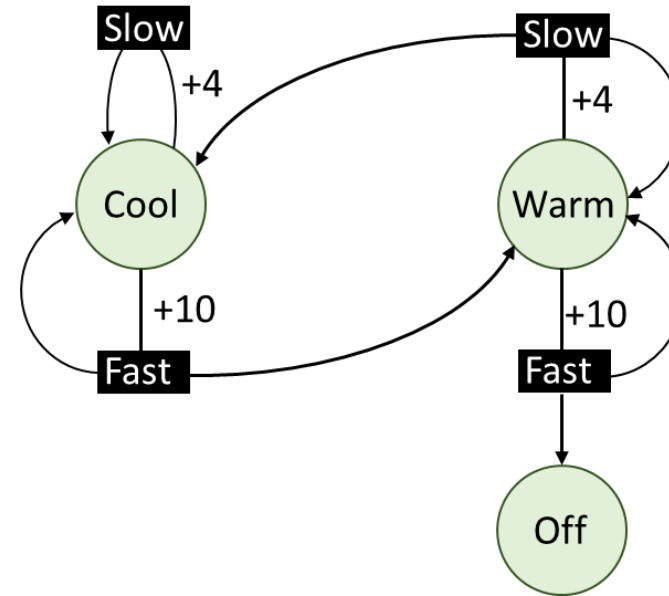
Robot Cleaner



<https://youtu.be/bFTvQt3Sj3Y?t=66>

Optimum Planning as a Greedy Search

s	a	s'	$T(s, a, s')$
cool	slow	cool	1
cool	fast	cool	1/2
cool	fast	warm	1/2
warm	slow	cool	1/2
warm	slow	warm	1/2
warm	fast	warm	1/2
warm	fast	off	1/2



$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

$$V^*(\text{cool}) = \max \left\{ \begin{array}{l} 1(4 + 0.9V^*(\text{cool})) \\ 0.5(10 + 0.9V^*(\text{cool})) + 0.5(10 + 0.9W) \end{array} \right\}$$

$$V^*(\text{warm}) = \max \left\{ \begin{array}{l} 0.5(4 + 0.9V^*(\text{cool})) + 0.5(4 + 0.9V^*(\text{warm})), \\ 0.5(10 + 0.9V^*(\text{warm})) + 0.5(10 + 0.9V^*(\text{off})) \end{array} \right\}$$

Optimum Planning as a Greedy Search

- **Reconstructing optimal policy with $Q^*(s, a)$ and $V^*(s)$**

$$\begin{aligned} a^* &= \operatorname{argmax}_a Q^*(s, a) \\ &= \operatorname{argmax}_a \mathbb{E}[R(s, a, s') + \gamma V^*(s') | s_t = s, a_t = a] \end{aligned}$$

- Any greedy policy with respect to the optimal value function $V^*(s)$ is an optimal policy
→ because $V^*(s)$ already takes into account the reward consequences of all possible future behavior
- The Q function effectively caches the results of all one-step-ahead search

Dynamic Programming

- The term **dynamic programming (DP)** refers to a collection of algorithms that can be used to compute optimal policies given a **perfect model of the environment** as a Markov decision process (MDP)
- The key idea of DP (and reinforcement learning) is the use of **value functions** to organize and structure the search for good policies
- Optimal policies can be derived from the optimal value functions that satisfy the Bellman optimality equations

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\} \\ Q^*(s, a) &= \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma \max_{a'} Q^*(s', a')\} \\ &= \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\} \quad \because V^*(s') = \max_{a'} Q^*(s', a') \end{aligned}$$



Optimal policy

$$\begin{aligned} \pi^*(s) &= \operatorname{argmax}_a Q^*(s, a) \\ &= \operatorname{argmax}_a \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\} \end{aligned}$$

Policy Evaluation

Policy evaluation :

A method to compute the state-value function $V^\pi(s)$ for an arbitrary policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$

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

➤ A system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknown

Algorithm

Initialize $V_{t=0}^\pi(s) \leftarrow 0$ for all states $s \in \mathcal{S}$

Repeat (iteration $t = 0, \dots$):

For each state s :

$$V_{t+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^\pi(s')\}$$

Until $\max_{s \in \mathcal{S}} |V_{t+1}^\pi(s) - V_t^\pi(s)| \leq e$

Full backup:

Each iteration of iterative policy evaluation backs up the value of every state once to produce the new approximate value function V_{t+1}^π

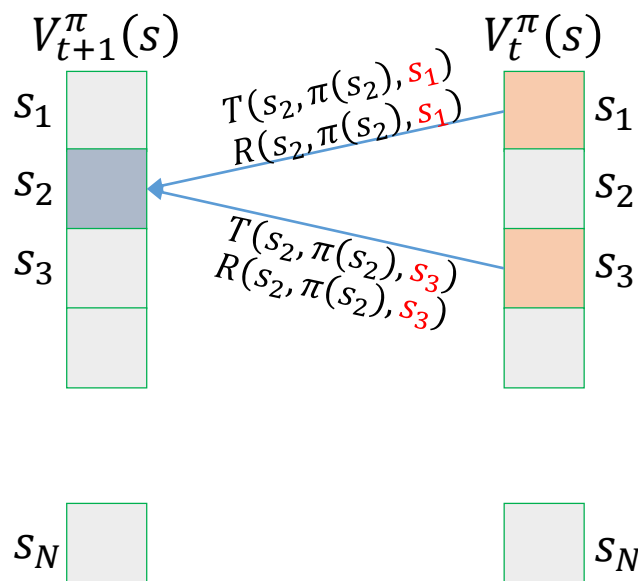
Policy Evaluation

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

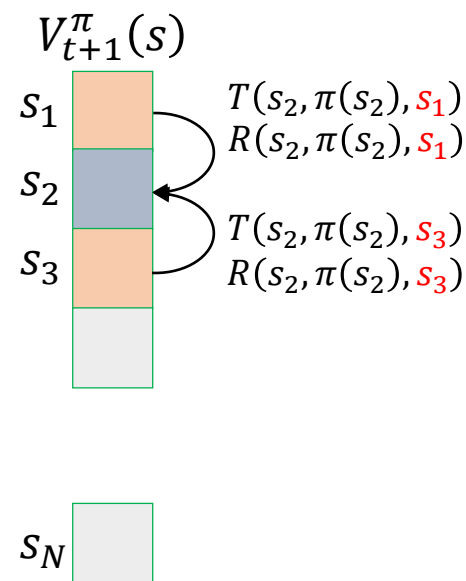
Example:

$$\begin{aligned} V_{t+1}^{\pi}(s_2) &= \sum_{s'} T(s_2, \pi(s_2), s') \{R(s_2, \pi(s_2), s') + \gamma V_t^{\pi}(s')\} \\ &= T(s_2, \pi(s_2), s_1) \{R(s_2, \pi(s_2), s_1) + \gamma V_t^{\pi}(s_1)\} + T(s_2, \pi(s_2), s_3) \{R(s_2, \pi(s_2), s_3) + \gamma V_t^{\pi}(s_3)\} \end{aligned}$$

“Two-arrays” update





“In place” update



Usually faster!
Less memory

Policy Evaluation

Example : Grid world

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

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

- $\mathcal{S} = \{1, 2, \dots, 14\}$
- $\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$
- $T(s, s', a) = \begin{cases} 1, & \text{if move is allowed} \\ 0, & \text{if move is not allowed} \end{cases}$

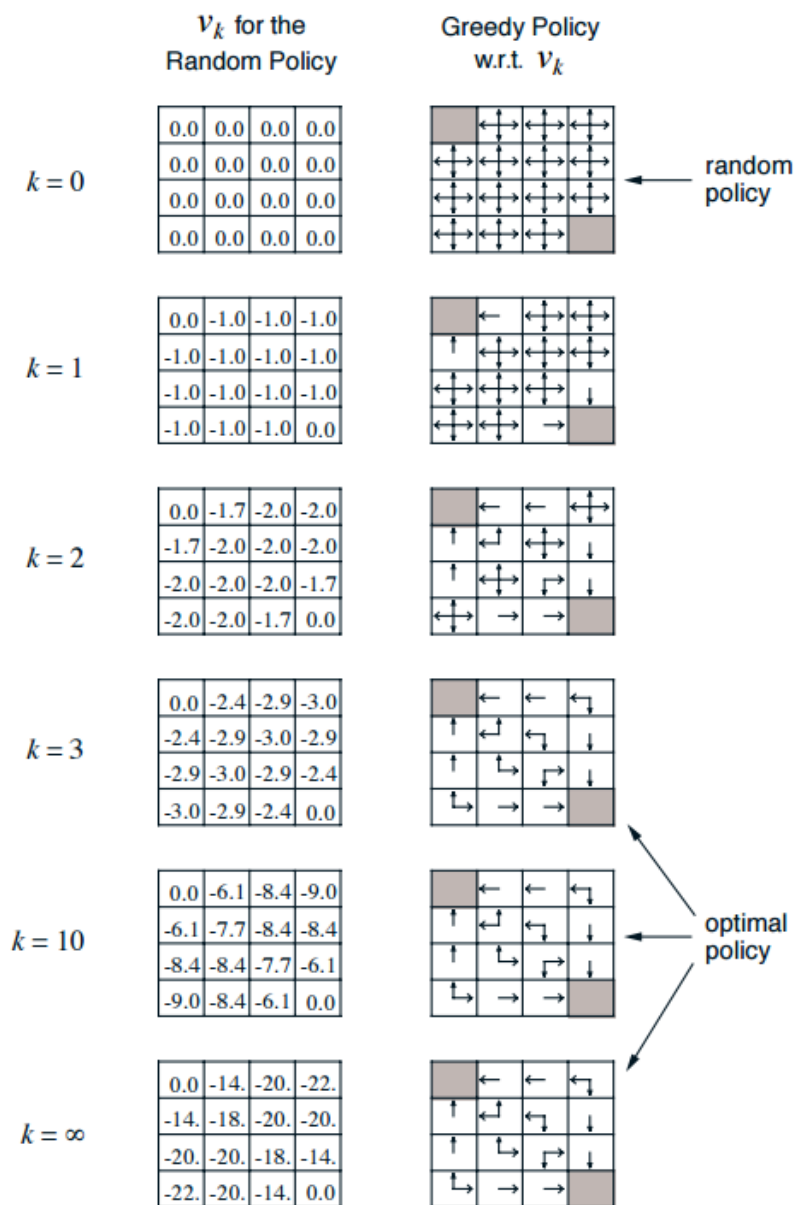
$$T(5, 6, \rightarrow) = 1 \quad T(5, 10, \rightarrow) = 0 \quad T(7, 7, \rightarrow) = 1$$

The actions that would take the agent off the grid leave the state unchanged

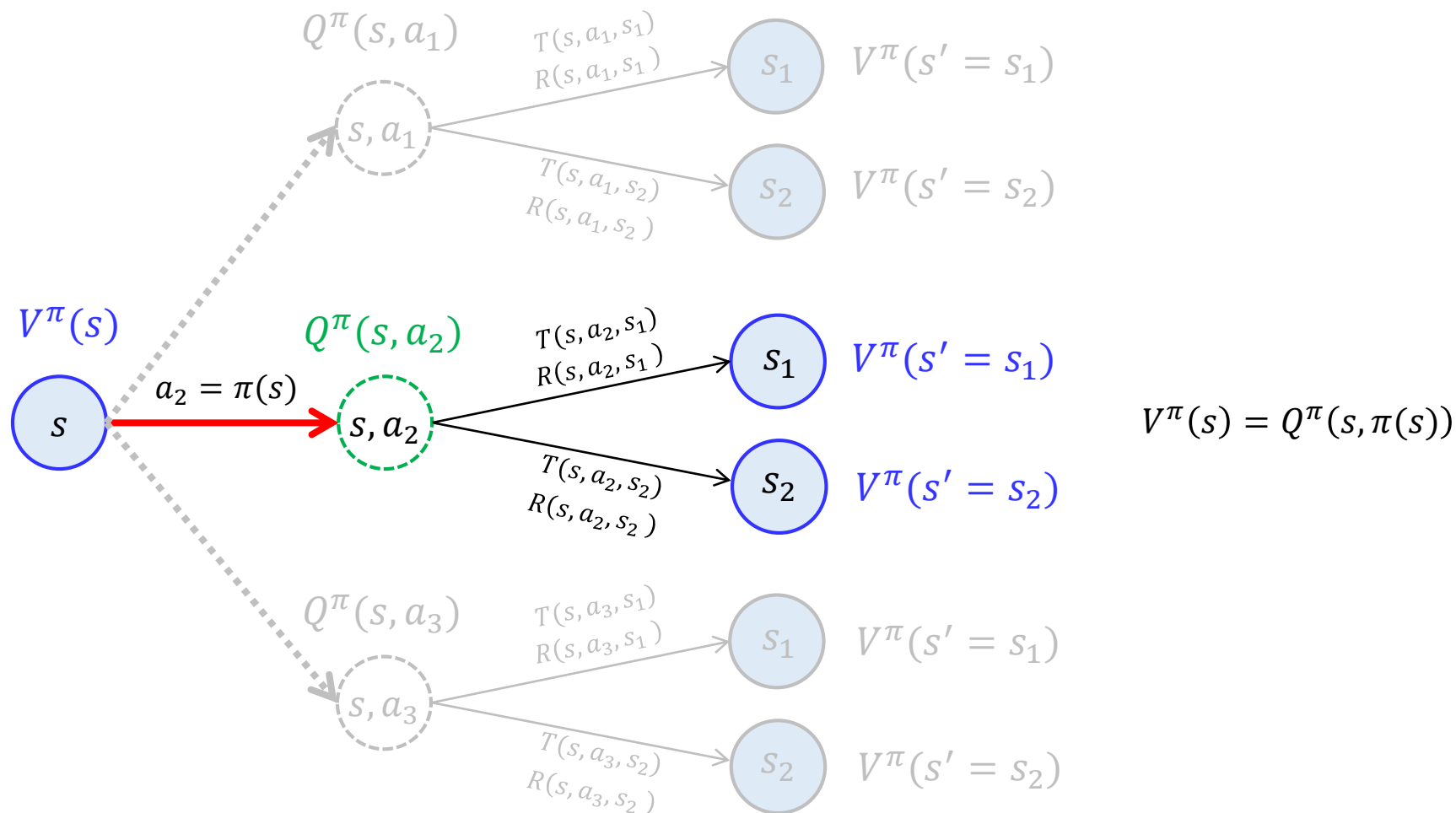
- $R(s, s', a) = -1$ for all s, s', a
- $\gamma = 1$

Suppose the agent follows the equiprobable random policy (all actions equally likely), what is the value function?

Policy Evaluation



Policy Improvement

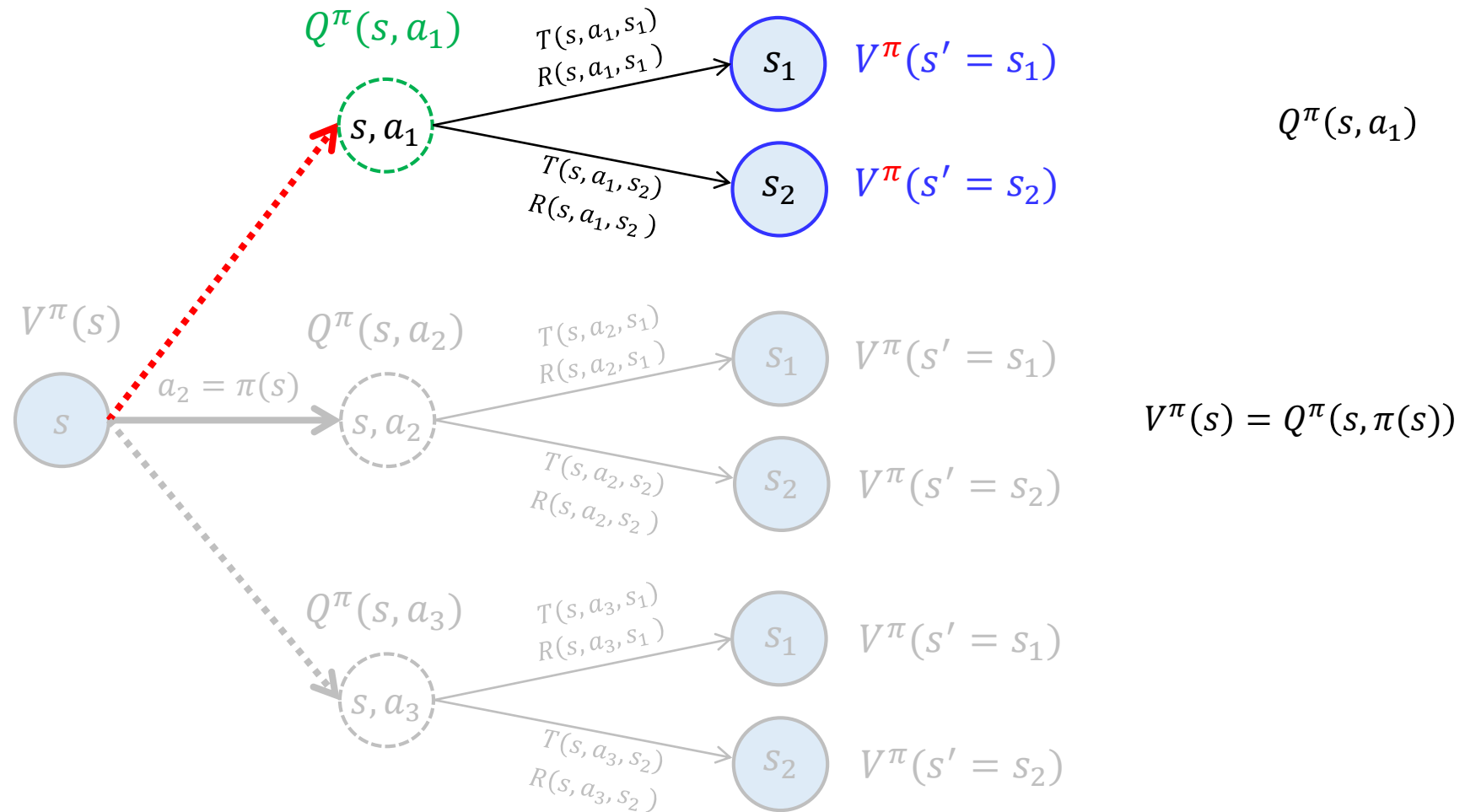


We know how good it is to follow the current policy from s based on $V^\pi(s)$

Would it be better or worse to change to the new policy?

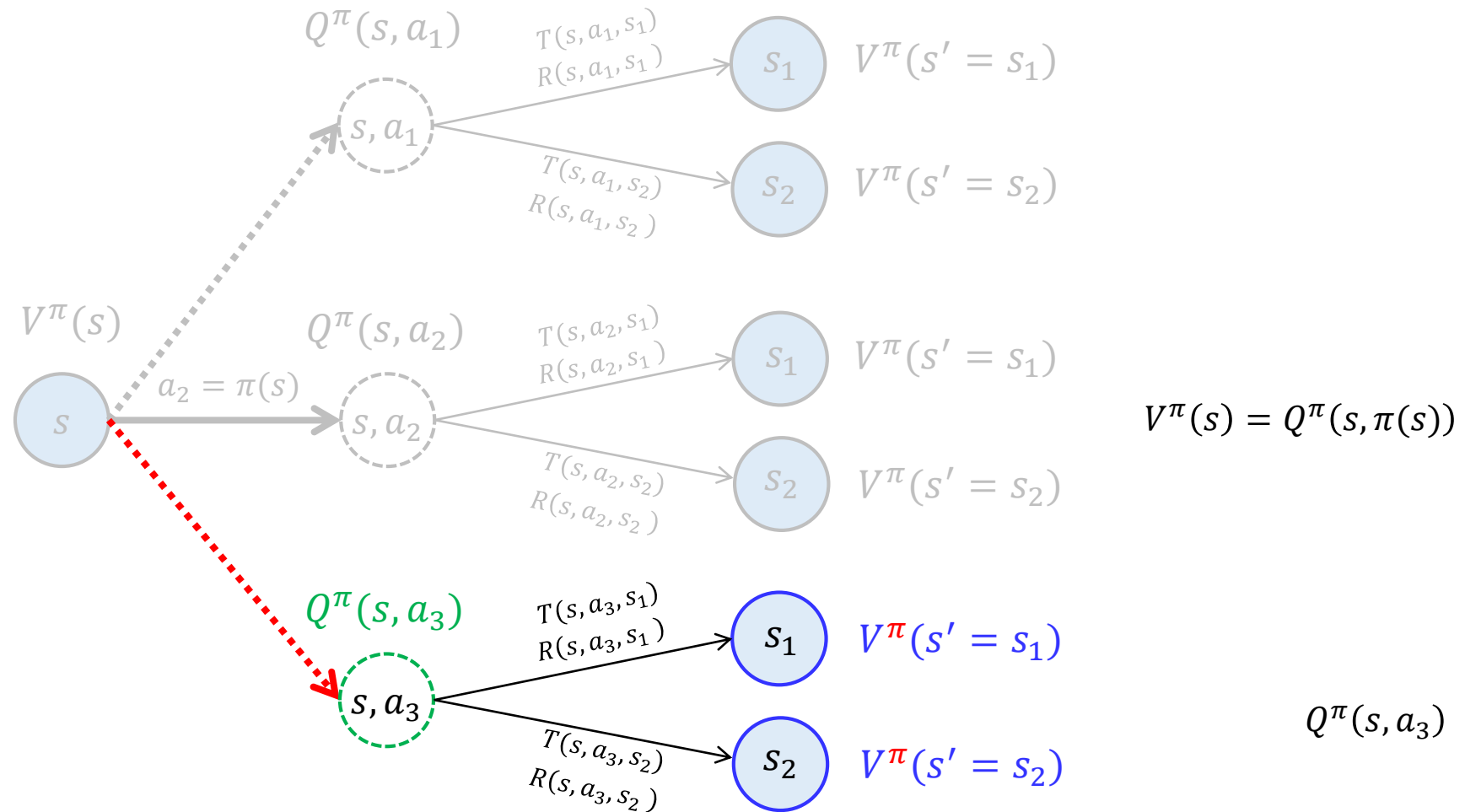
→ Select a given s and thereafter following the existing policy π (a single step change)

Policy Improvement



$$Q^\pi(s, a_1) \geq V^\pi(s) = Q^\pi(s, \pi(s))?$$

Policy Improvement



$$Q^\pi(s, a_3) \geq V^\pi(s) = Q^\pi(s, \pi(s))?$$

Policy Improvement

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

$$\rightarrow Q^\pi(s, \pi'(s)) \geq Q^\pi(s, \pi(s)) = V^\pi(s)$$

$$x^* = \operatorname{argmax}_x f(x)$$

$$\rightarrow f(x^*) \geq f(x) \text{ for all } x$$

Improvement criterion =

Expected reward provided by **changing one step action** and **following the original policy**

If it is better to select $a = \pi'(s)$ once in s and thereafter follow π than it would be to follow π all the time,



It is better still to select $a = \pi'(s)$ whenever s is encountered
(The new policy $\pi'(s)$ is a better policy overall)

Policy Improvement

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

$$\rightarrow Q^\pi(s, \pi'(s)) \geq Q^\pi(s, \pi(s)) = V^\pi(s)$$

$$x^* = \operatorname{argmax}_x f(x)$$

$$\rightarrow f(x^*) \geq f(x) \text{ for all } x$$

Improvement criterion =

Expected reward provided by **changing one step action** and **following the original policy**

Proof (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 $\pi(s) = \pi^*(s)$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \rightarrow \underline{V^{\pi'}(s) \geq V^\pi(s) \text{ for all states } s \in \mathcal{S}}$$

$$\pi' \geq \pi$$

Policy Improvement

Proof (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 $\pi(s) = \pi^*(s)$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \rightarrow V^{\pi'}(s) \geq V^\pi(s) \text{ for all states } s \in \mathcal{S}$$

$$V^\pi(s) \leq Q^\pi(s, \pi'(s))$$

Given

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s]$$

$\mathbb{E}_{\pi'}$ is expectation over s_{t+1} induced by π'

$$\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s] \quad \because V^\pi(s_{t+1}) \leq Q^\pi(s_{t+1}, \pi'(s_{t+1}))$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \mathbb{E}_{\pi'}[r_{t+2} + \gamma V^\pi(s_{t+2})] | s_t = s]$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) | s_t = s]$$

$$\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q^\pi(s_{t+2}, \pi'(s_{t+2})) | s_t = s]$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 \mathbb{E}_{\pi'}[r_{t+3} + \gamma V^\pi(s_{t+3})] | s_t = s]$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) | s_t = s]$$

\vdots

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots | s_t = s]$$

$$= V^{\pi'}(s)$$

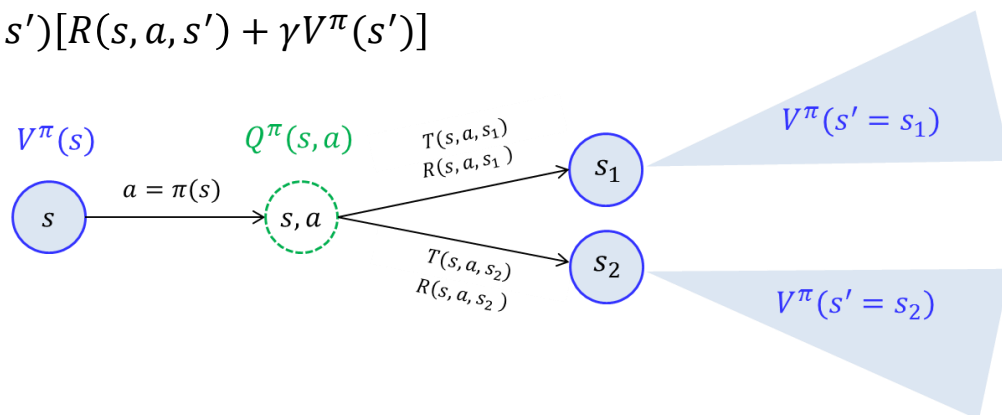
Thus, $\pi \leq \pi'$

Policy Improvement

Policy improvement :

The process of making a new policy π^{new} that improves the original policy π , by making it greedy or nearly greedy with respect to the value function of the original policy

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



Algorithm

Input : value of policy $V^\pi(s)$

Output: new policy π'

For each state $s \in \mathcal{S}$

1. Compute $Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$ for each a

2. Compute $\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} Q^\pi(s, a)$

$$= \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$$

Policy Iteration

Policy iteration :

Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement



Algorithm

```
 $\pi \leftarrow \text{arbitrary}$   
For  $t = 1, \dots, t_{PI}$  (or until  $\pi$  stops changing)  
  Run policy evaluation to compute  $V^\pi$   
  Run policy improvement to get new improved policy  $\pi'$   
   $\pi \leftarrow \pi'$ 
```

- Policy evaluation require iterative computation, requiring multiple sweeps through the state set
- policy evaluation starts with the value function for the *previous policy*

→ *Fast convergence*

Policy Iteration

Policy iteration :

Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement

Iteration

For $t = 0, \dots$ until convergence

For each state s :

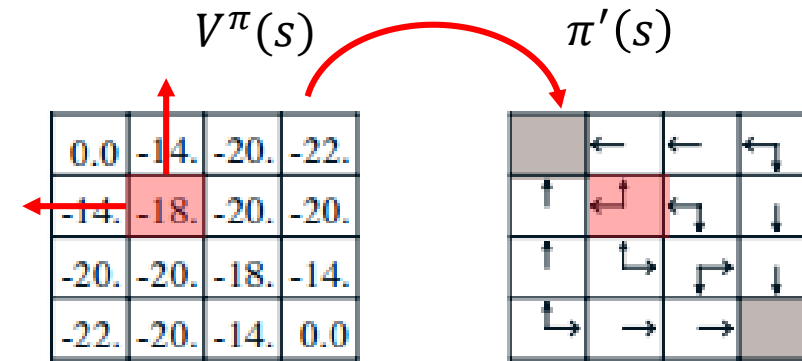
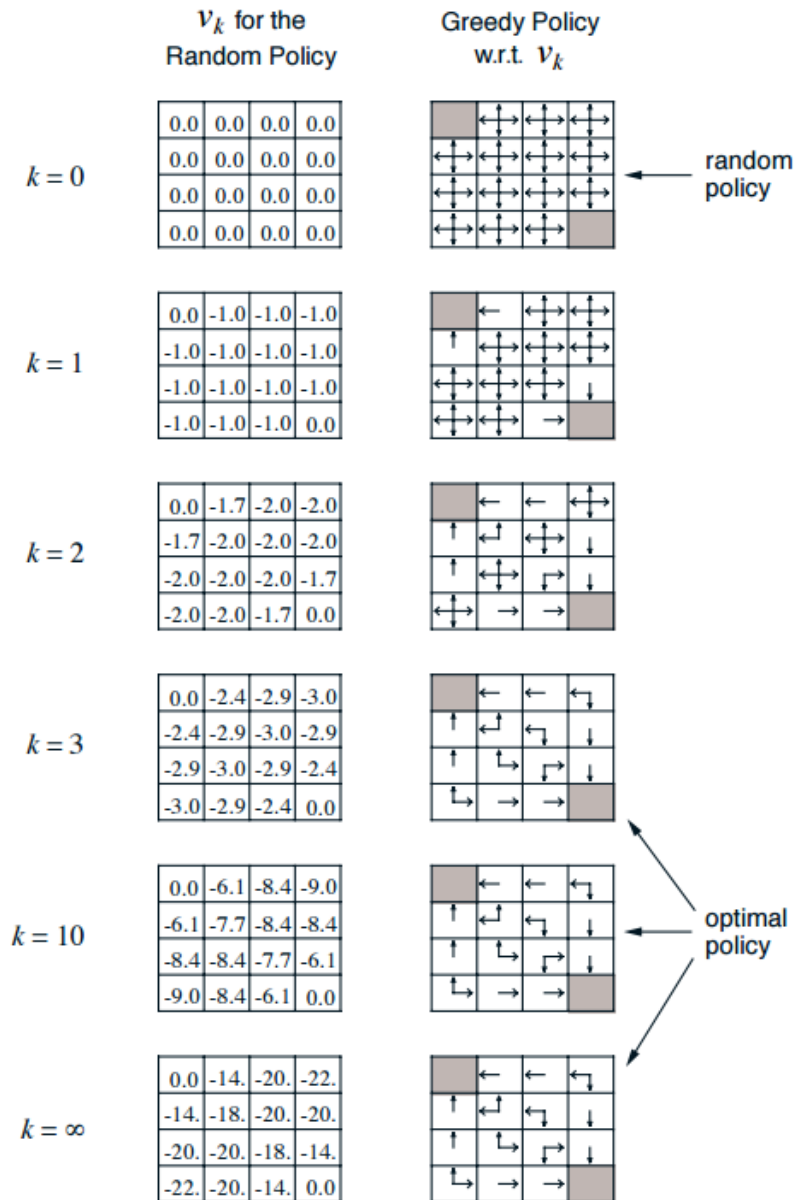
$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

Converged state value function $V^{\pi}(s)$

For each state s :

$$\begin{aligned} \pi'(s) &= \operatorname{argmax}_{a \in \mathcal{A}(s)} Q^{\pi}(s, a) \\ &= \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi}(s')] \end{aligned}$$

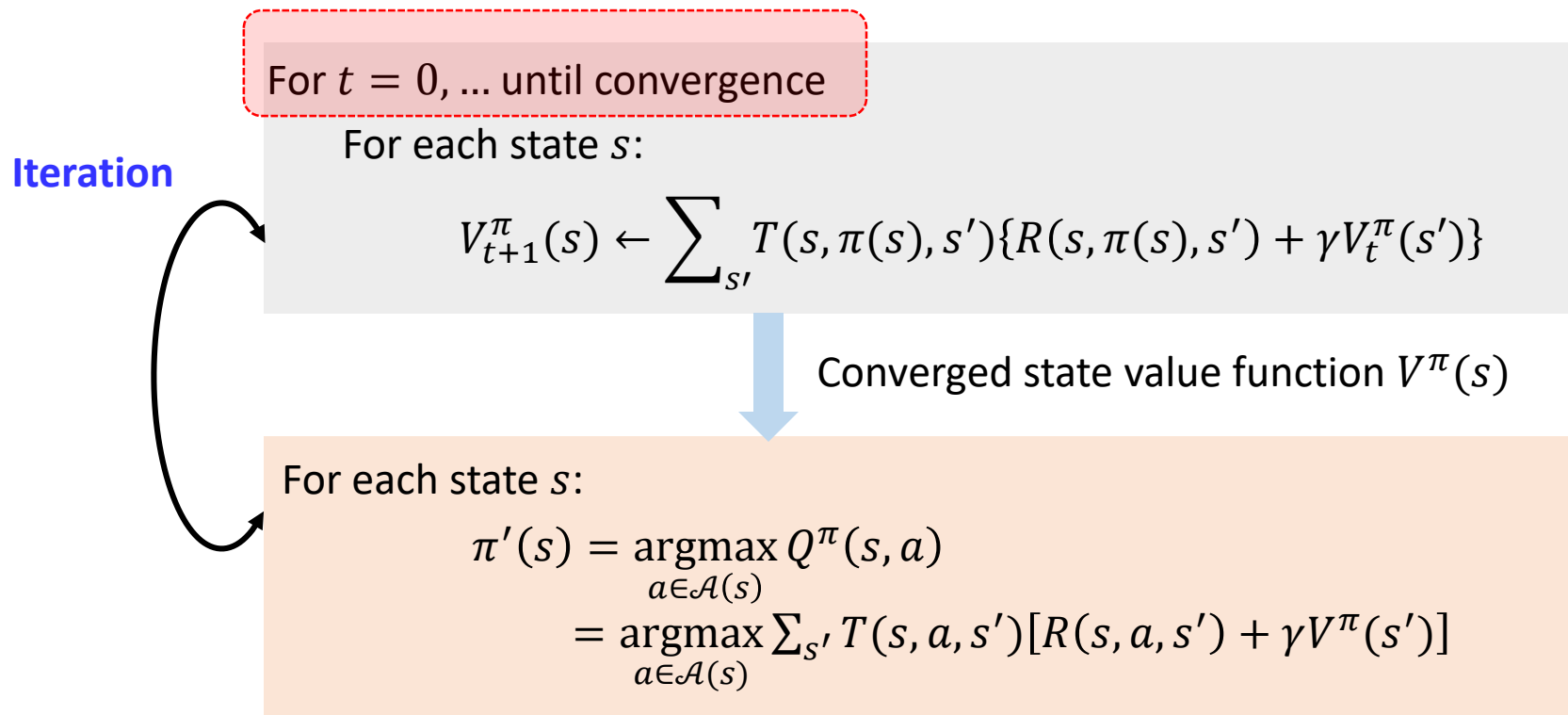
Policy Iteration



Policy Iteration

Policy iteration :

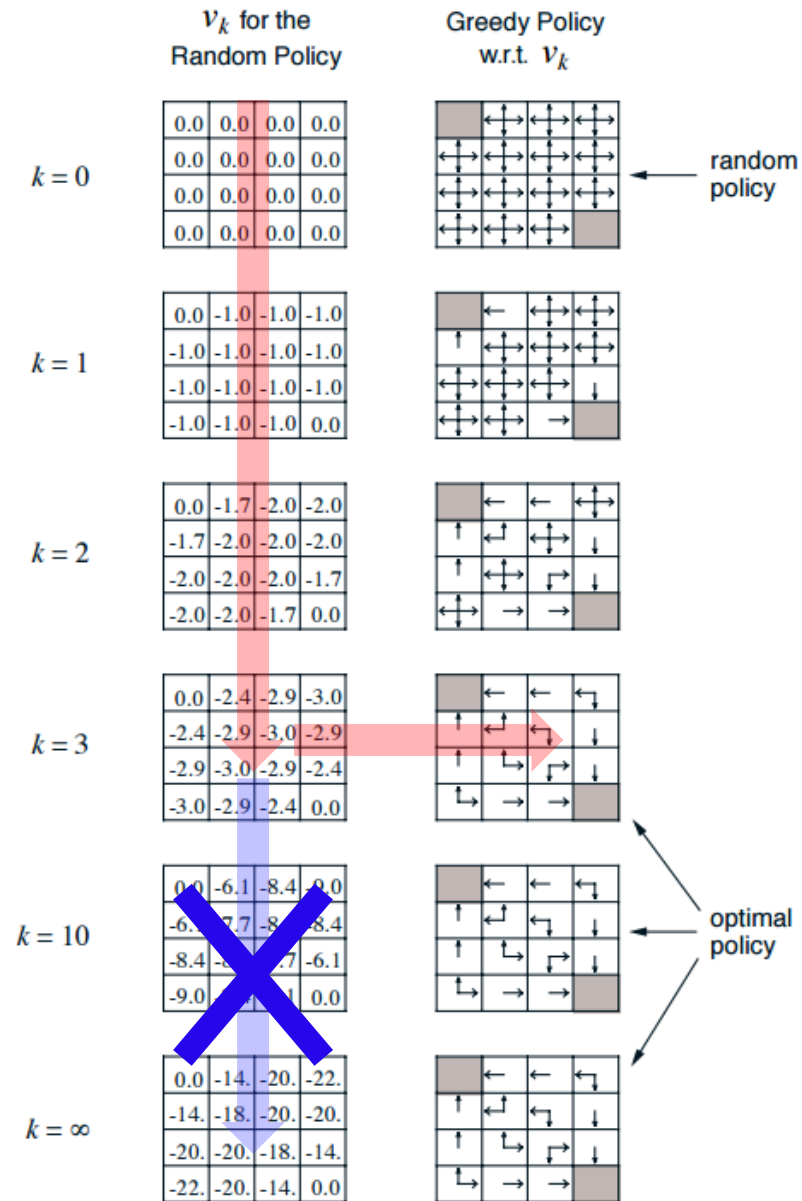
Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement



Issues :

Policy evaluation requires iterative computation, requiring multiple sweeps through the state set → slow to converge

Policy Iteration



Value Iteration

Solution:

- Stop policy evaluation after just one sweep (one backup of each state)
- Combine one sweep of policy evaluation and one sweep of policy improvement

For each state s :

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

+

For each state s :

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_{t+1}^{\pi}(s')]$$

↓

For each state s :

Value Iteration

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V_t(s')\}$$

or, can be obtained simply by turning the Bellman optimality equation into an update rule :

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

Value Iteration

Value Iteration:

A method to compute the optimum state-value function $V^*(s)$ by combining one sweep of policy evaluation and one sweep of policy improvement

Algorithm

Initialize $V(s) \leftarrow 0$ for all states $s \in S$

Repeat

For each state s :

$$V(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V(s)\}$$

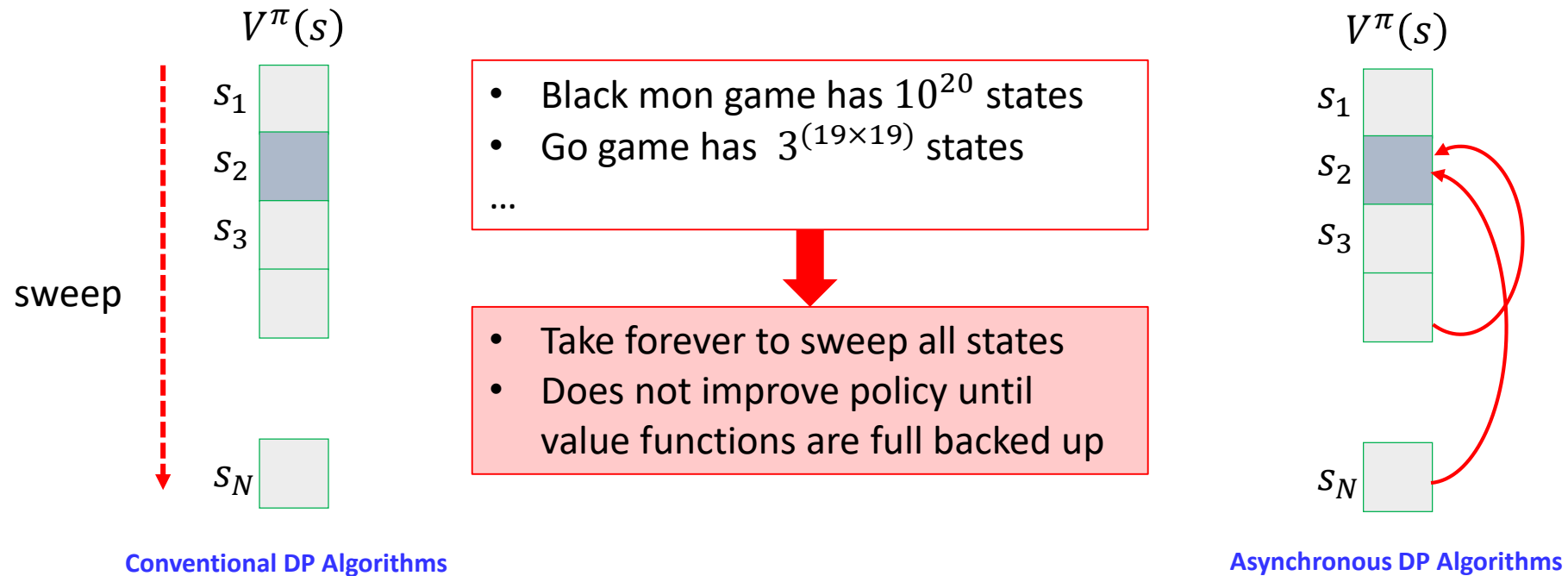
Until $\max_{s \in S} |V_t(s) - V_{t-1}(s)| \leq e$

Optimum policy can be obtained from the converged $V^*(s)$:

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s)\}$$

Asynchronous DP Algorithms

A major drawback to the DP methods is that they involve operations over the entire state set of the MDP



- Back up the values of states in any order whatsoever, using whatever values of other states happen to be available
- Allow great flexibility in selecting states to which backup operations are applied
- Make it easier to intermix computation with real-time interaction: To solve a given MDP, we can run iterative DP algorithm at the same time that an agent is actually experiencing the MDP (Reinforcement Learning !!!!)

Value Iteration

Policy Evaluation

For $t = 1, \dots$

For each state s :

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

Policy Improvement

For each state s :


$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi}(s')]$$

Policy Iteration




Value Iteration

For each state s :

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V_t(s)\}$$


Asynchronous Value iteration

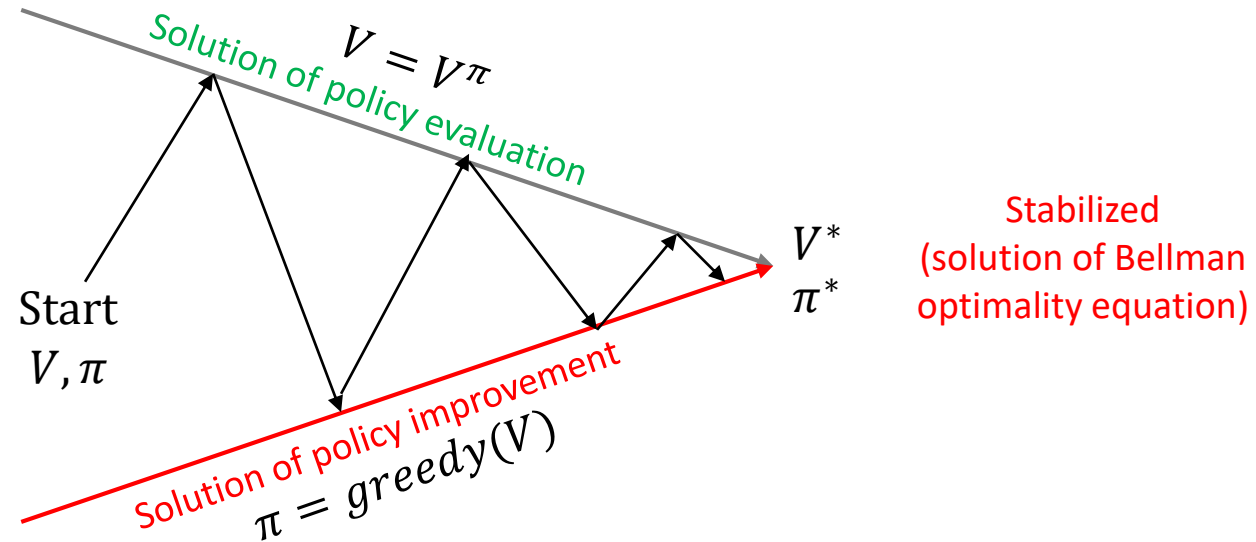
For any single state s :

$$V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V(s)\}$$


As long as both processes continue to update all states, the ultimate result is typically the same-convergence to the optimal value function and an optimal policy

Generalized Policy Iteration

Generalized Policy Iteration :
an general idea of interaction between policy evaluation and policy improvement processes

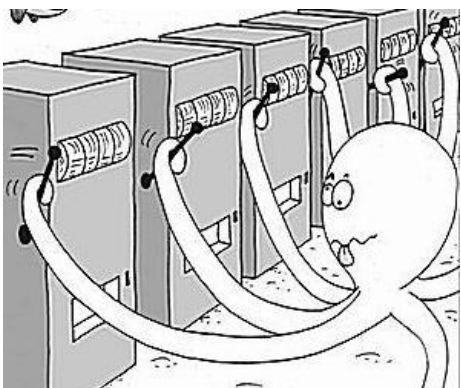


- Making policy greedy with respect to the value function typically makes the value function incorrect for the changed policy
- Making the value function consistent with the policy typically causes that policy no longer to be greedy
- In the long run, however, these two processes interact to find a single joint solution: the optimal value function and optimal policy

Introduction to Value Based 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 $r \sim P(\theta_i)$: θ_i is unknown
- $a_t \in \{1, \dots, n\}$: the choice of machine at time t
- r_t : the reward at time t
- Policy π 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 π^* 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

ϵ - *greedy* action selection rule

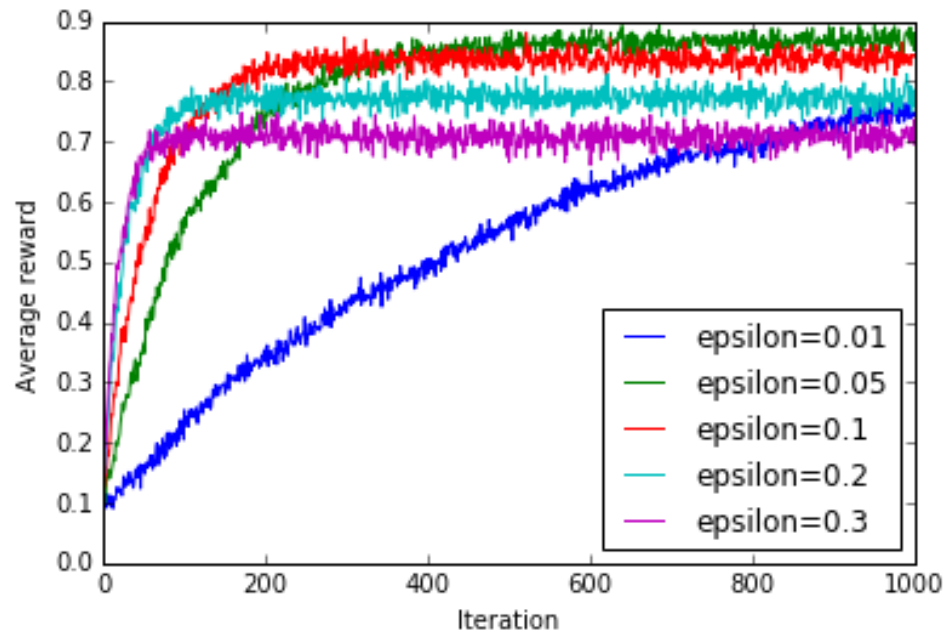
$$\pi(a) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)| & \text{If } a = a^* \\ \epsilon/|A(s)| & \text{If } a \neq a^* \end{cases}$$

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

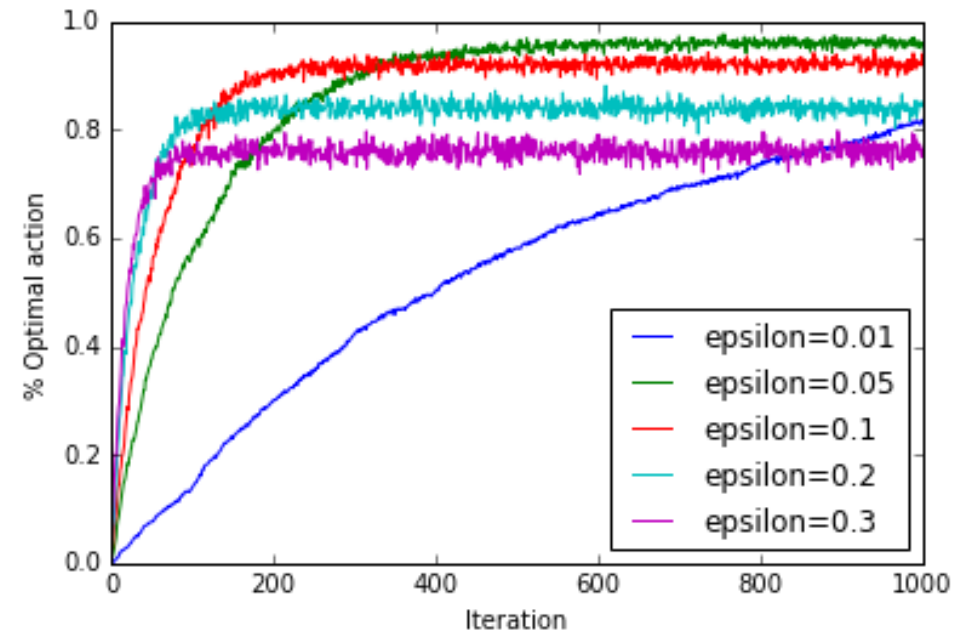
- ✓ In the limit as the number of plays increases, every a will be sampled an infinite number of times, guaranteeing $k_a \rightarrow \infty$ thus $Q_t(a) \rightarrow Q^*(a)$
- ✓ The probability of selecting the optimum action converges to greater than $1 - \epsilon$

Action-Value Methods

Average reward $E[r_t]$



% of selecting a^*



Relationship with model based approach

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

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



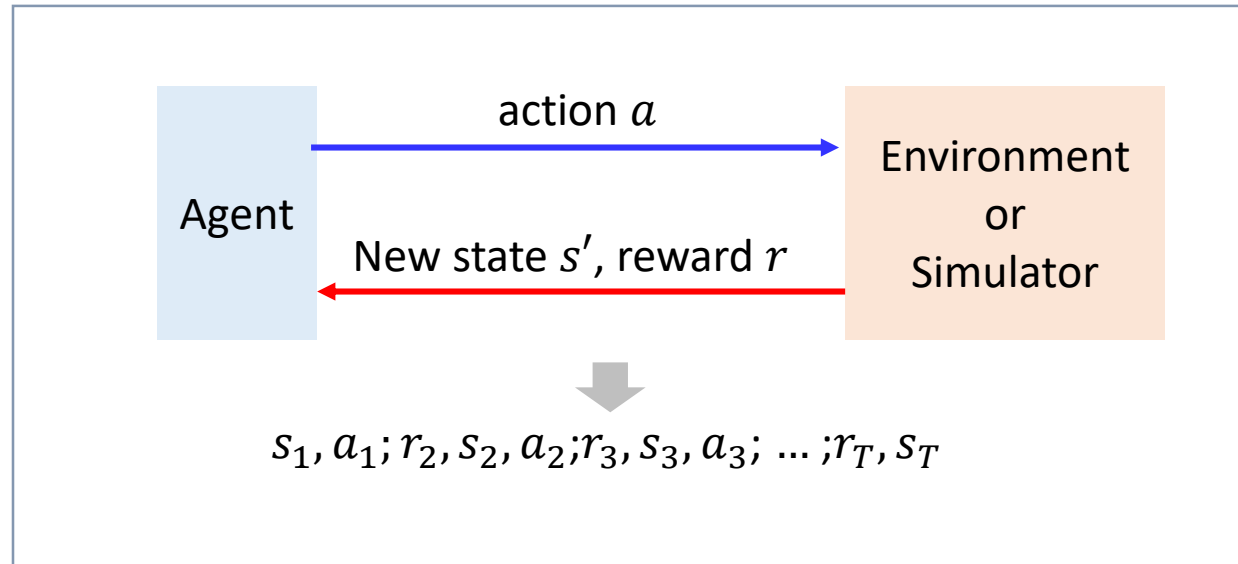
Reinforcement Learning (Offline & Online)

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

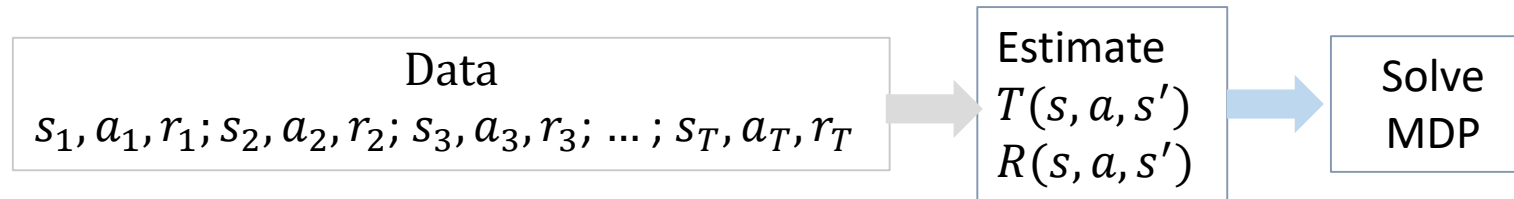
Choose action $a_t = \pi(s_t)$ (how?) : **Decision making**

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

		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

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

- Data following policy π

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

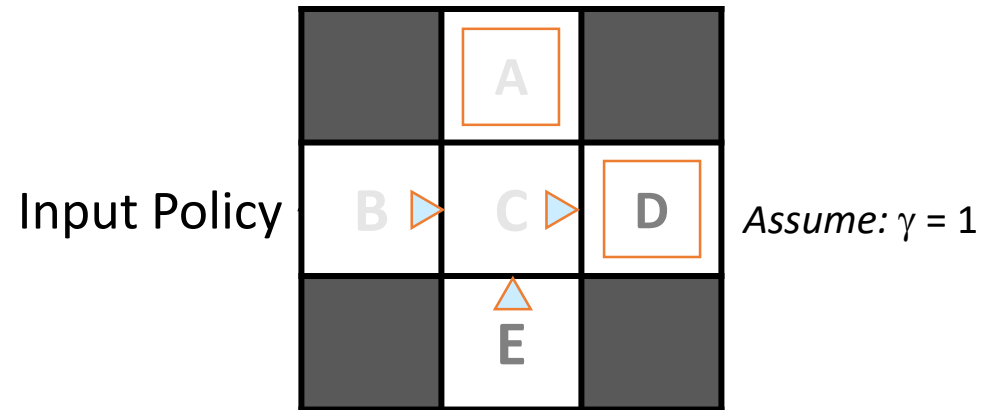
- Transition model

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

- 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

Monte Carlo Policy Evaluation

Key Idea : Monte Carlo Policy Evaluation

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

Data (following policy π):

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

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 π
- The value of state is the expected utility - **expected accumulative future reward** starting from s and following the policy π

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

(State, Action, Reward) pairs generated by policy π

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 π
- The value of state is the expected utility - **expected accumulative future reward** starting from s and following the policy π

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

(State, Action, Reward) pairs generated by policy π

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

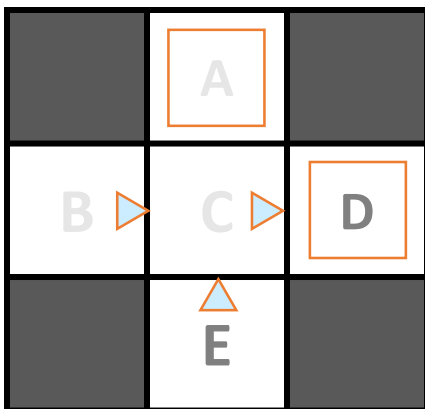
Every visit to s

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

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

Example

Input Policy π



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	

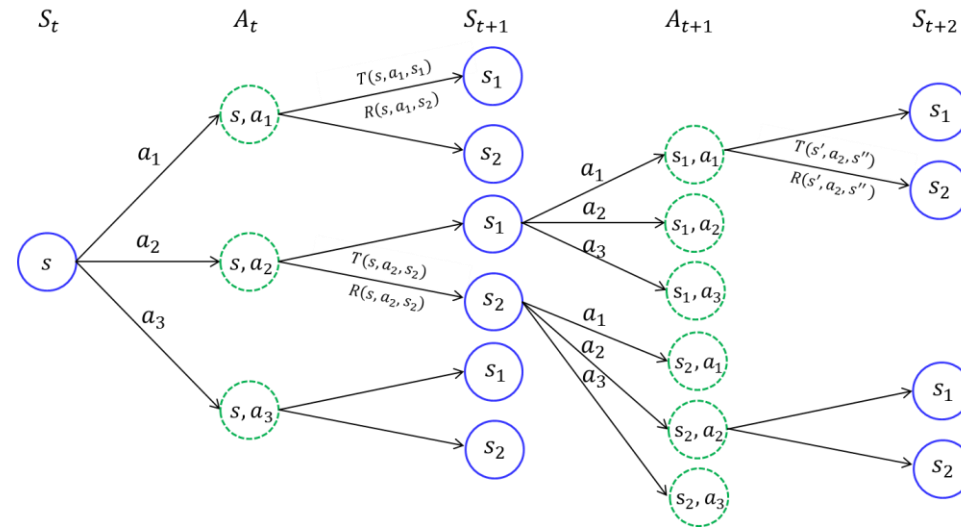
Example

- 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

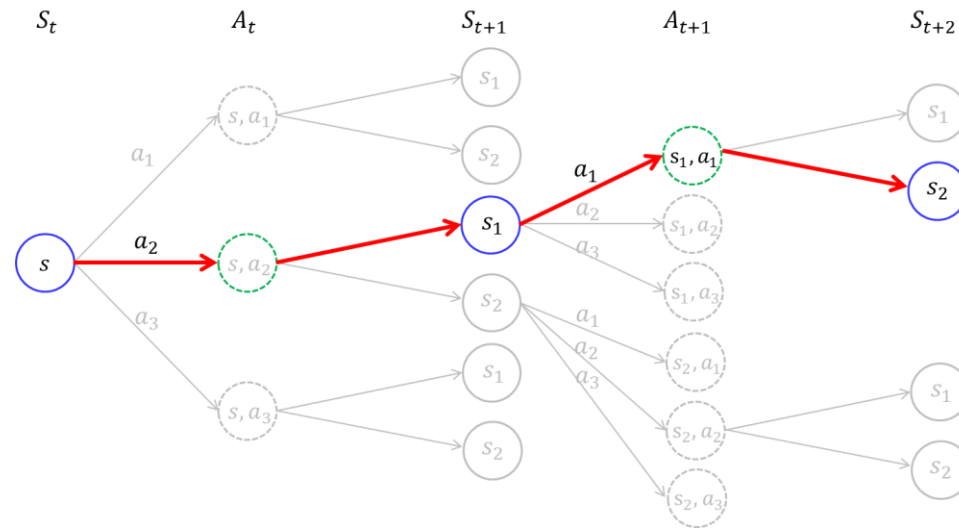
Output Values

	-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 $Q^*(s, a)$

Monte Carlo Estimation of Action Values

Key Idea : Monte Carlo Policy Evaluation

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

Data (following policy π):

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

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 π
- The value of action-state is the expected utility - **expected accumulative future reward** starting from s and following the policy π

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

(State, Action, Reward) pairs generated by policy π

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
 \vdots
Episode ∞

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:

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

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

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 π is a deterministic policy, many relevant state –action pairs may never be visited
 - ✓ then in following π , 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 π , 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?

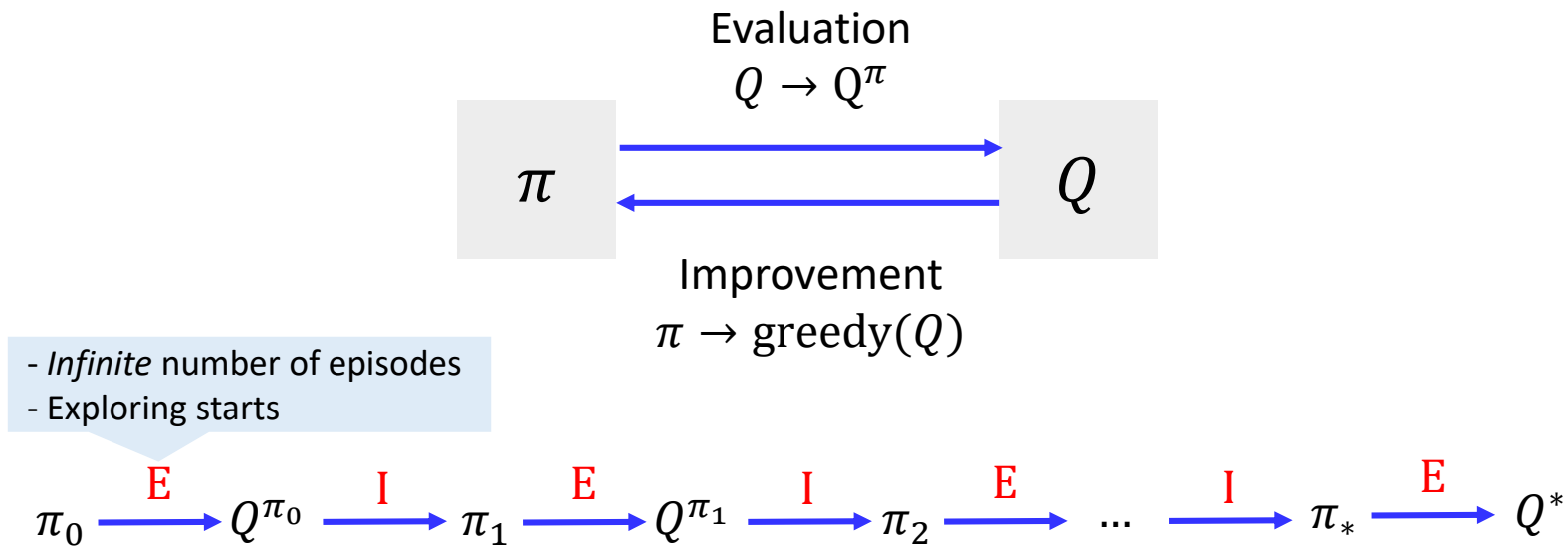
- **Exploring starts + infinite number of episodes**
 - ✓ Start from an arbitrary state-action pair (s, a) with a non-zero probability
 - ✓ Guarantees **that all state-action pair will be visited an infinite number of times in the limit of an infinite number of episodes**
 - ✓ Cannot be used when learning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

Monte Carlo Control

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 π
- **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 π
 - Infinite number of episodes under policy π
 - 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
$s_1, a_1, r_1; s_2, a_2, r_2; s_3, \ddots, r_3; \dots; s_T, a_T, r_T$	$\rightarrow u$	Episode ∞

Average expected reward

I Policy Improvement:

- Policy improvement can be done by constructing each π_{k+1} as the greedy policy with respect to Q^{π_k}

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

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

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

(Policy improvement theorem)

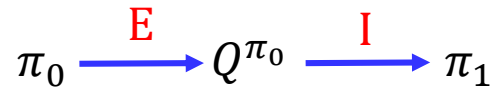
Overall process converges to an optimal policy and the optimal value function

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 π

(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:

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

} Policy improvement

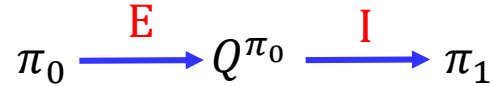
In an single episode, both Policy evaluation and policy improvement proceeds together

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

Monte Carlo Control

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

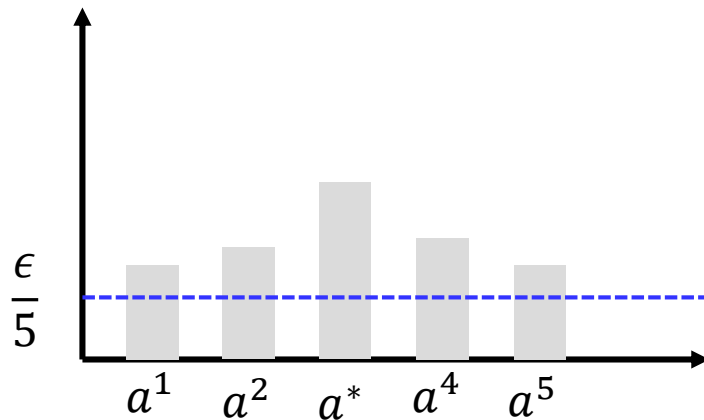
Key Idea : On-policy methods: Soft policies

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

ϵ – soft policy

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

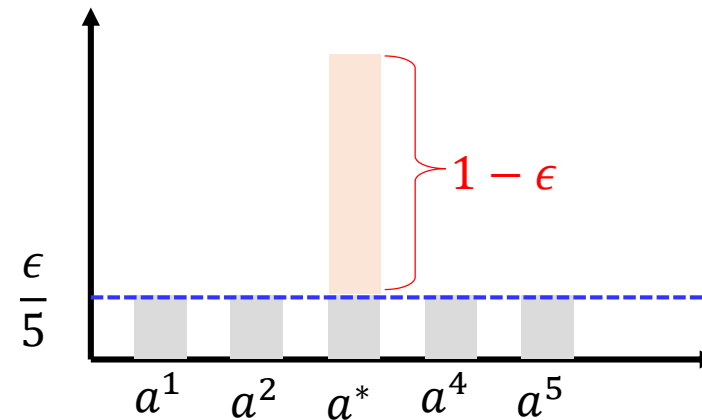
$$\sum_a \pi(s, a) = 1$$



ϵ – 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 π' respect to Q^π is an improvement over any $\epsilon - soft$ policy π

Let π' 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 : $\epsilon - \text{soft}$ 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 $\epsilon - \text{soft policy}$

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

Repeat forever:

(a) Generate *an episode* using π

(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

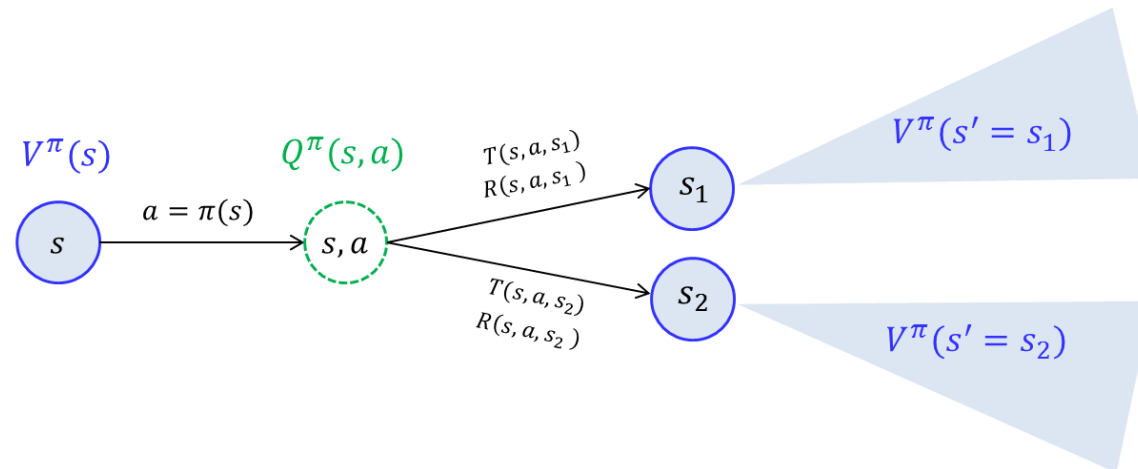
On-Policy TD Control (SARSA)

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- 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

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

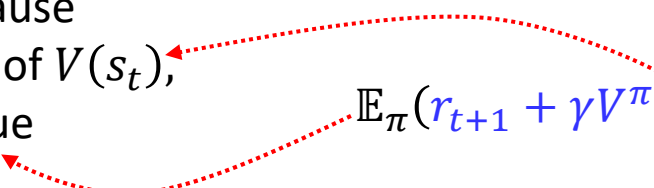
Bootstrapping

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^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'

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

$s \leftarrow s'$

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

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

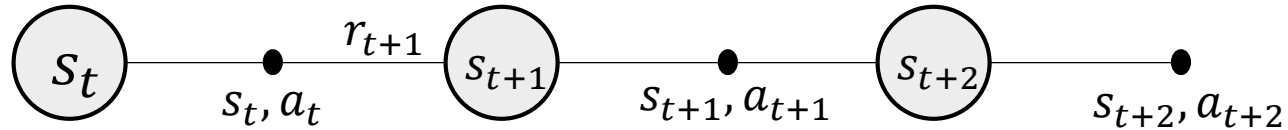
Advantages of TD Policy Evaluation (prediction)

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^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 *constant* – α 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

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

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):

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

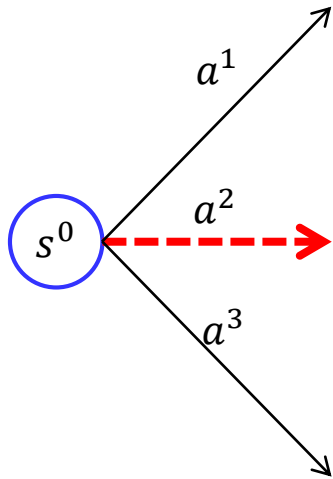
Behavioral policy
||
Estimation policy

Until s is terminal

- As in all on-policy methods, we continually estimate Q^{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}

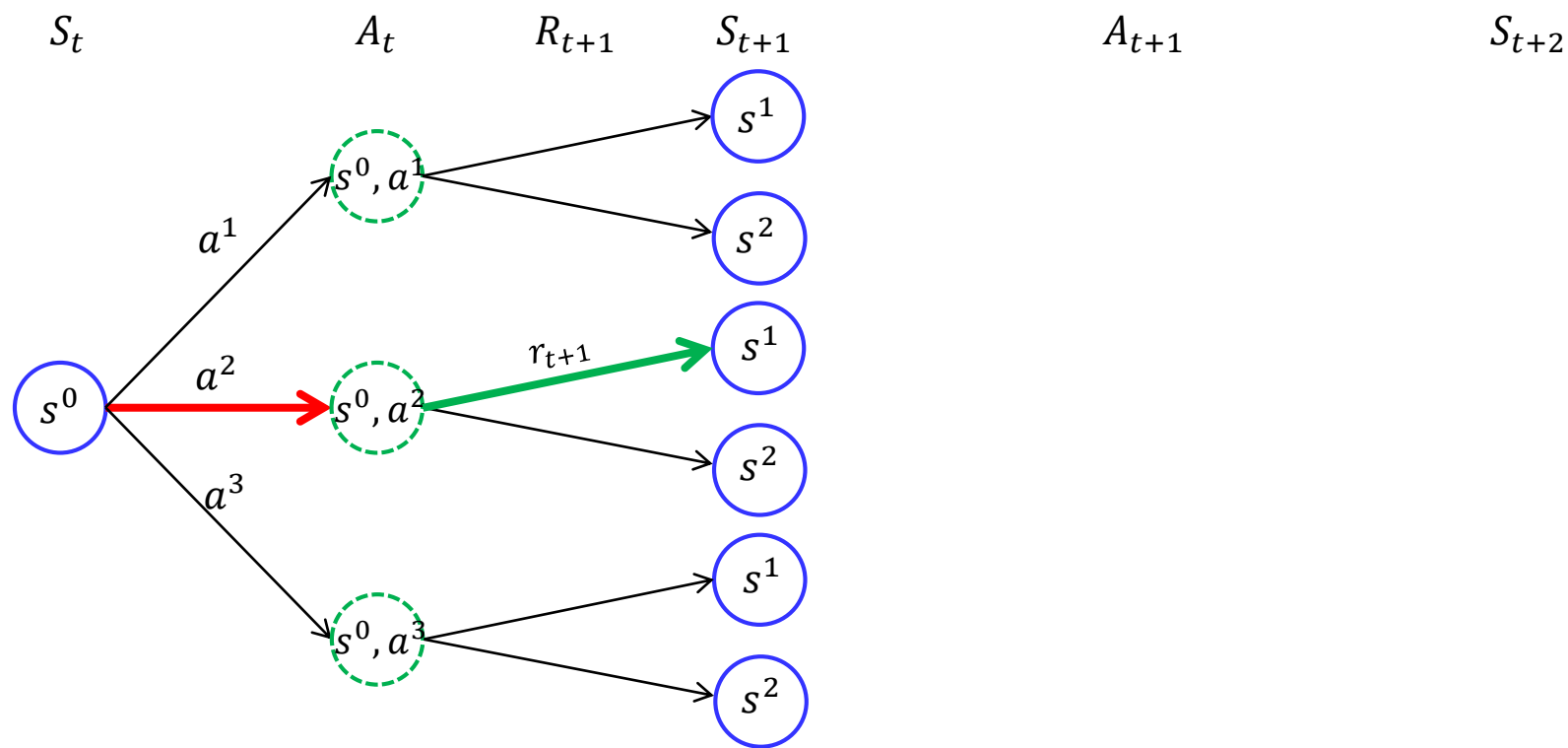


Choose a_t from $s_t = s^0$ using current Q

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

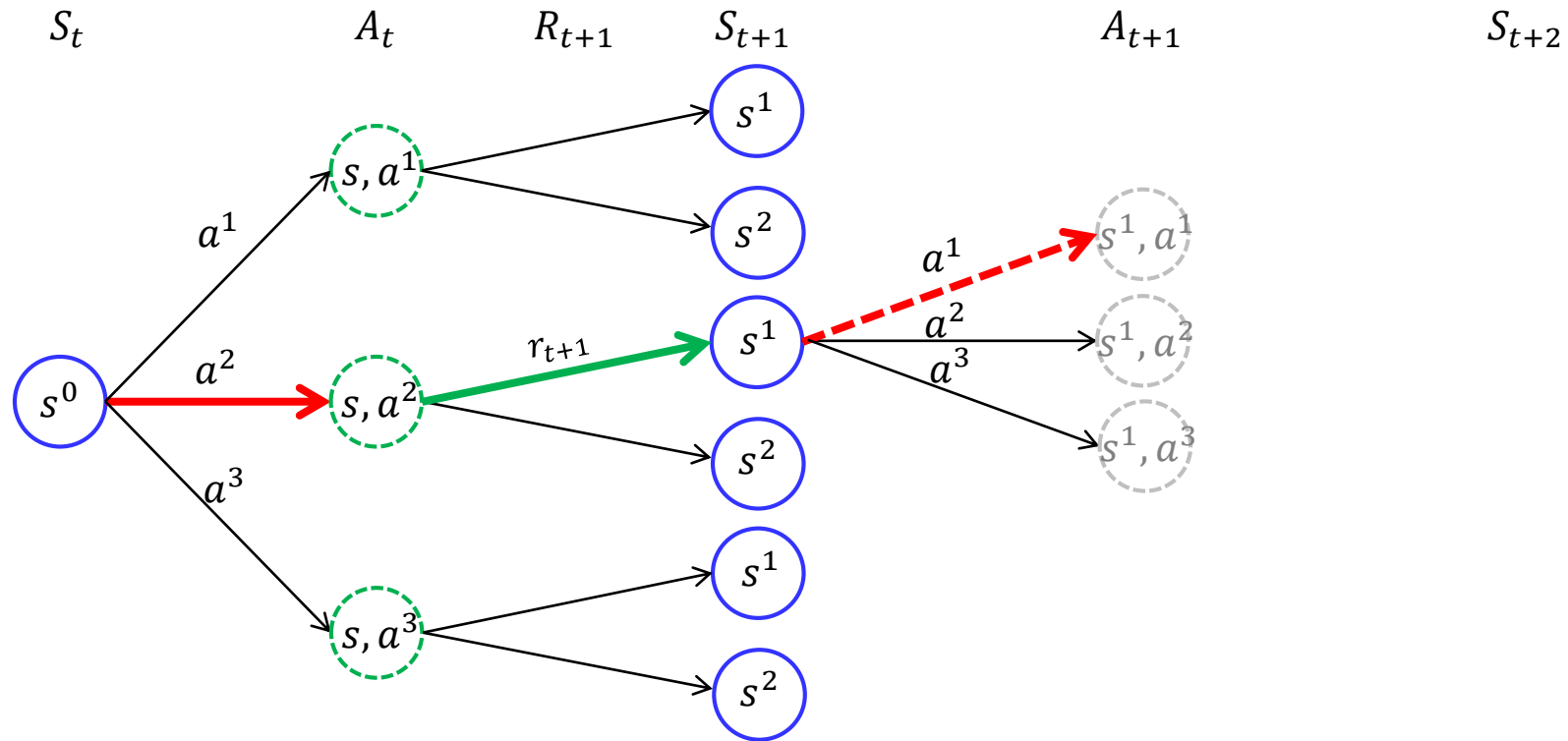
Assume a^2 is chosen

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

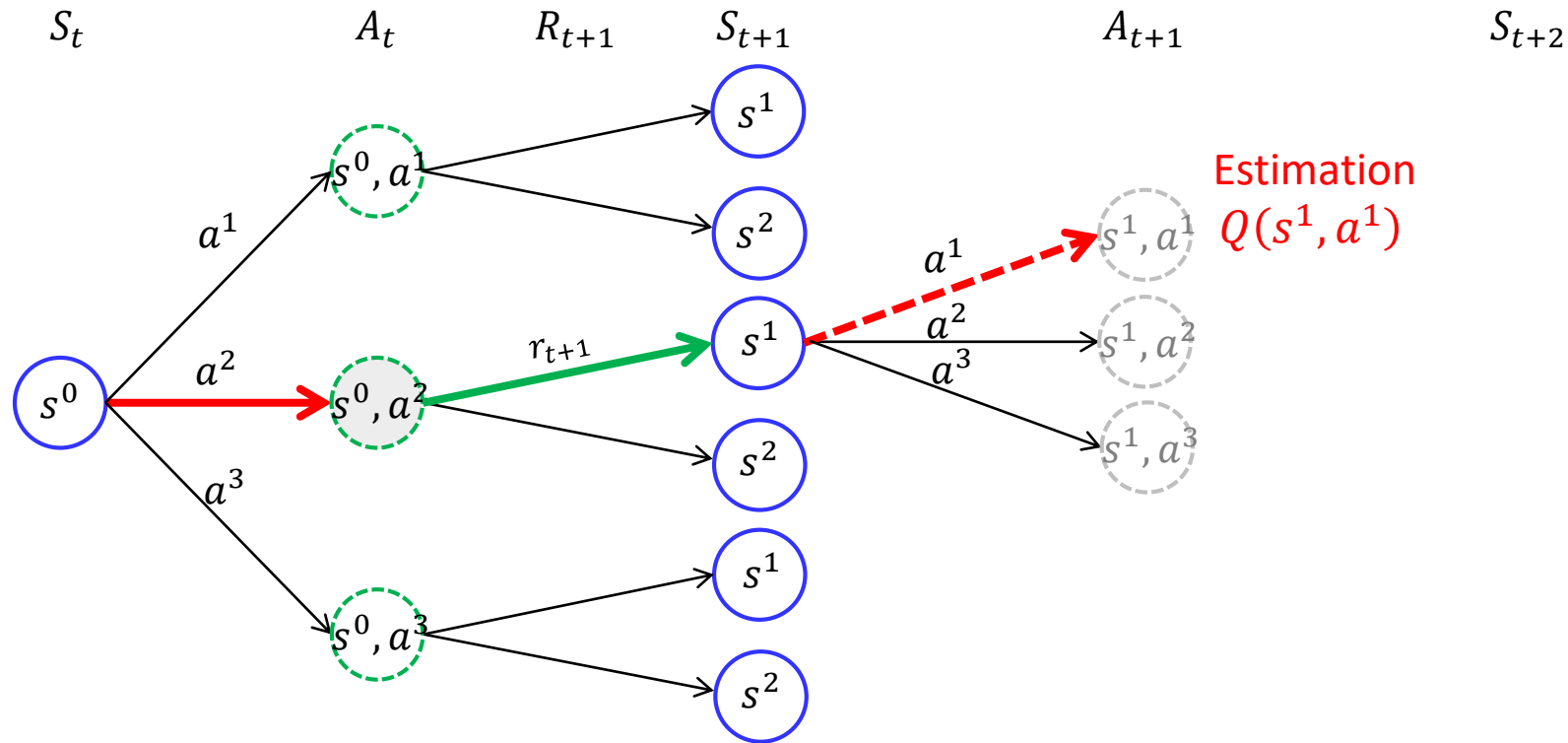


Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

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

Assume a^1 is chosen

Sarsa: On-Policy TD Control

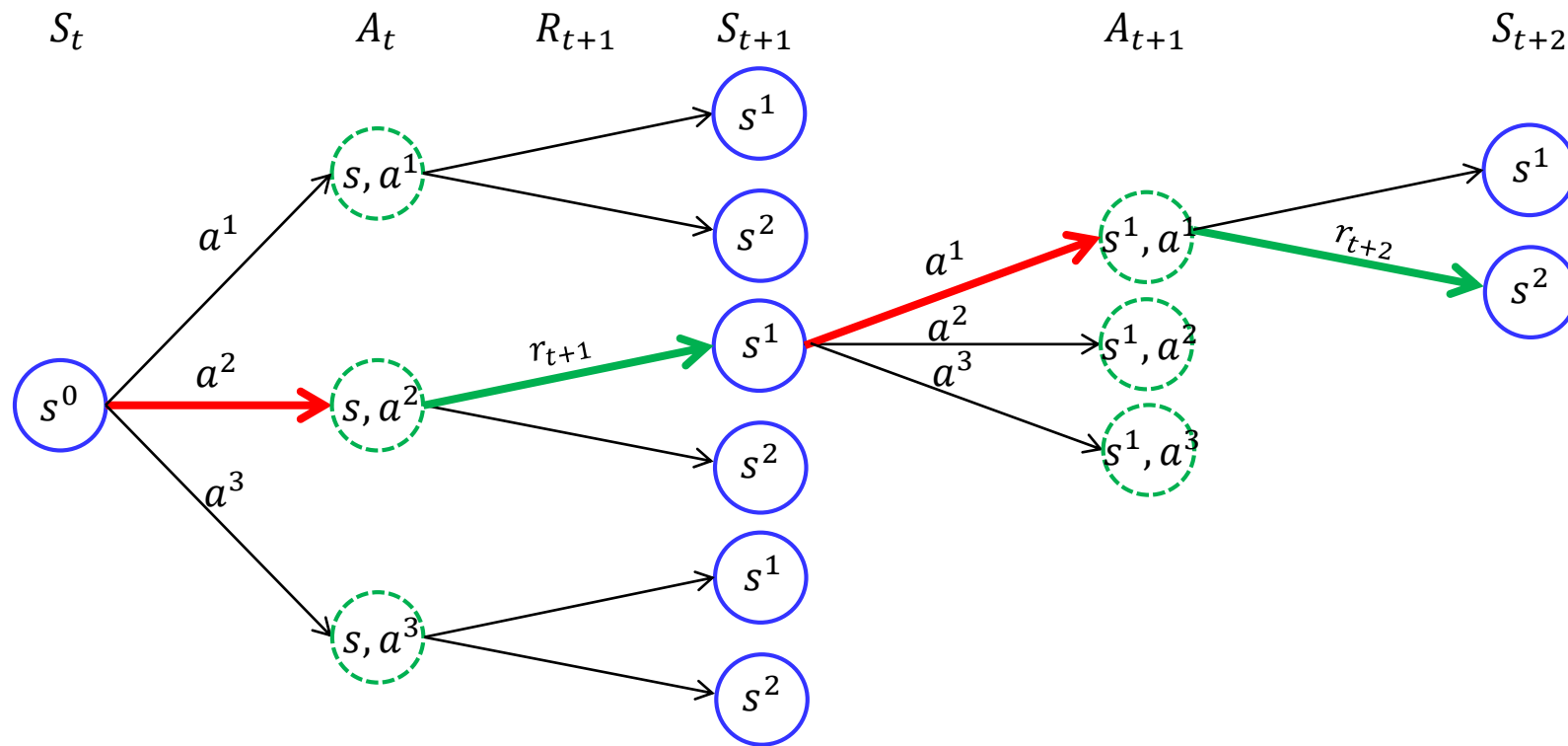


Update Q function with the **estimation** $Q(s_{t+1}, a_{t+1})$

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

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

Sarsa: On-Policy TD Control



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

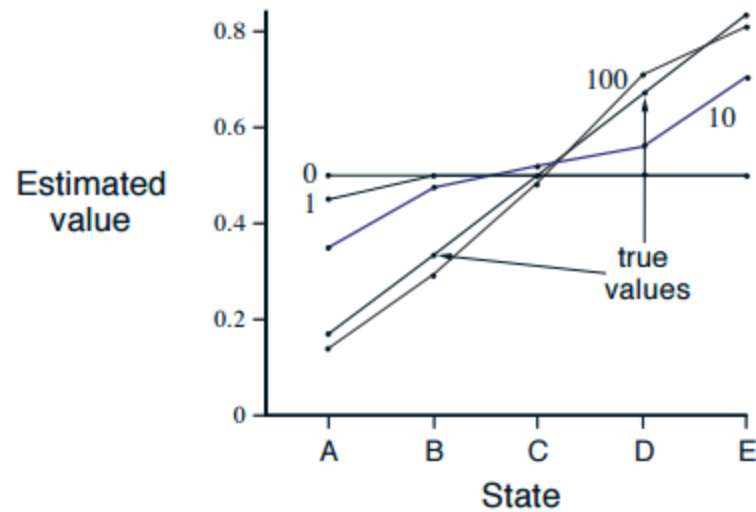
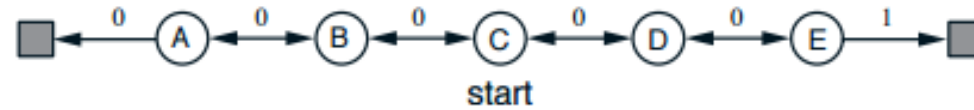
Why Q-learning is considered as Off-Policy method

The diagram illustrates the off-policy nature of Q-learning. A green arrow points from the Q-value update equation to the state s in the tuple (s, a, r, s', a') . Blue curved arrows point from $\pi(s)$ to s and from $\pi(s')$ to s' . Red straight arrows point from s to a and from s' to a' .

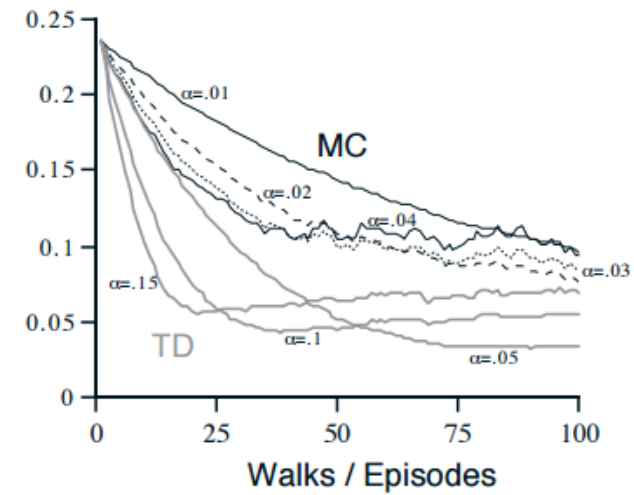
$$Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma Q(s', a') - Q_\pi(s, a))$$

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

A small Markov process for generating random walk

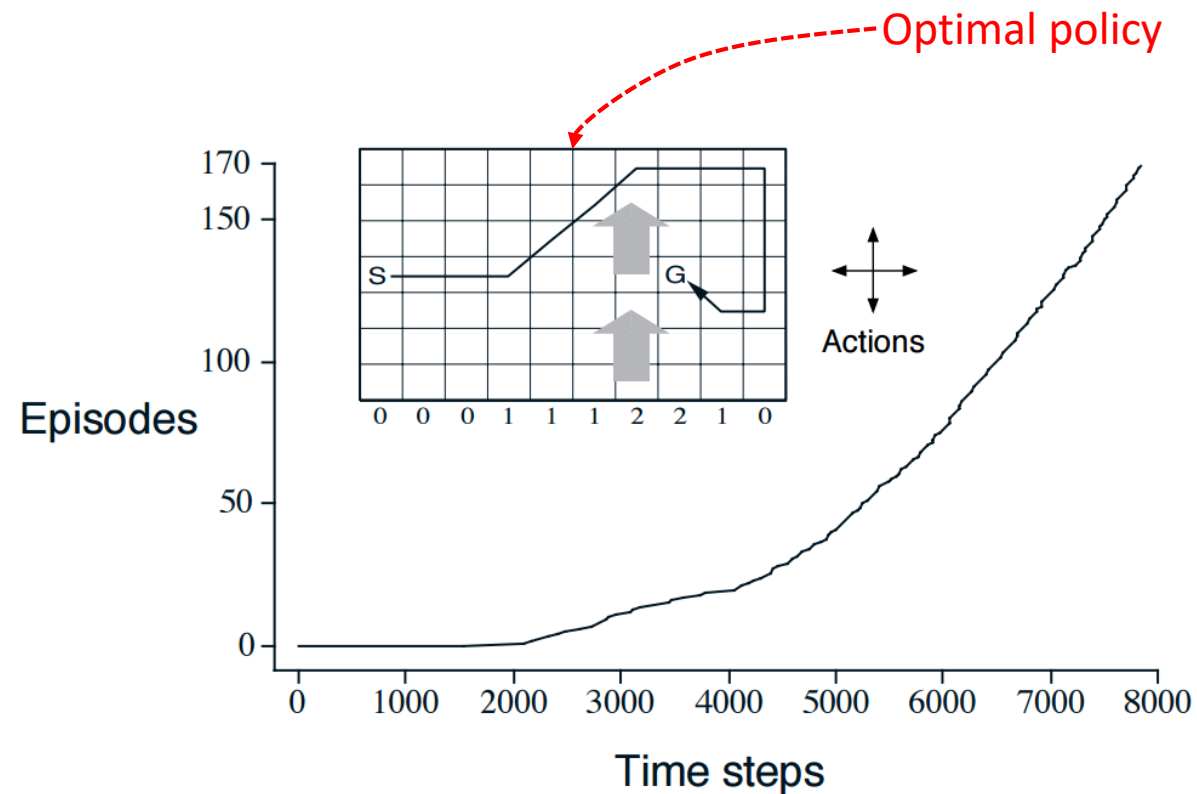
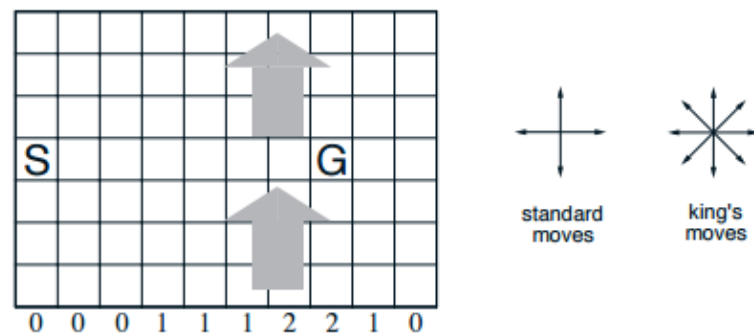


RMS error,
averaged
over states



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

State transition is encoded by this figure



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)

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

- 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)

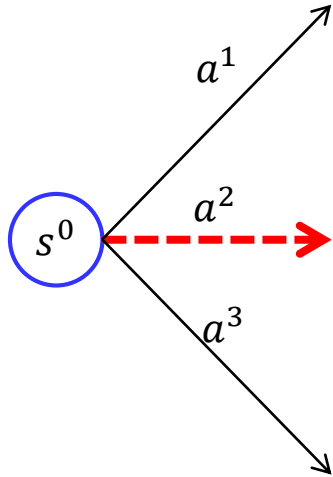
-Estimation can be greedy)

$a^* = \operatorname{argmax}_{a'} Q(s', a)$ 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}

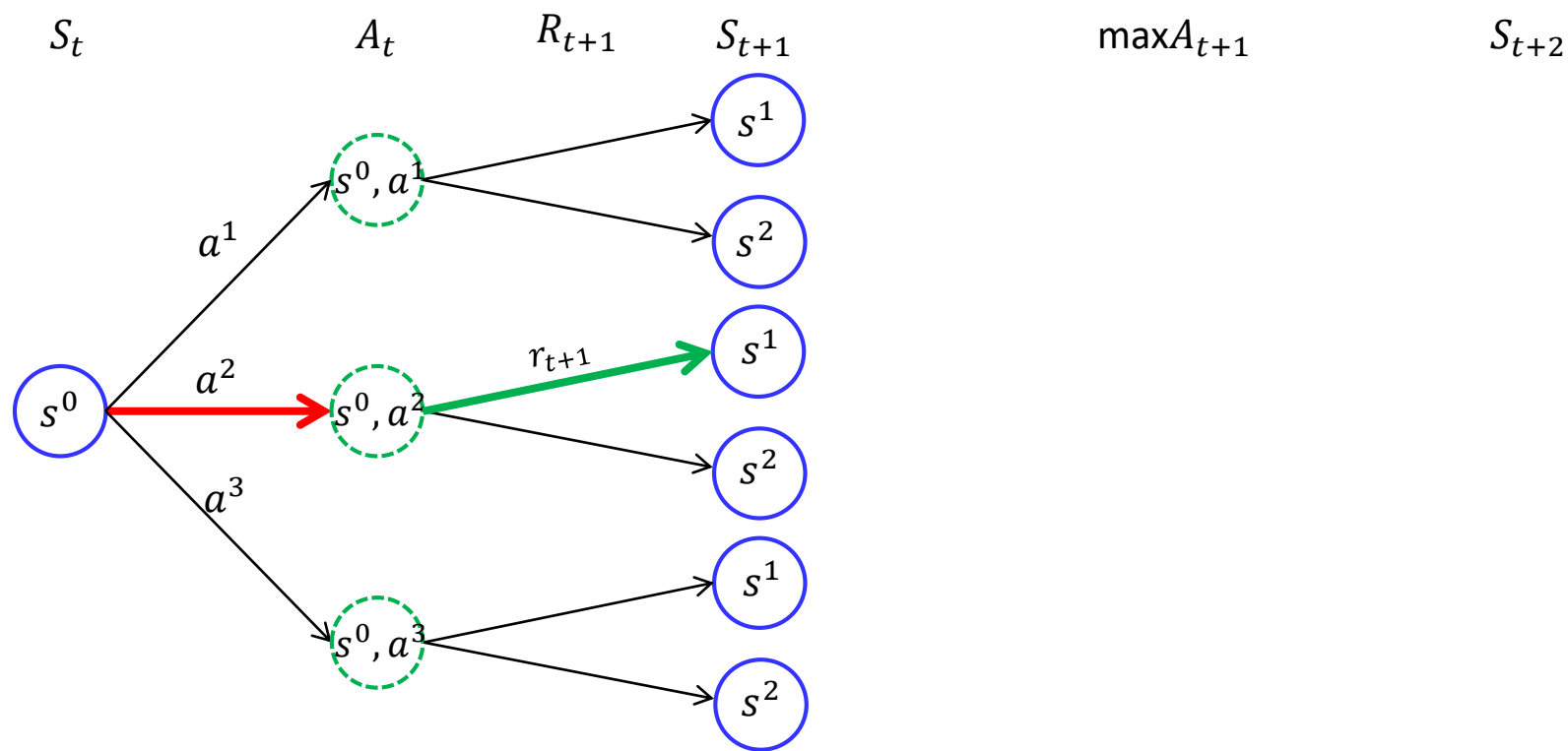


Choose a_t from $s_t = s^0$ using current Q

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

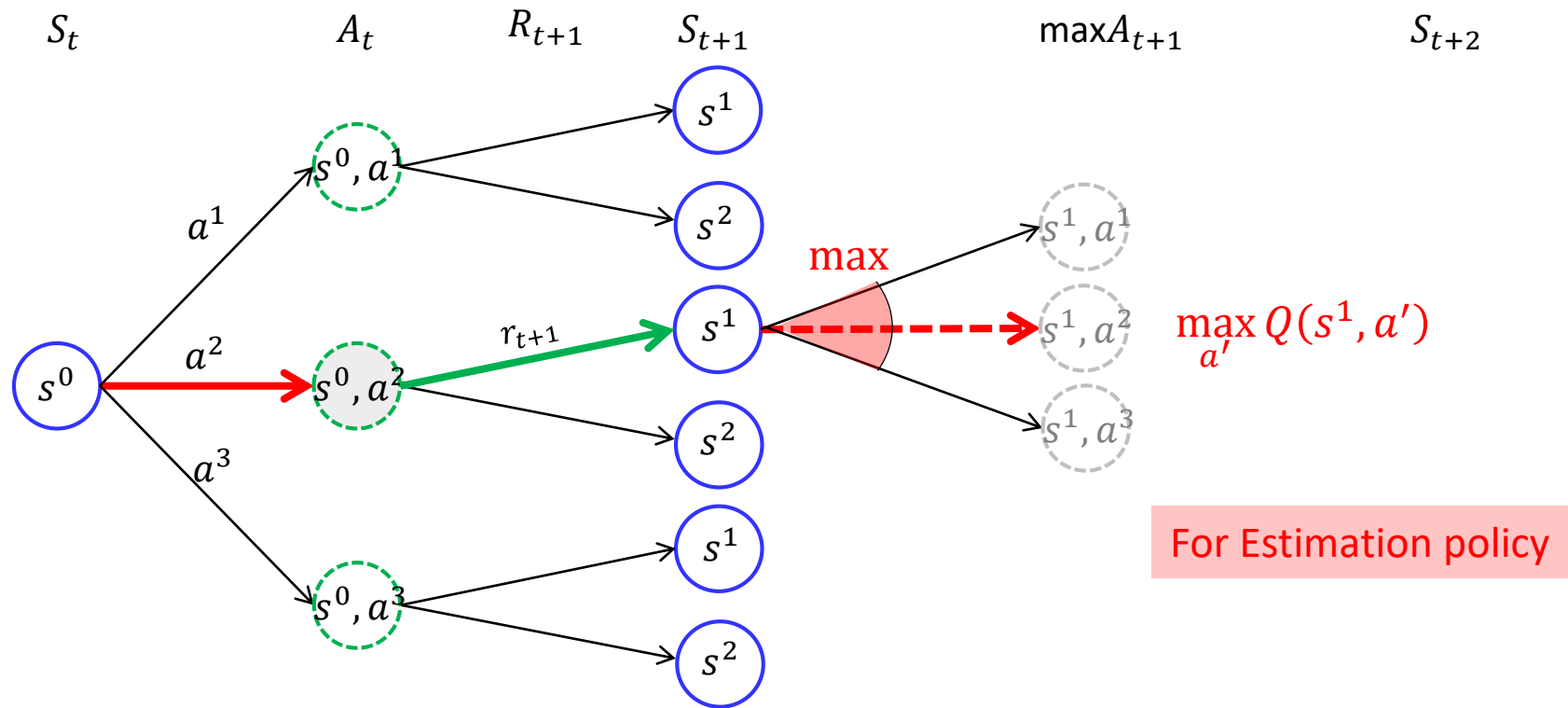
Assume a^2 is chosen

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

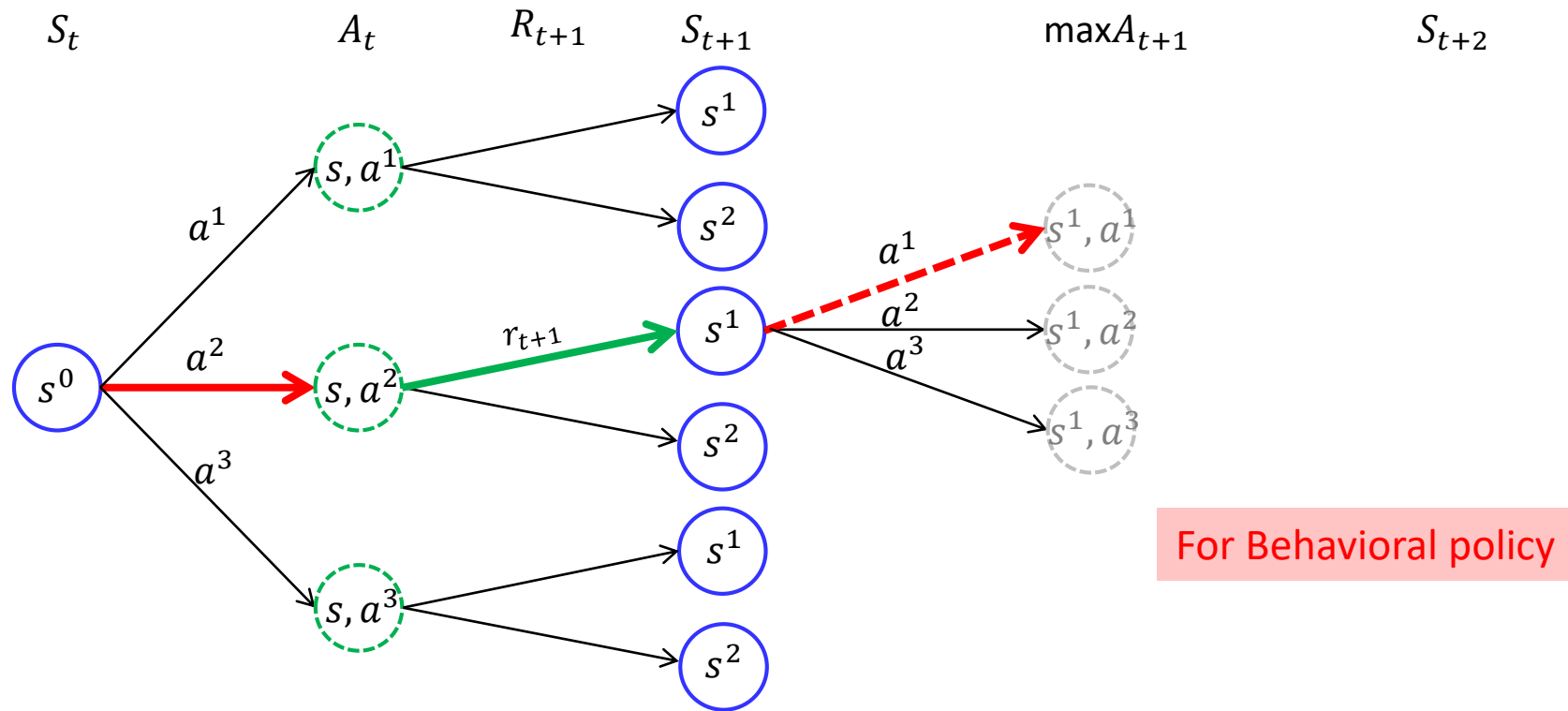


Update Q function with the $\max_{a'} Q(s^1, a')$

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

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

Q-Learning: Off-Policy TD Control

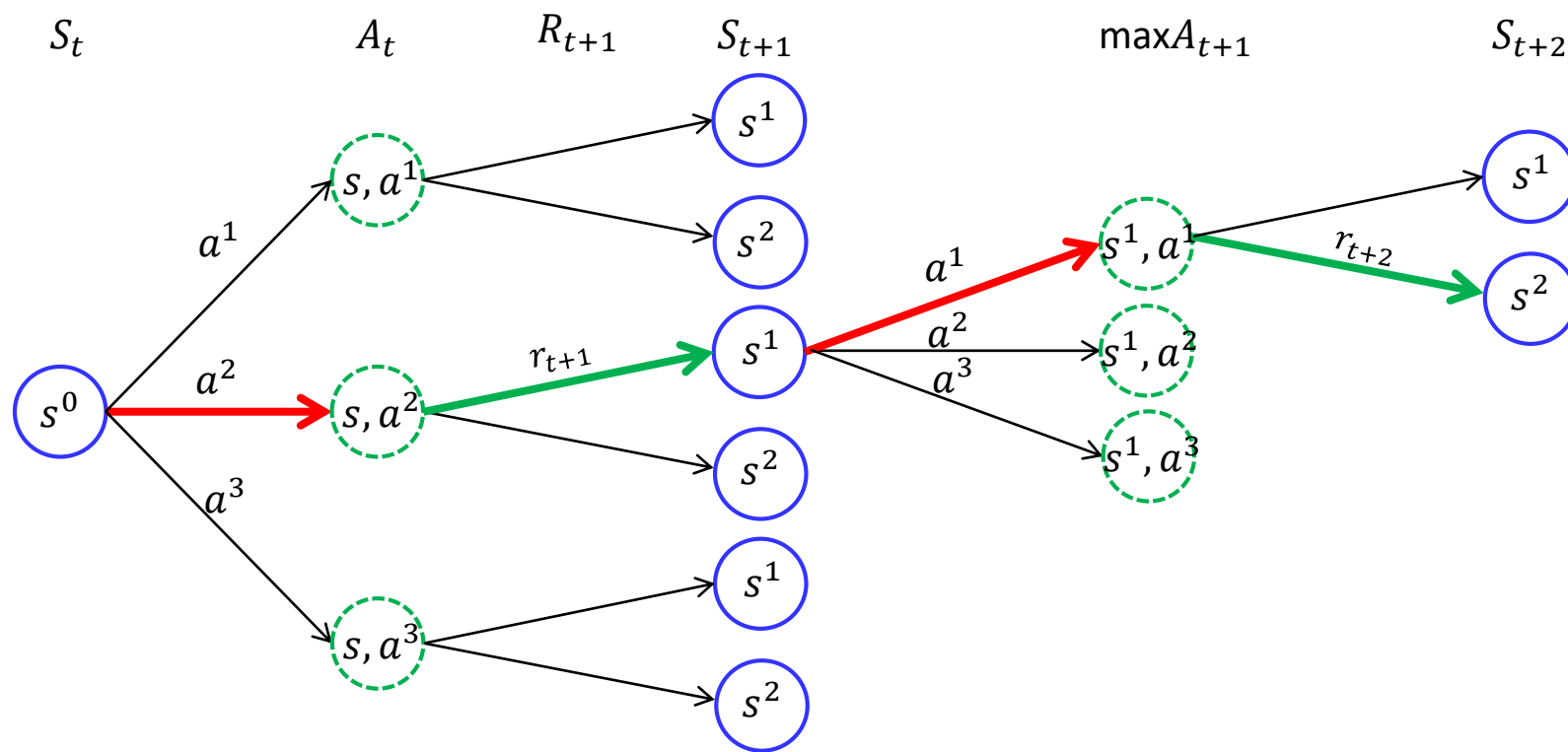


Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

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

Assume a^1 is chosen

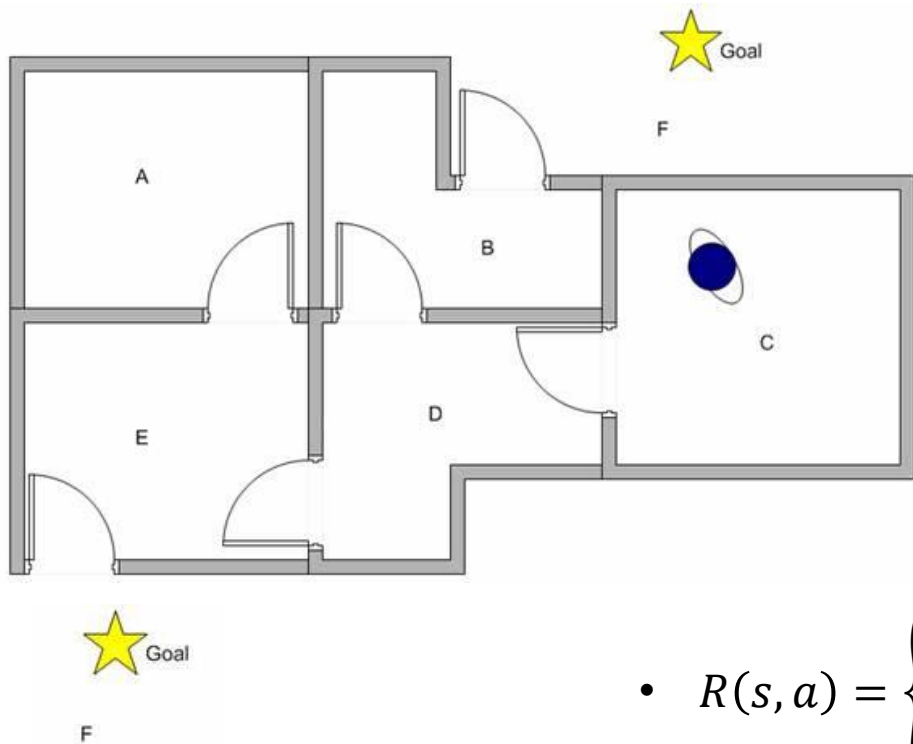
Q-Learning: Off-Policy TD Control



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)

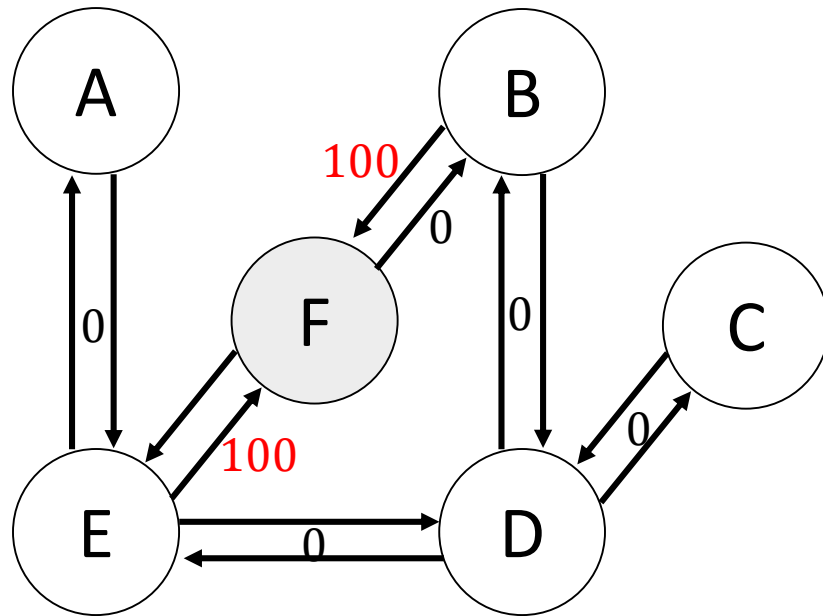


$$MDP = \{\mathcal{S}, \mathcal{A}, T, R, \gamma\}$$

- $s \in \mathcal{S} = \{A, B, C, D, E, F\}$
- $a \in \mathcal{A} = \{A, B, C, D, E, F\}$
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



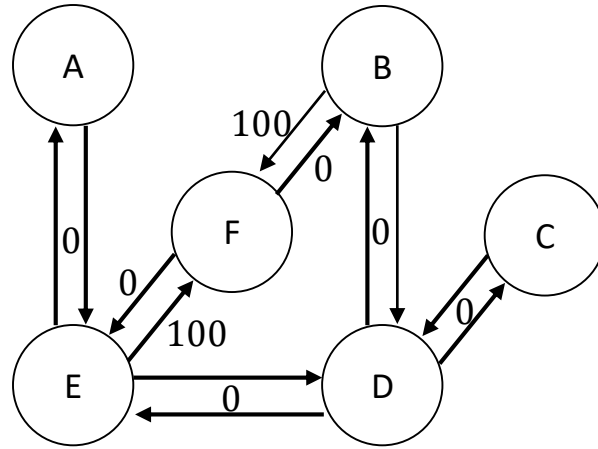
$R(s, a) =$

$s \backslash a$	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:

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

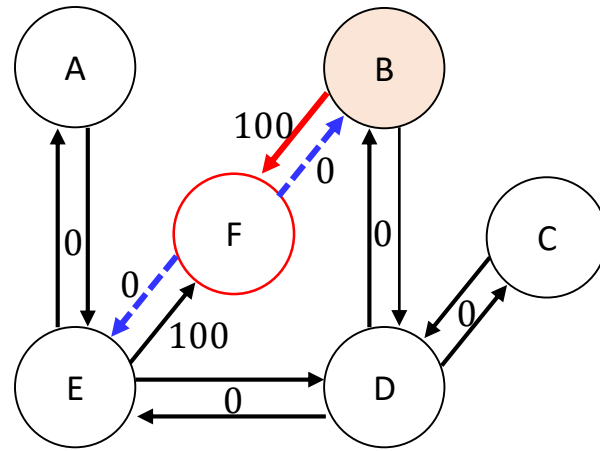
Q-learning Step-by-Step Example

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


[illegible]

Q-learning Step-by-Step Example

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



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

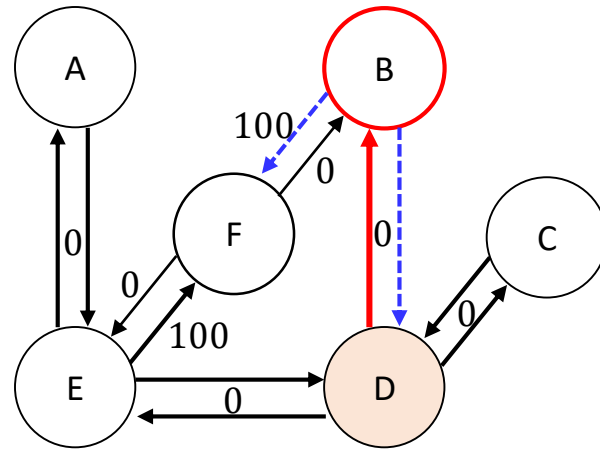
1. Assume the initial state is B and take action F randomly (stochastic policy):

$$Q(B, F) \leftarrow Q(B, F) + 0.5 \left(R(B, F) + 0.8 \max_a \{Q(F, B), Q(F, E)\} - Q(B, F) \right)$$
$$Q(B, F) \leftarrow 0 + 0.5(100 + 0.8 \times 0 - 0) = 50$$

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

Q-learning Step-by-Step Example

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



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

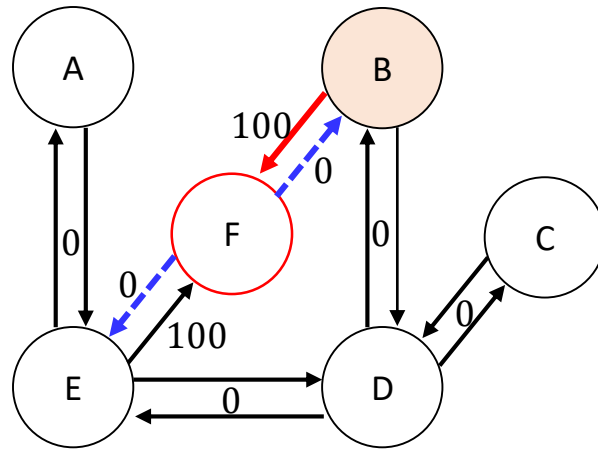
1. Assume the initial state is D and take action B randomly (stochastic policy):

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

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

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):

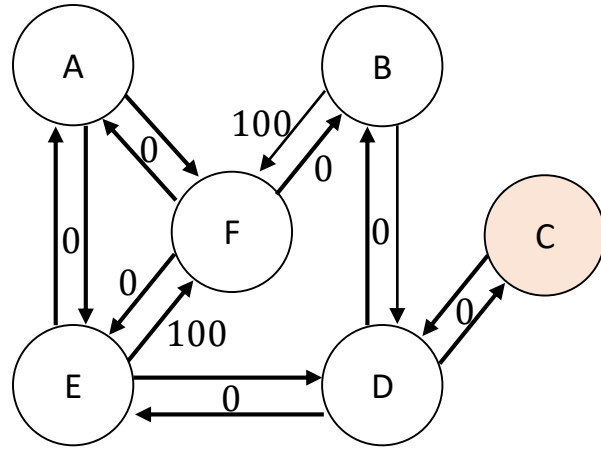
$$Q(B, F) \leftarrow Q(B, F) + 0.5 \left(R(B, F) + 0.8 \max_a \{Q(F, B), Q(F, E)\} - Q(B, F) \right)$$

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

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

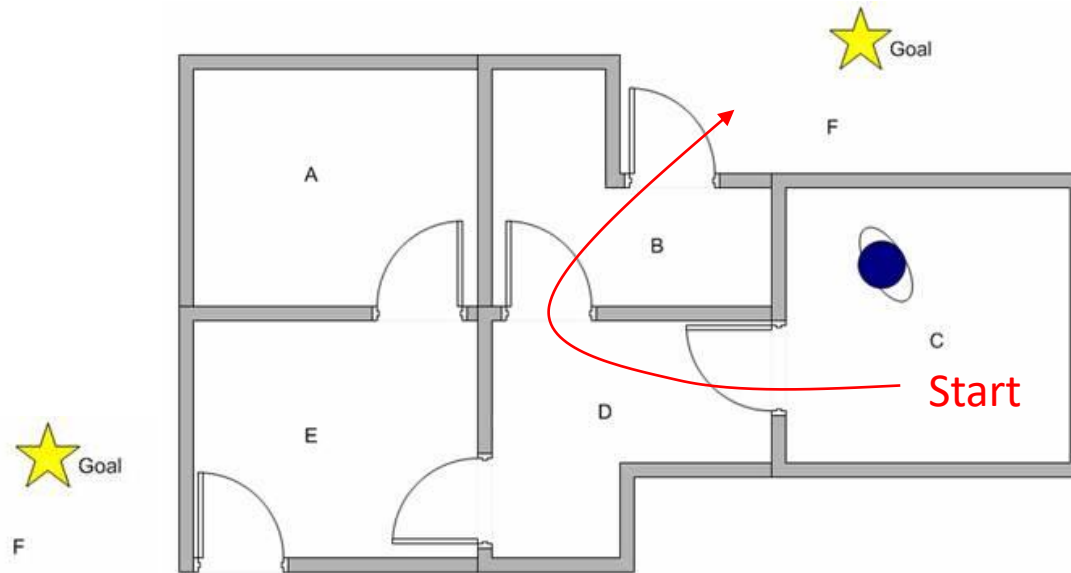
Q-learning Step-by-Step Example

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

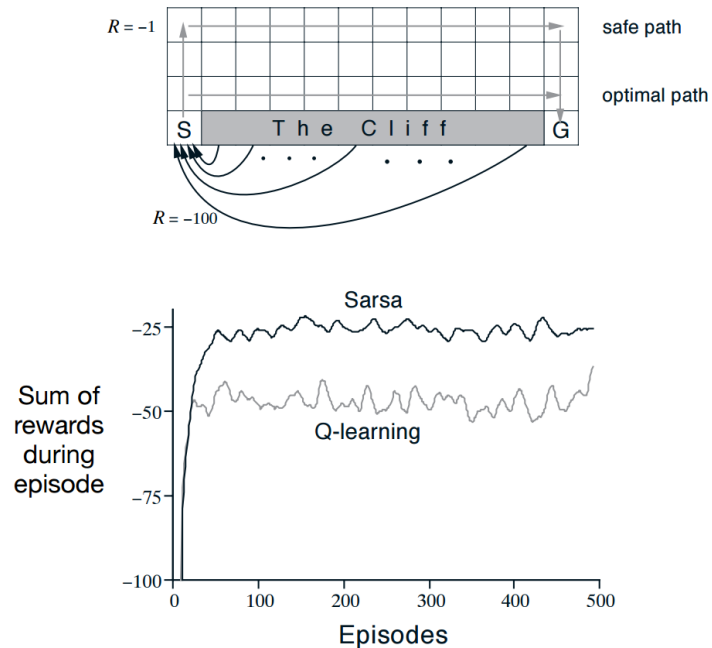


$$Q^*(s, a) = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\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

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.

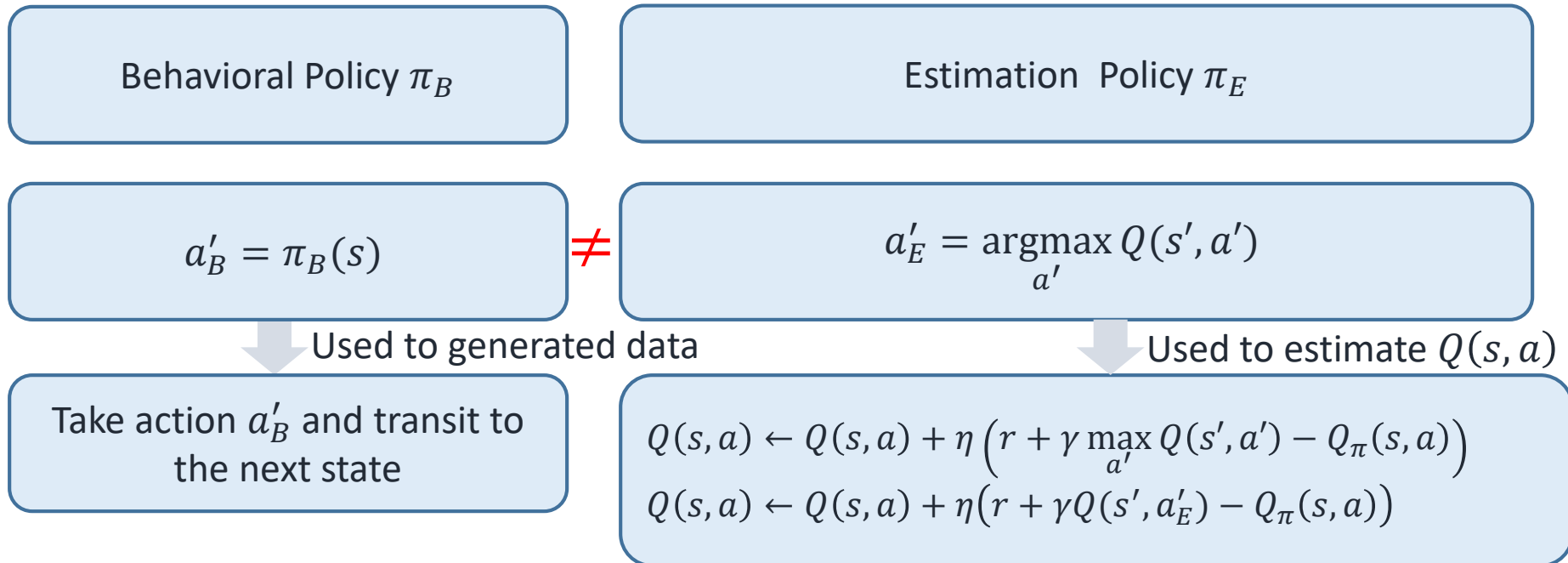
- **SARSA** learns about a policy that sometimes takes optimal actions (as estimated) and sometimes explores other actions (Estimation policy = Behavioral policy)
 - SARSA will learn to be careful in an environment where exploration is costly
- **Q-learning** learns about the policy that doesn't explore 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

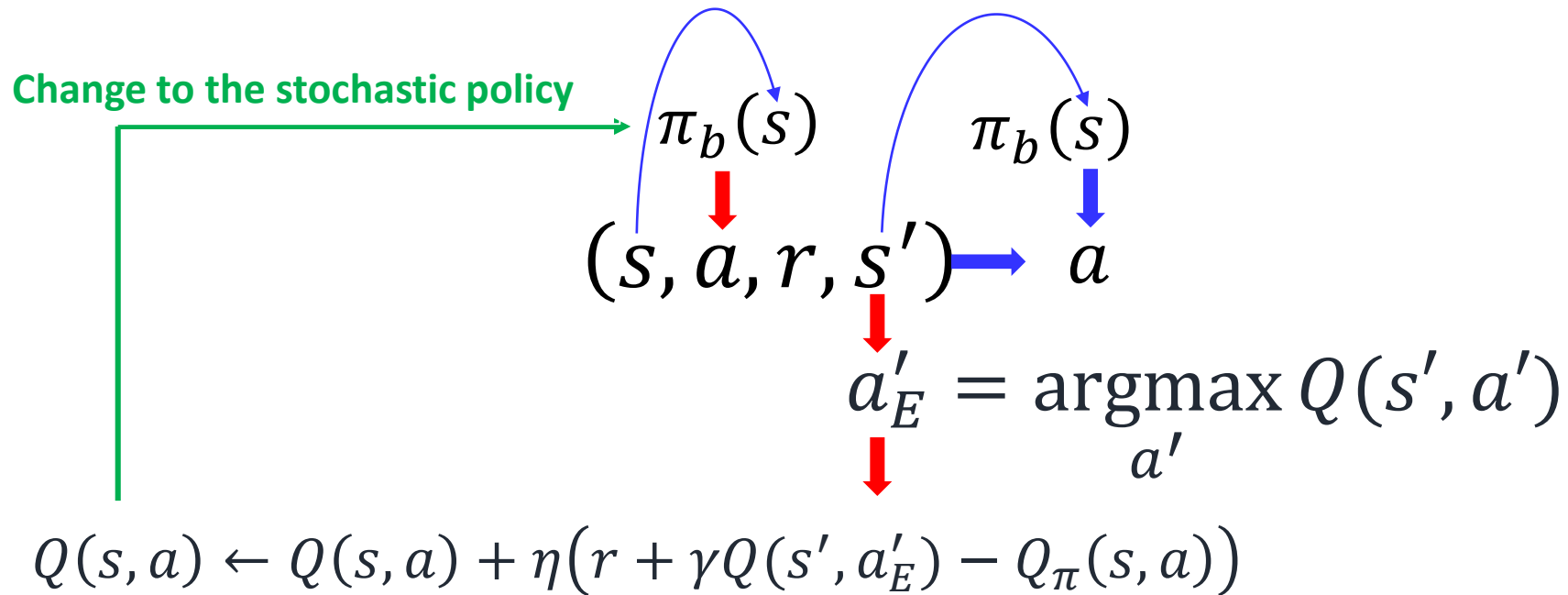
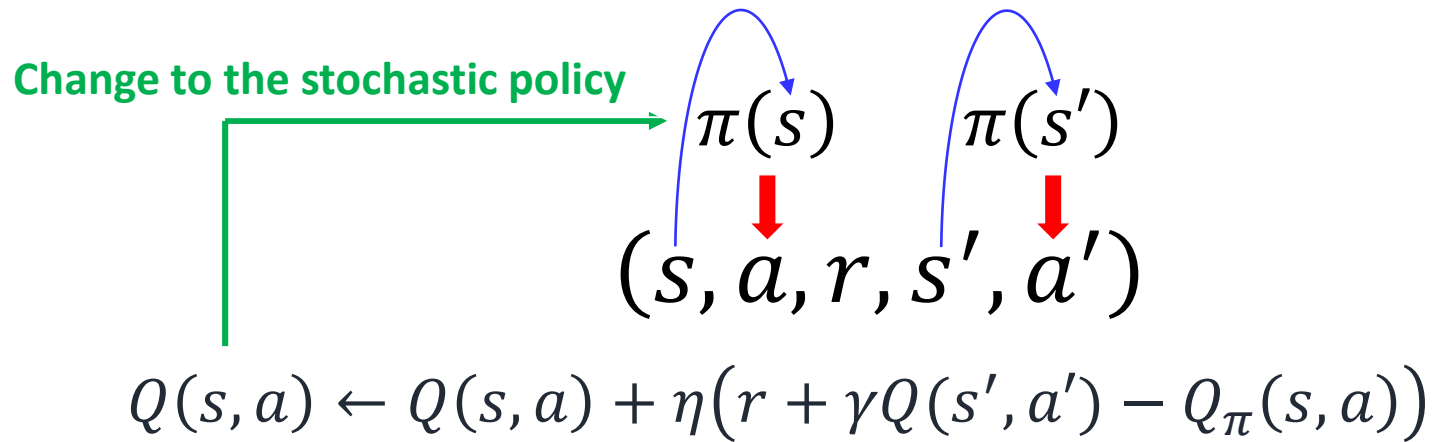
Why Q-learning is considered as Off-Policy method

- 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

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



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 with function approximation

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:

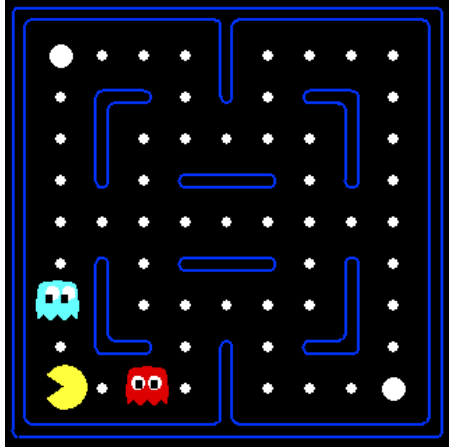
- Basic Q-Learning keeps a table of all q-values
- We cannot possibly learn about every single state!
 - Too many states to visit them all in training
 - Too many states to hold the q-tables in memory
- Doesn't generalize to unseen states/actions

Solution:

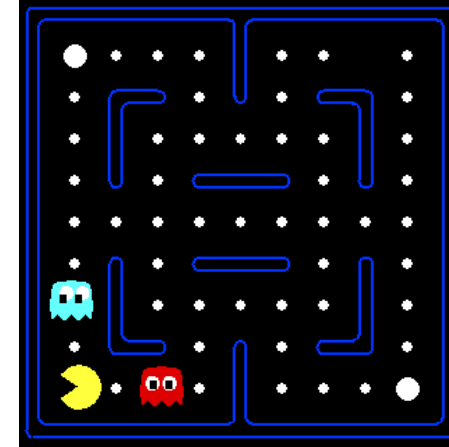
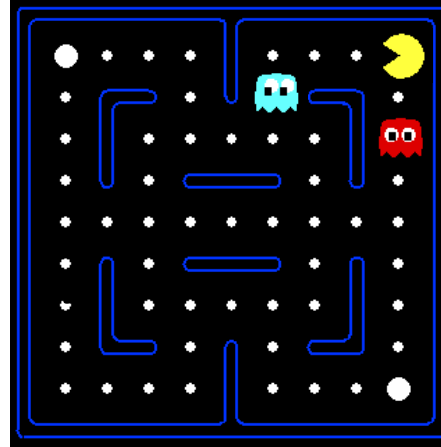
- Learn about some small number of training states from experience
- Generalize that experience to new, similar situations
 - fundamental idea in machine learning

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

Q-learning with function approximation

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

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

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

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

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

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)
- ϵ -greedy, train over 10M frames with 1M replay memory
- Human-level performance on some games (breakout)

Q-Learning with Neural Network

Screen size : 84×84 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

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

- 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)

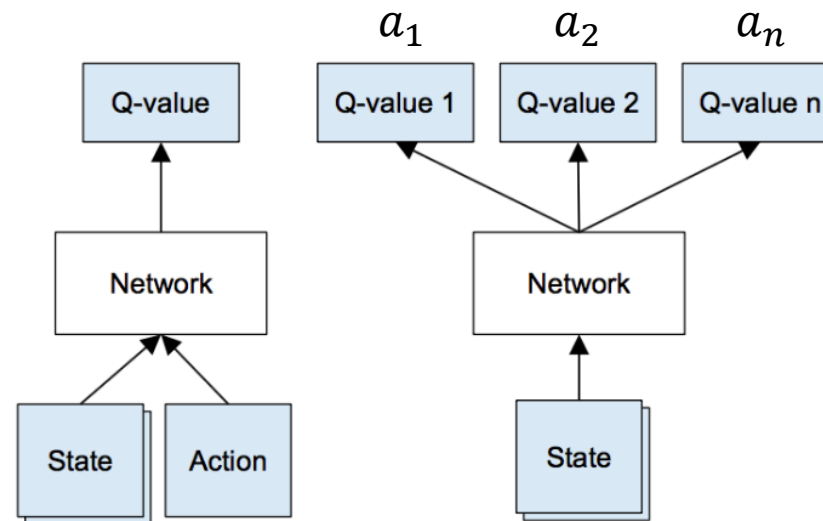


Figure 3: Left: Naive formulation of deep Q-network. Right: More optimized architecture of deep Q-network, used in DeepMind paper.

Summary

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