

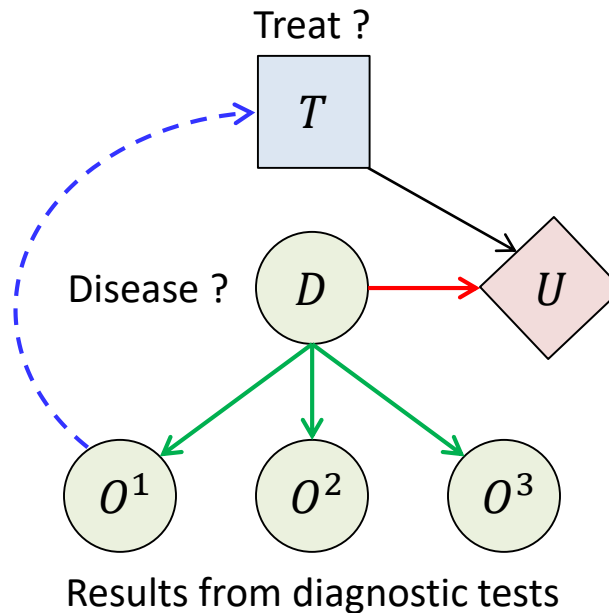
Final Exam Reviews




L11. Influential Diagram

Introduction

Bayesian Network + **Decision node** + **Utility node** = **Decision network (Influential Diagram)**

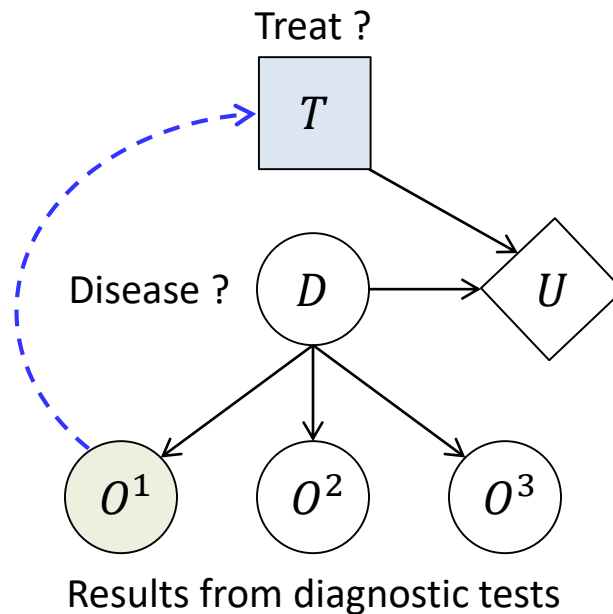
- make rational decisions based on a probabilistic model and utility function



-  **A chance node** corresponds to a random variable
-  **A decision node** corresponds to each decision to be made
-  **A utility node** corresponds to an additive utility component

Decision Network

Assume we only have a single observation $O^1 = 1 (= o_1^1)$ from test 1



$$\begin{aligned} EU(t^1|o_1^1) &= \sum_{o_3} \sum_{o_2} \sum_d P(d, o_2, o_3 | t^1, o_1^1) U(t^1, d, o_1^1, o_2, o_3) \\ &= \sum_d \underbrace{P(d | t^1, o_1^1)}_{\text{Can be computed using many inference methods}} U(t^1, d) \end{aligned}$$

Can be computed using many inference methods

Compare $EU(t^0|o_1^1)$ and $EU(t^1|o_1^1)$ and chose the treatment that lead maximum EU

Value of Information

- It may be beneficial to administer additional diagnostic tests to reduce the uncertainty about the disease. Then, how to choose a test type to be conducted?
- Expected utility of optimal action given observation o :

$$EU^*(o) = \operatorname{argmax}_a EU(a|o)$$

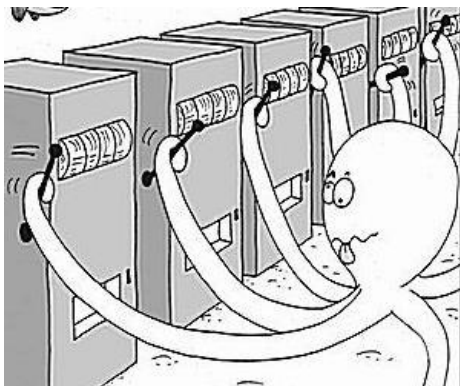
- The value of information (VPI) about new variable O^{new} (**unobserved**) given the current observation o (**observed**):

$$VOI(O^{new}|o) = \left(\sum_{o^{new}} P(o^{new}|o) EU^*(o^{new}, o) \right) - EU^*(o)$$

- The value of information about a variable is the increase in expected utility with the observation of that variable
- VPI can only capture the increase in expected utility → need to consider the cost associated with observing the new information

L12. Bandit Problem

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)

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 $1 - \epsilon$

Softmax Action Selection

Disadvantage in ϵ – *greedy* action selection:

- When it explores it chooses equally among all actions. That is it is as likely to chose the worst-appearing action as it is to choose the next-to best action

Solution:

- Vary the action probabilities as a graded function of estimated value
- The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their values estimates

Softmax action selection rule

It chooses action a on the t th play with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}}$$

- ✓ τ is a positive parameter called the temperature:
 - high $\tau \rightarrow$ all actions are equiprobable
 - low $\tau \rightarrow$ greater difference in selection probability for action
 - When $\tau = 0$, softmax selection rule become greedy one

Whether softmax action selection or greedy action selection rule is better is unclear and may depend on the task and on human factors (i.e., setting τ and ϵ)

Reinforcement Comparison

Reinforcement Comparison algorithm

- $p_t(a)$: The preference for each action at time t (not an actual action value)
- Action determination rule (soft max) :

$$\pi_t(a) = \frac{e^{p_t(a)/\tau}}{\sum_{b=1}^n e^{p_t(b)/\tau}}$$

- After selecting action and observing reward, the preference $p_t(a)$ is updated :

$$p_{t+1}(a_t) = p_t(a_t) + \beta[r_t - \bar{r}_t]$$

- ✓ High rewards should increase the probability of reselecting the action taken
- ✓ β is a positive step-size parameter
- ✓ The reference reward \bar{r}_t (i.e., average reward) is updated using all recently received rewards whichever actions were taken:

$$\bar{r}_{t+1} = \bar{r}_t + \alpha[r_t - \bar{r}_t]$$

Reinforcement comparison method can be very effective sometimes outperforming ϵ – *greedy* method

Relationship with model based approach

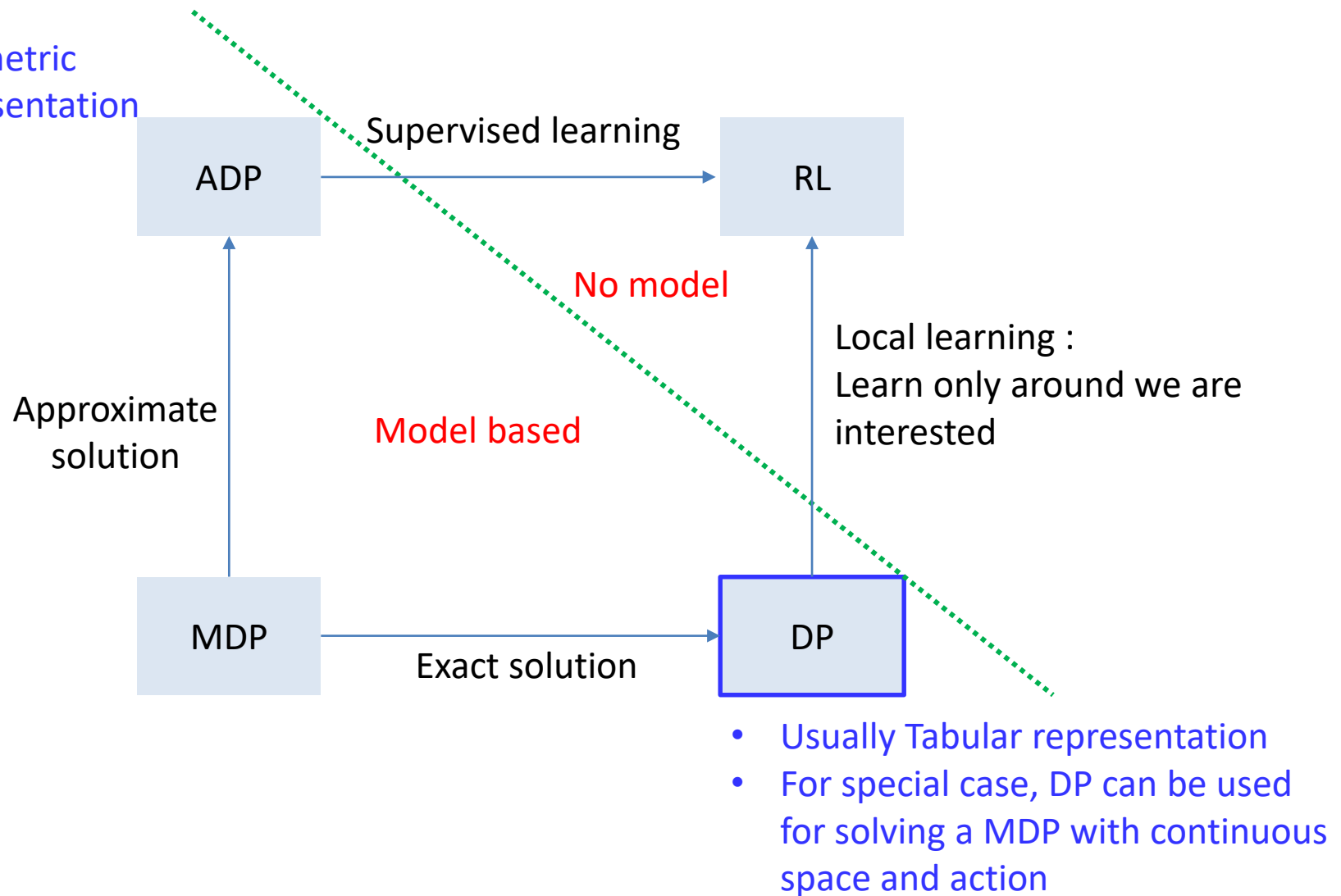
		Only Exploitation	Exploration vs Exploitation
		Model is known	Model is unknown
Optimum action	Single state	Optimization	Bandit
Optimum policy	Multiple state	Dynamic Programing	Contextual Bandit
			Reinforcement learning

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

L13. Markov Decision Process (Formulation)

From MDP to RL

Parametric
representation



- All other methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment

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)

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

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

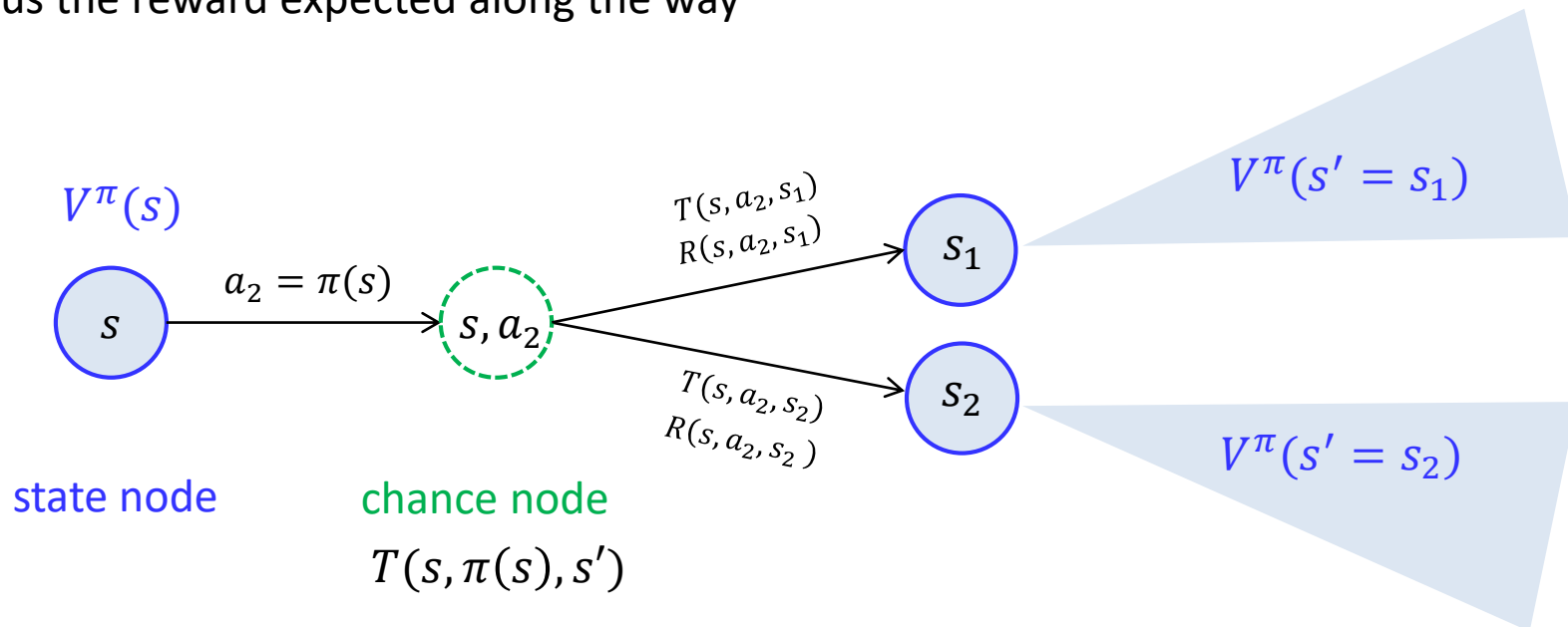
Because the agent can expect to receive in the future depend on what actions it will take
→ Value and Q functions are defined with respect to a particular policy mapping state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$

The Bellman Equation for Value Function

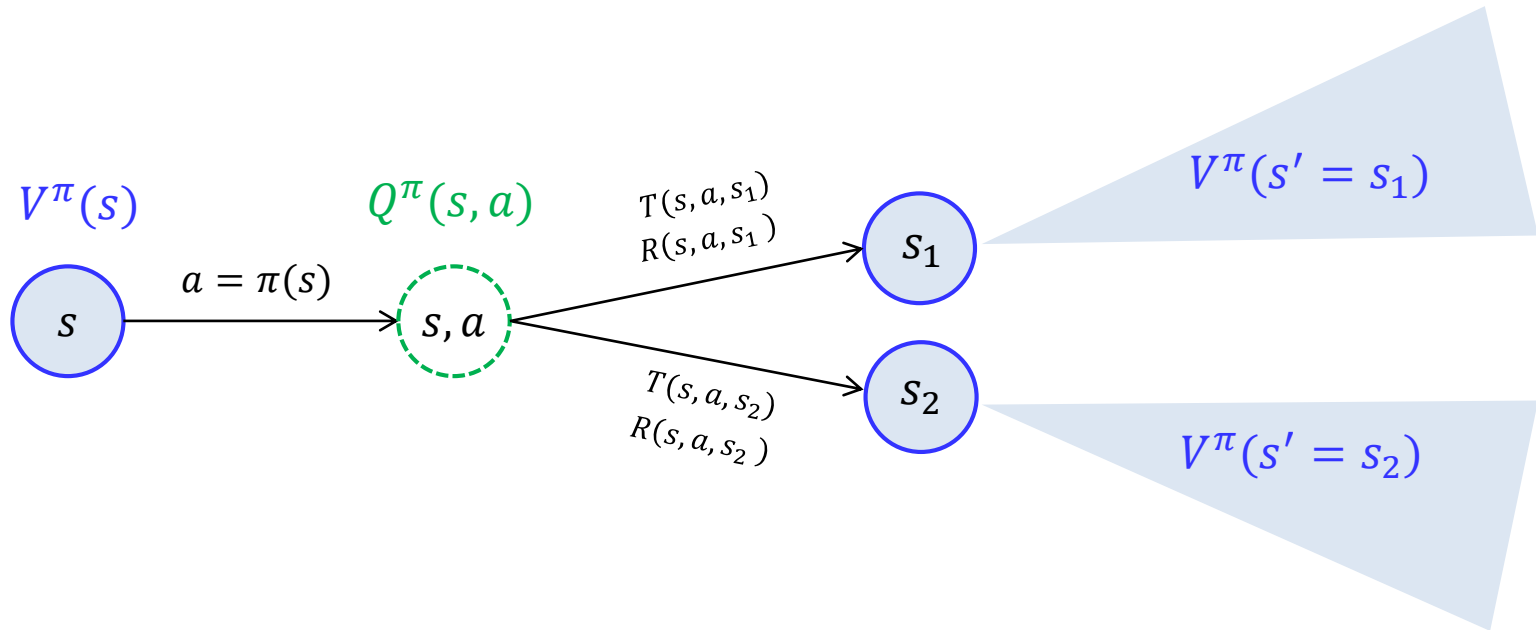
Recursive Formulation (The Bellman equation)

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

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



Summary for Value function and Q function



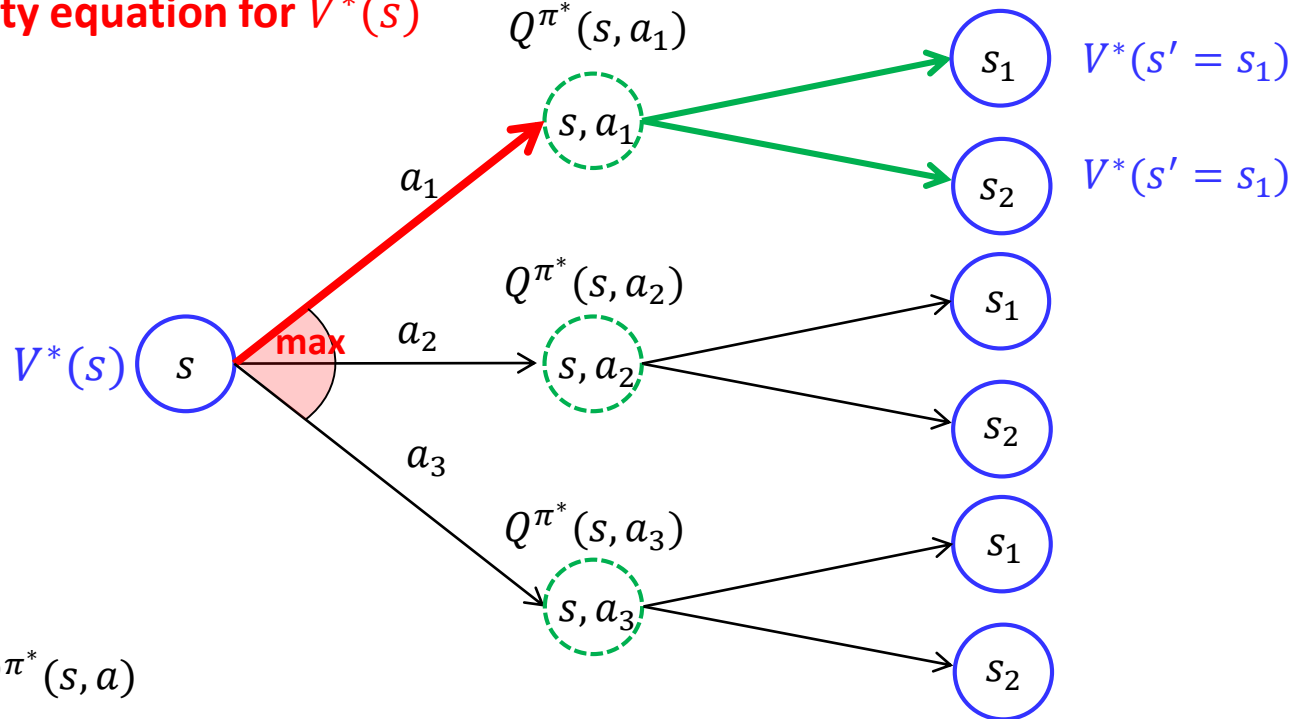
$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V^\pi(s')\}$$

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

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

Bellman Optimality Equation for State-Value Function

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



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

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

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

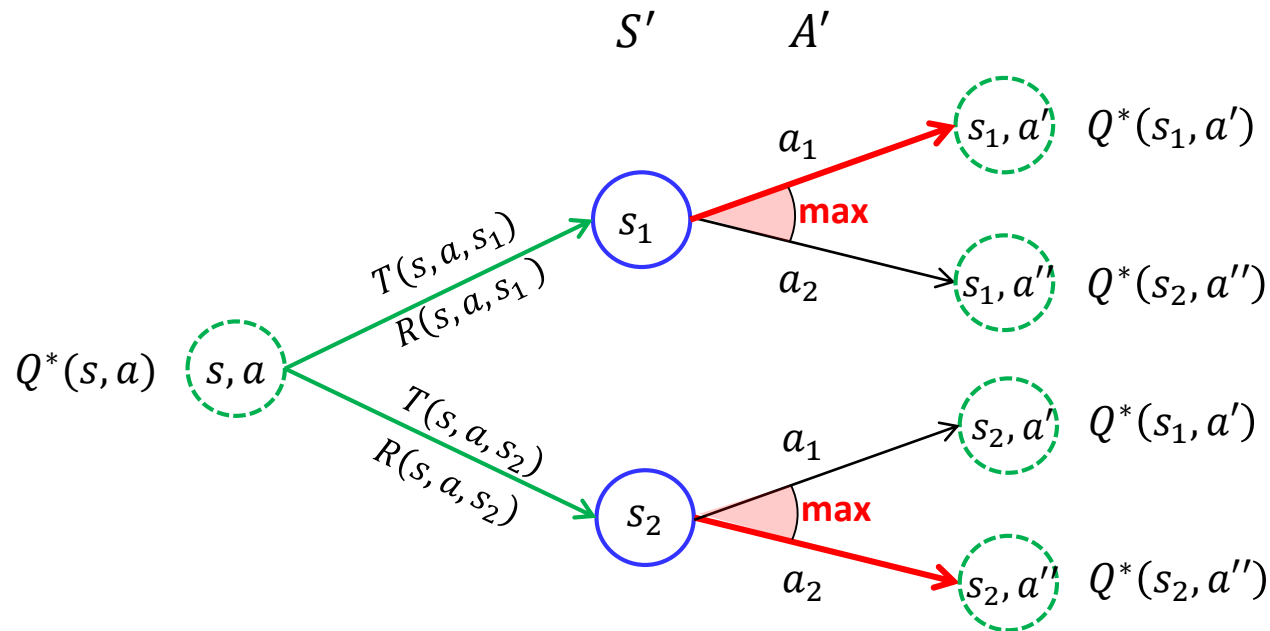
$$= \max_{a \in \mathcal{A}(s)} \mathbb{E} (r_t + \gamma V^*(s_{t+1}) \mid S_t = s, A_t = a) \quad \mathbb{E} \text{ is over transitions}$$

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

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

L14. Markov Decision Process (Dynamic Programming Approach)

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

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

DP Approaches

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

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

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

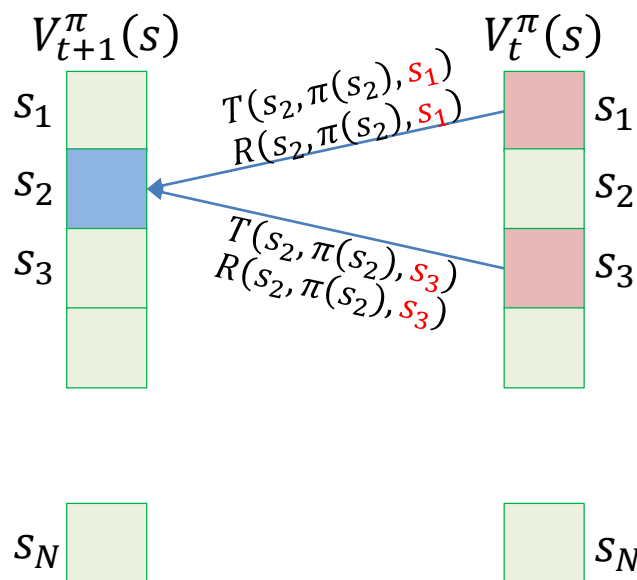
Policy Evaluation

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

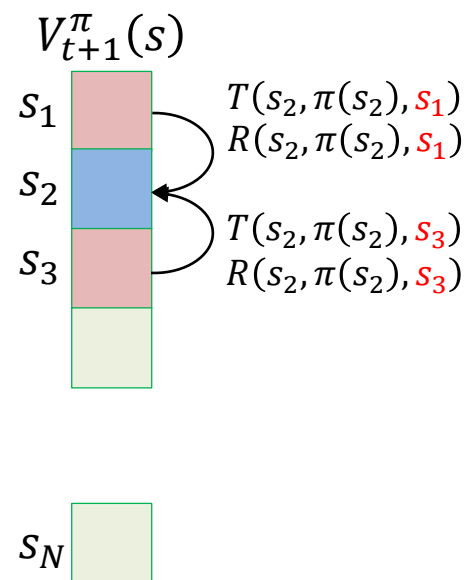
Example:

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

“Two-arrays” update



“In place” update



Usually faster!
Less memory

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

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

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

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

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

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

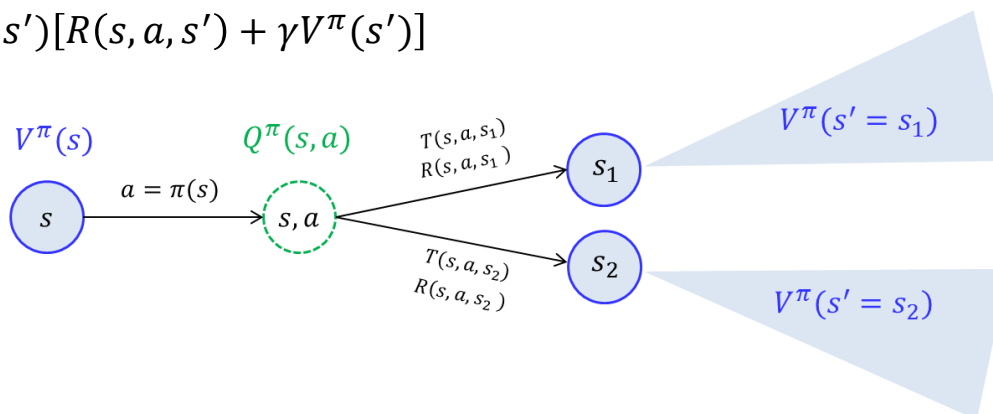
Thus, $\pi \leq \pi'$

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

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 :

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

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 :

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

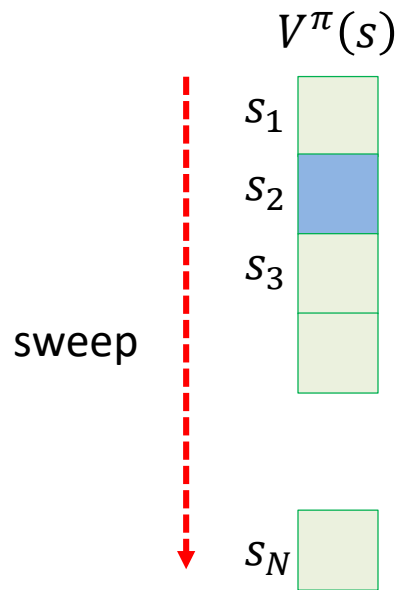
Until $\max_{s \in S} |V_t(s) - V_{t-1}(s)| \leq e$

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

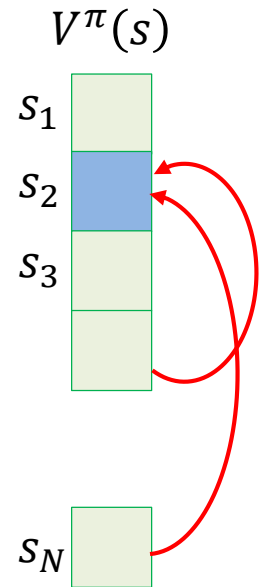


Conventional DP Algorithms

- Black mon game has 10^{20} states
- Go game has $3^{(19 \times 19)}$ states
- ...



- Take forever to sweep all states
- Does not improve policy until value functions are full backed up



Asynchronous DP Algorithms

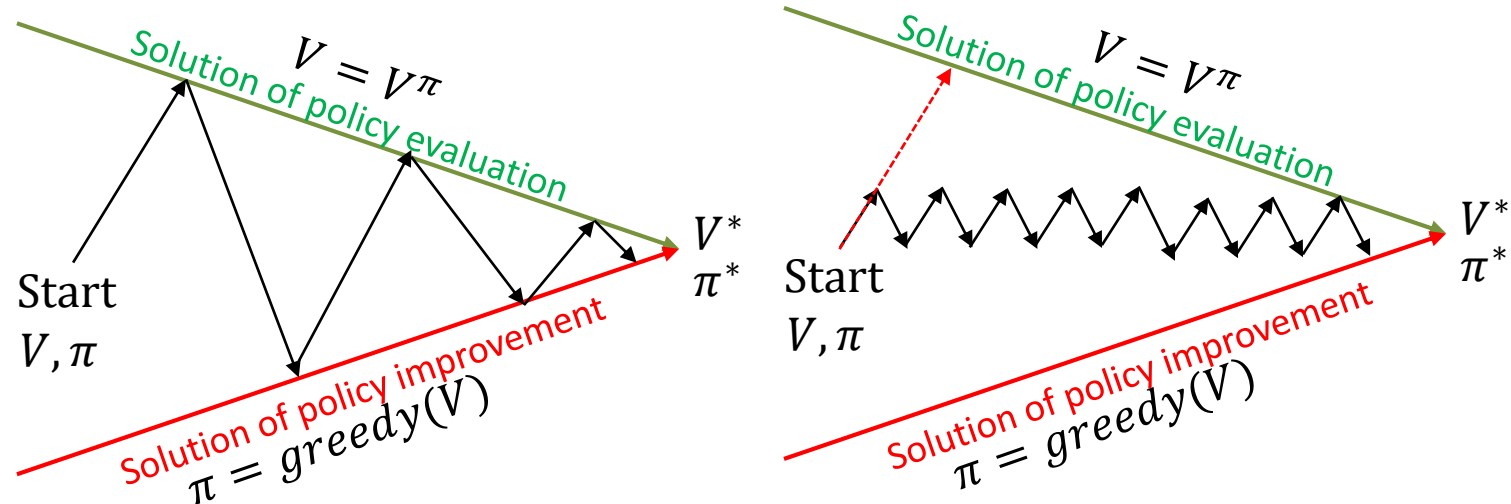
- Back up the values of states in any order whatsoever, using whatever values of other states happen to be available
- Allow great flexibility in selecting states to which backup operations are applied
- Make it easier to intermix computation with real-time interaction: To solve a given MDP, we can run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

Generalized Policy Iteration

Generalized Policy Improvement (GPI)

Dynamic Programming
“full backup”

Asynchronous
Dynamic Programming



Asynchronous Dynamic Programming is a core concept in Reinforcement learning

L15. Reinforcement Learning (Monte Carlo Methods)



Markov Decision Process (Offline)

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

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



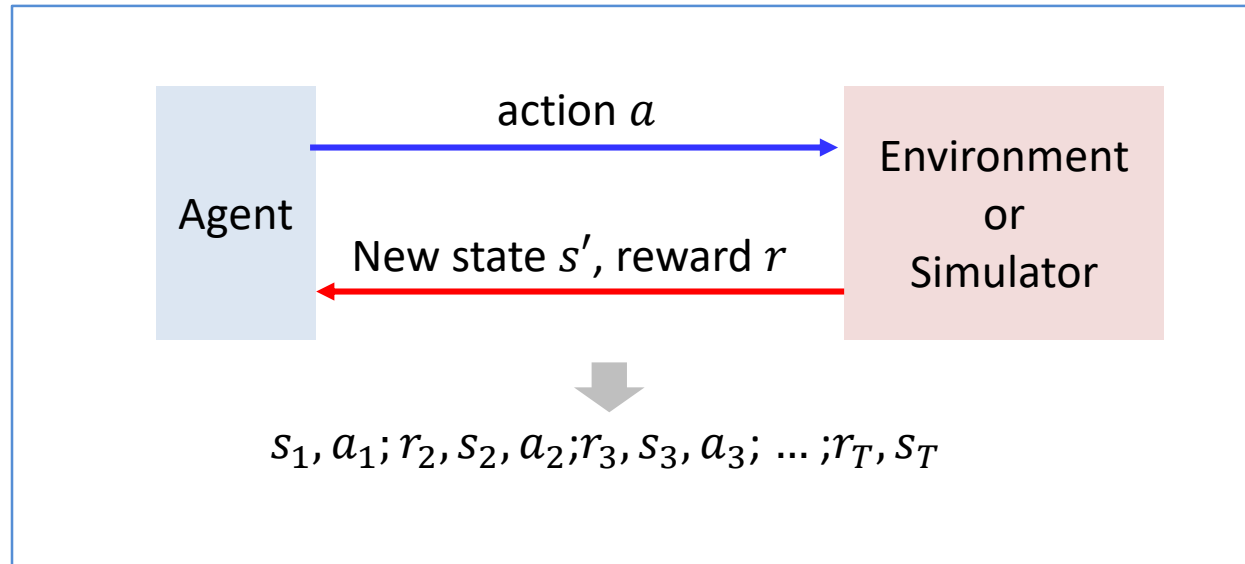
Reinforcement Learning (Offline & Online)

- 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

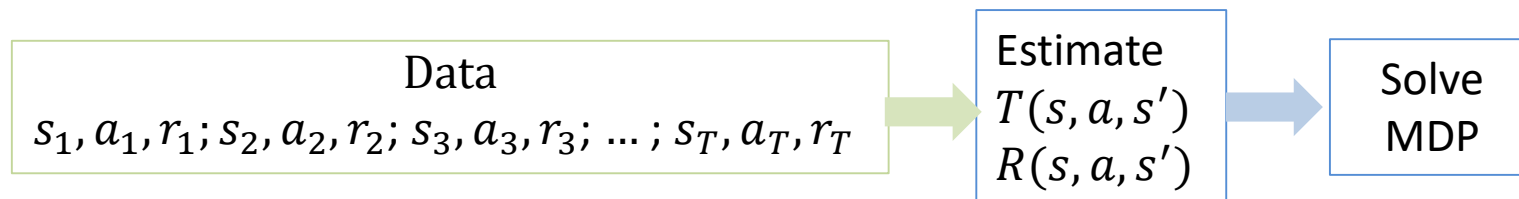
Receive reward r_{t+1} and observe new state s_{t+1} (Environment)

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

- **Monte Carlo Method (Sutton & Barto Ch.5)**
 - Model-Based Monte Carlo method
 - Model-free Monte Carlo method
 - Policy Evaluation
 - Policy Improvement
 - Policy Iteration (Monte Carlo control)
 - ✓ On-policy
 - ✓ Off-policy
- **Temporal Difference Learning (Sutton & Barto Ch.6)**
 - SARSA
 - Q-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

Model-Free Monte Carlo Based Methods

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

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

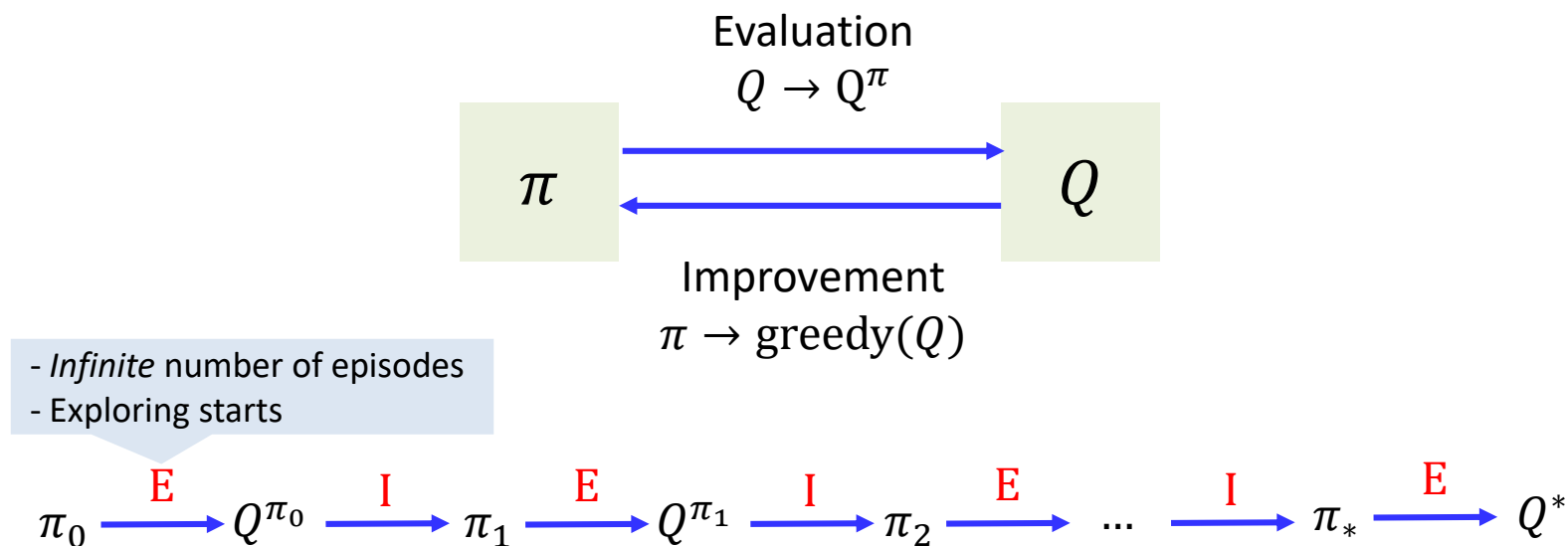
First visit to s

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

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

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

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

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

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

Key Idea : Off-policy algorithm

- Follows the behavior policy while learning about and improving the estimation policy
- *Behavior* policy $\pi'(s, a)$
 - ✓ The policy used to generate behavior
 - ✓ Requires that the behavior policy have a nonzero probability of selecting all actions that might be selected by the estimation policy (e.g., ϵ – *soft* policy)
- *Estimation* policy $\pi(s, a)$
 - ✓ The policy that is evaluated and improved
 - ✓ π can be deterministic
 - ✓ π can be the greedy policy with respect to Q (an estimation Q^π)

Disadvantages

→ Learning can be slow

Model-Free Monte Carlo Based Methods

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

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

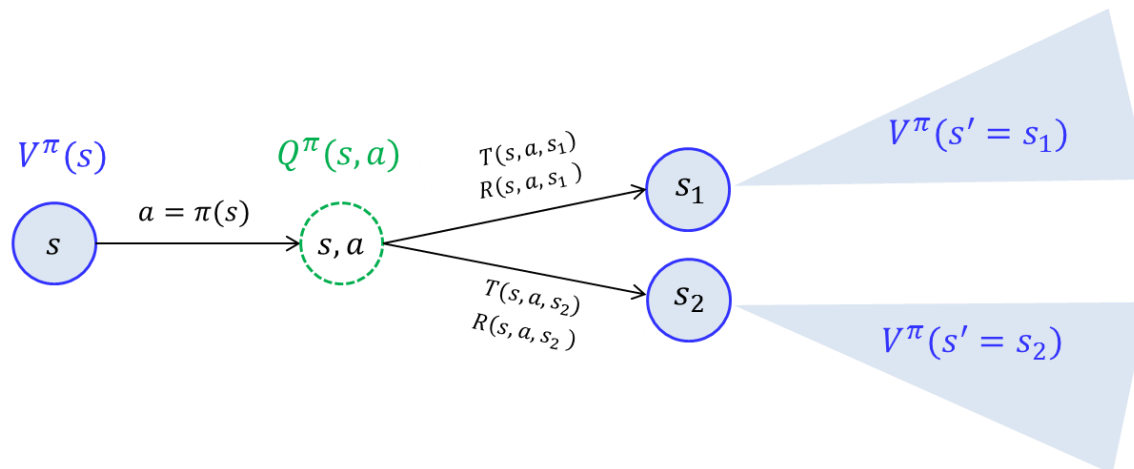
On-Policy TD Control (SARSA)

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



Temporal Difference Policy Evaluation

$$V^\pi(s) = \mathbb{E}_\pi(U_t | s_t = s)$$

$$= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right)$$

$$= \mathbb{E}_\pi\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\right)$$

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

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

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

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

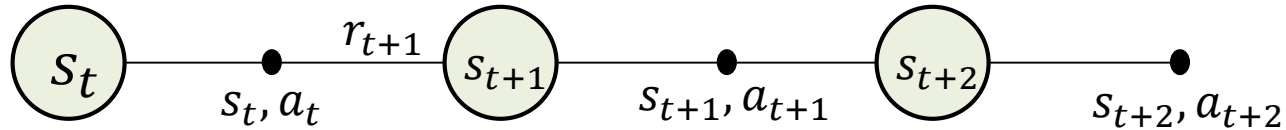
Recall:

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

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

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

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

Model-Free Monte Carlo Based Methods

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

• Episodic based

• Single-data-point based

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):

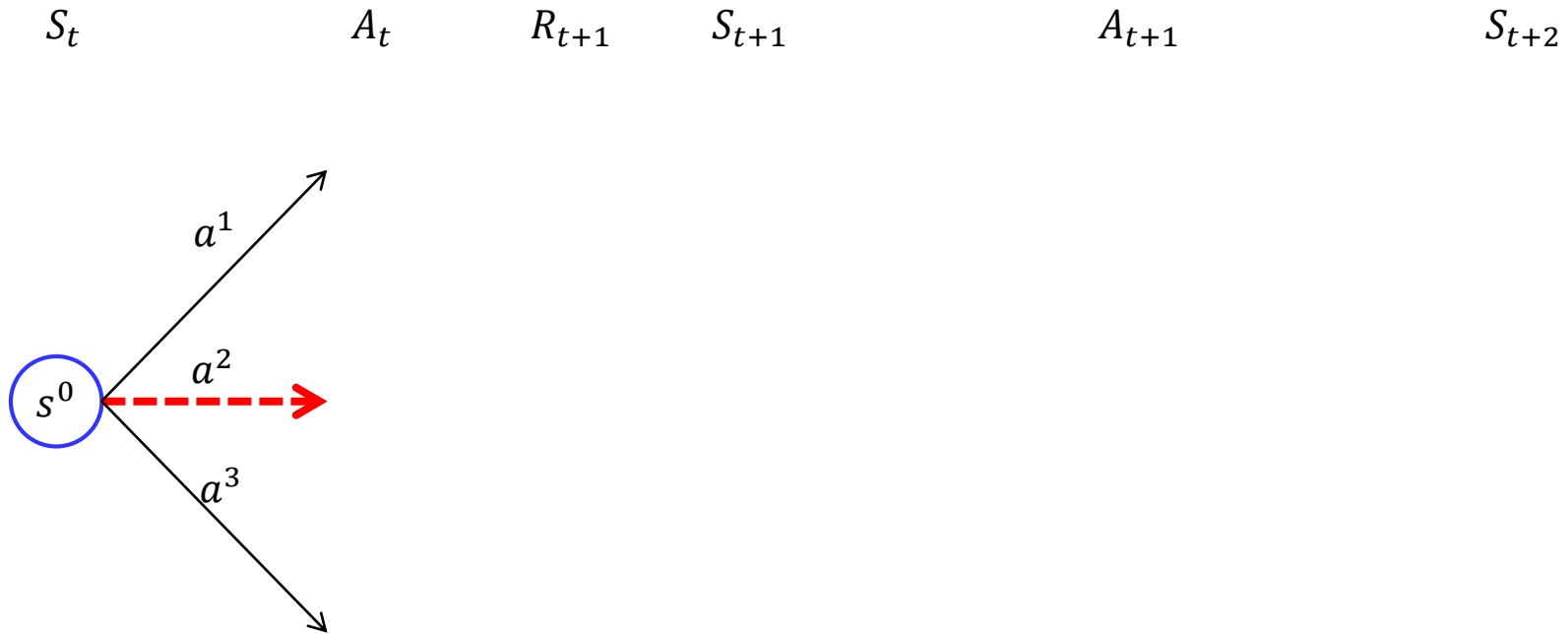
$\left\{ \begin{array}{l} \text{Take action } a \text{ given } s, \text{ observe } r, s' \\ \text{Choose } a' \text{ from } s' \text{ using policy derived from } Q \text{ (e.g., } \epsilon - greedy) \\ Q_{\pi}(s, a) \leftarrow Q_{\pi}(s, a) + \eta(r + \gamma Q_{\pi}(s', a') - Q_{\pi}(s, a)) \\ s \leftarrow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

Behavioral policy
||
Estimation policy

Until s is terminal

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

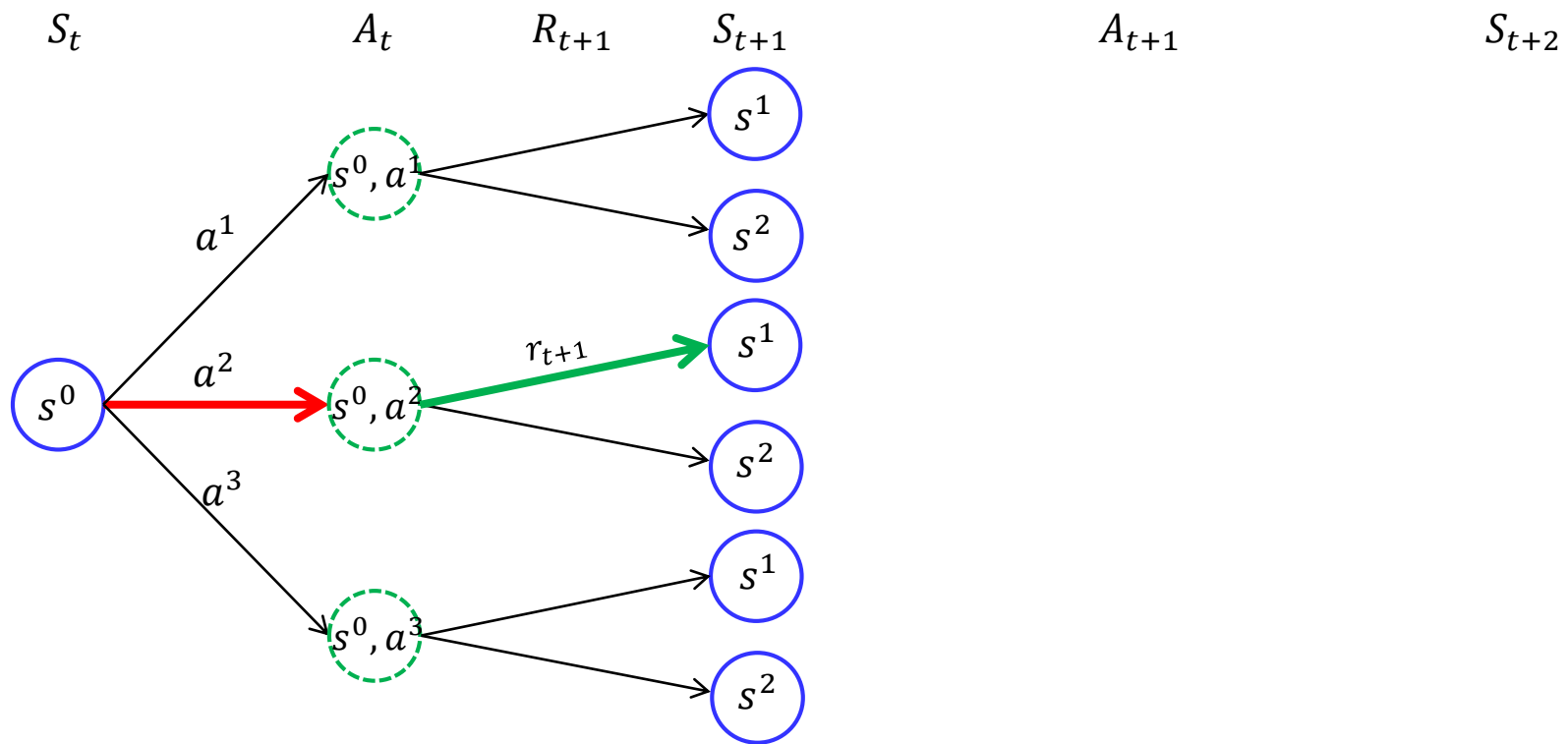


Choose a_t from $s_t = s^0$ using current Q

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

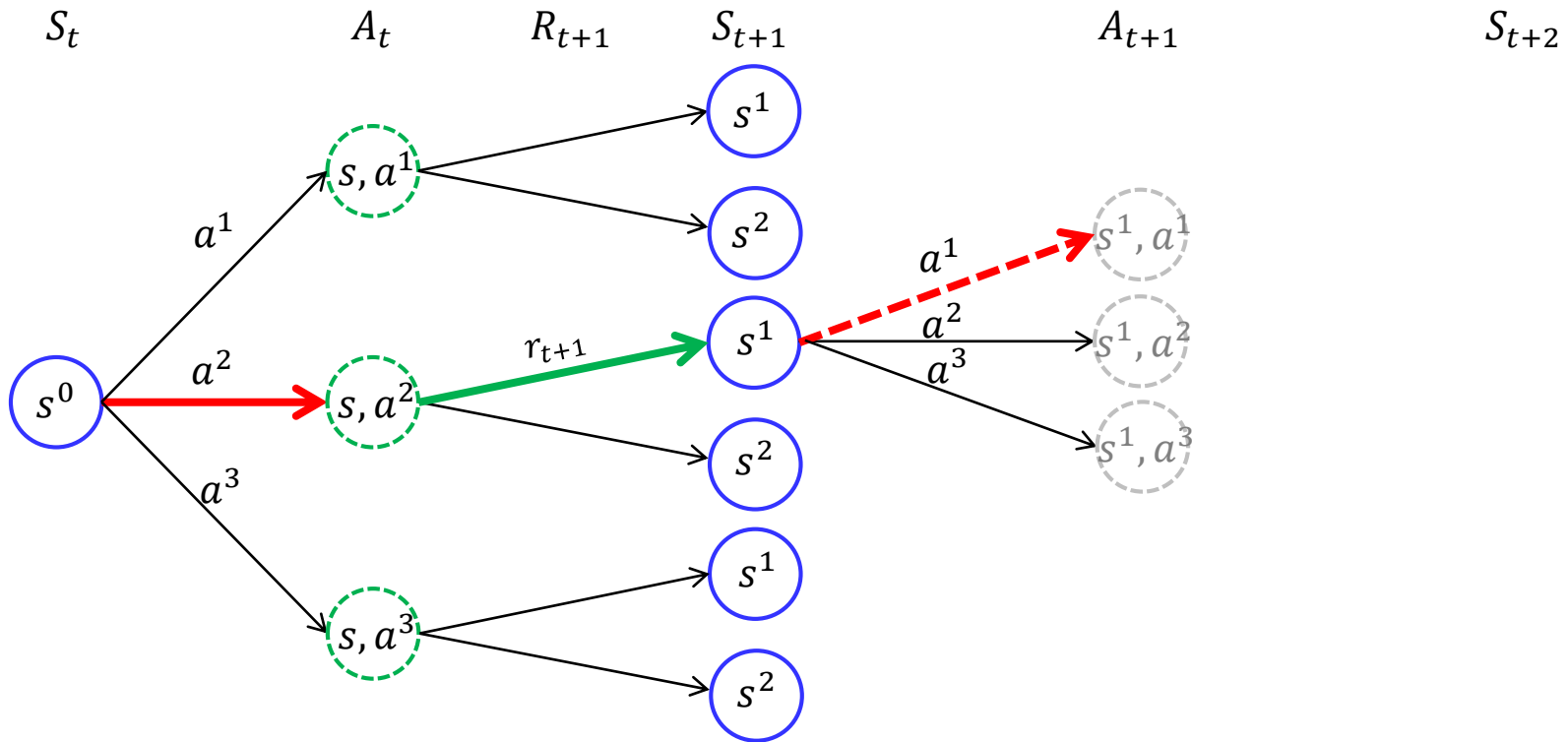
Assume a^2 is chosen

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

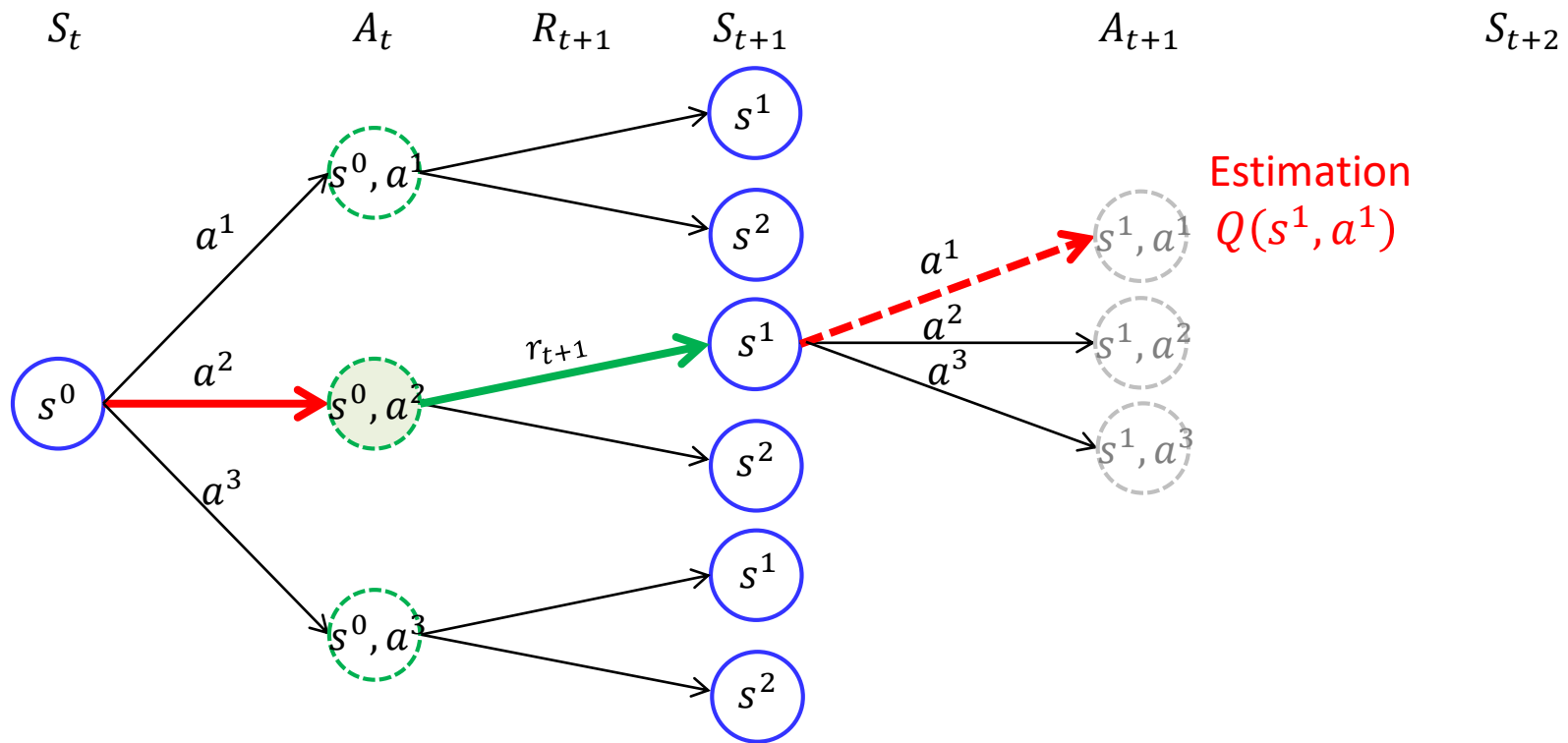


Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

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

Assume a^1 is chosen

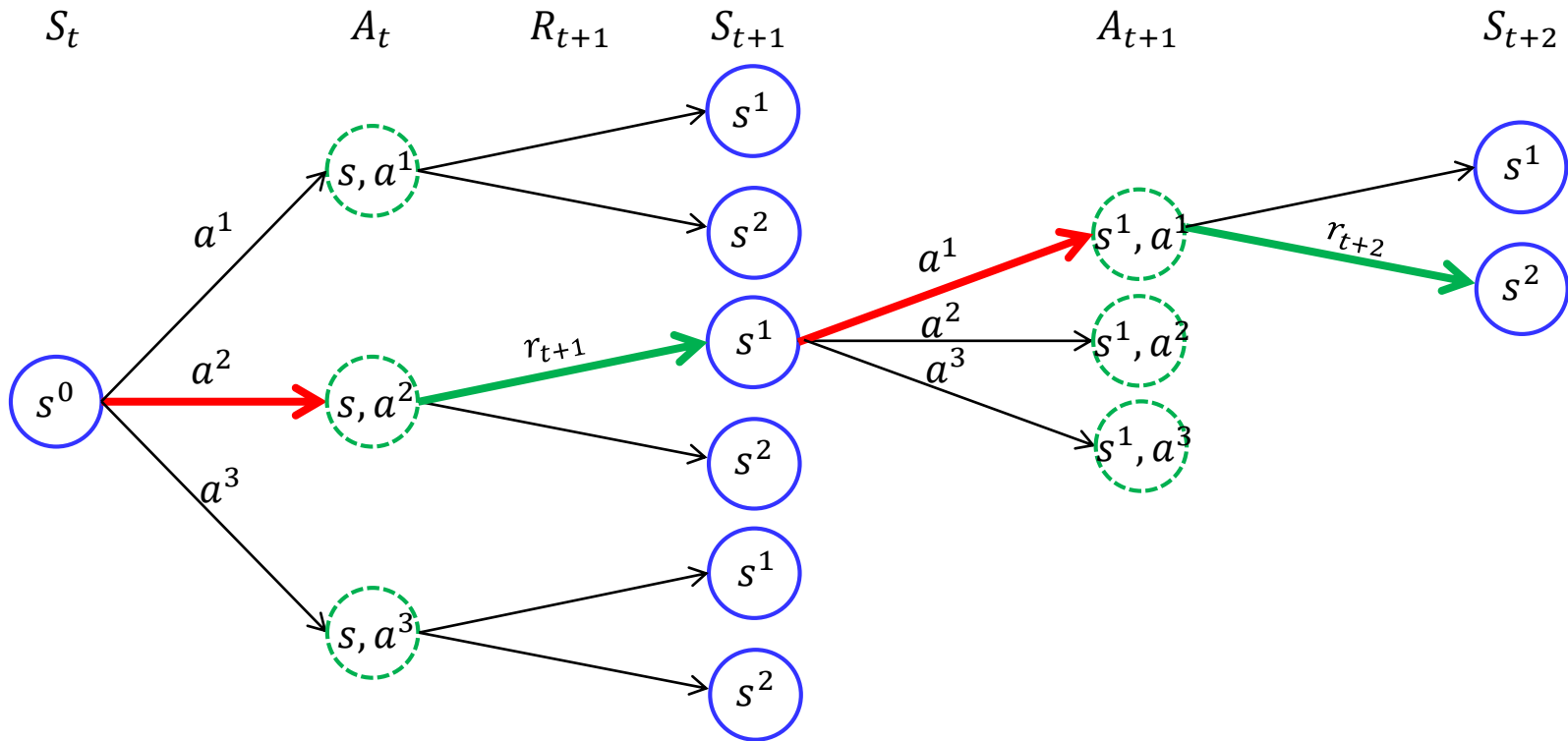
Sarsa: On-Policy TD Control



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

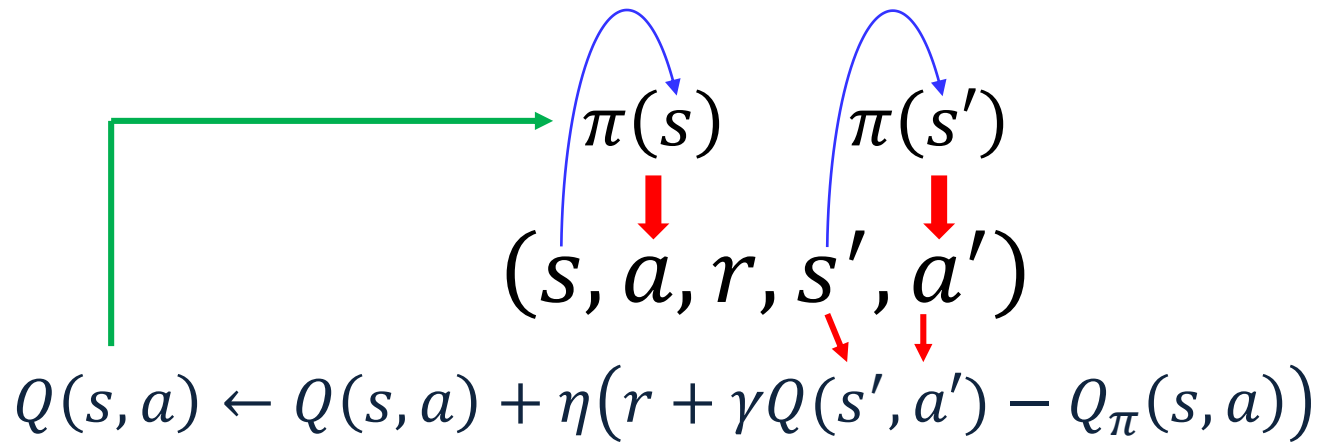
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$
$$\rightarrow Q(s^0, a^2) \leftarrow Q(s^0, a^2) + \alpha[r_{t+1} + \gamma Q(s^1, a^1) - Q(s^0, a^2)]$$

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



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

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

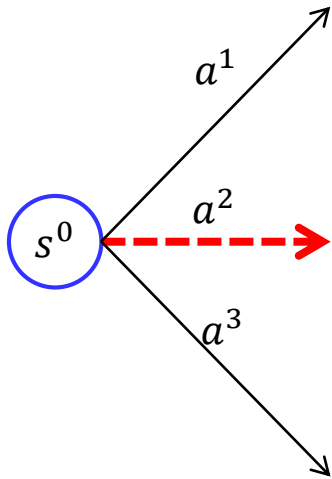
(Always try to estimate the optimal policy)

-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

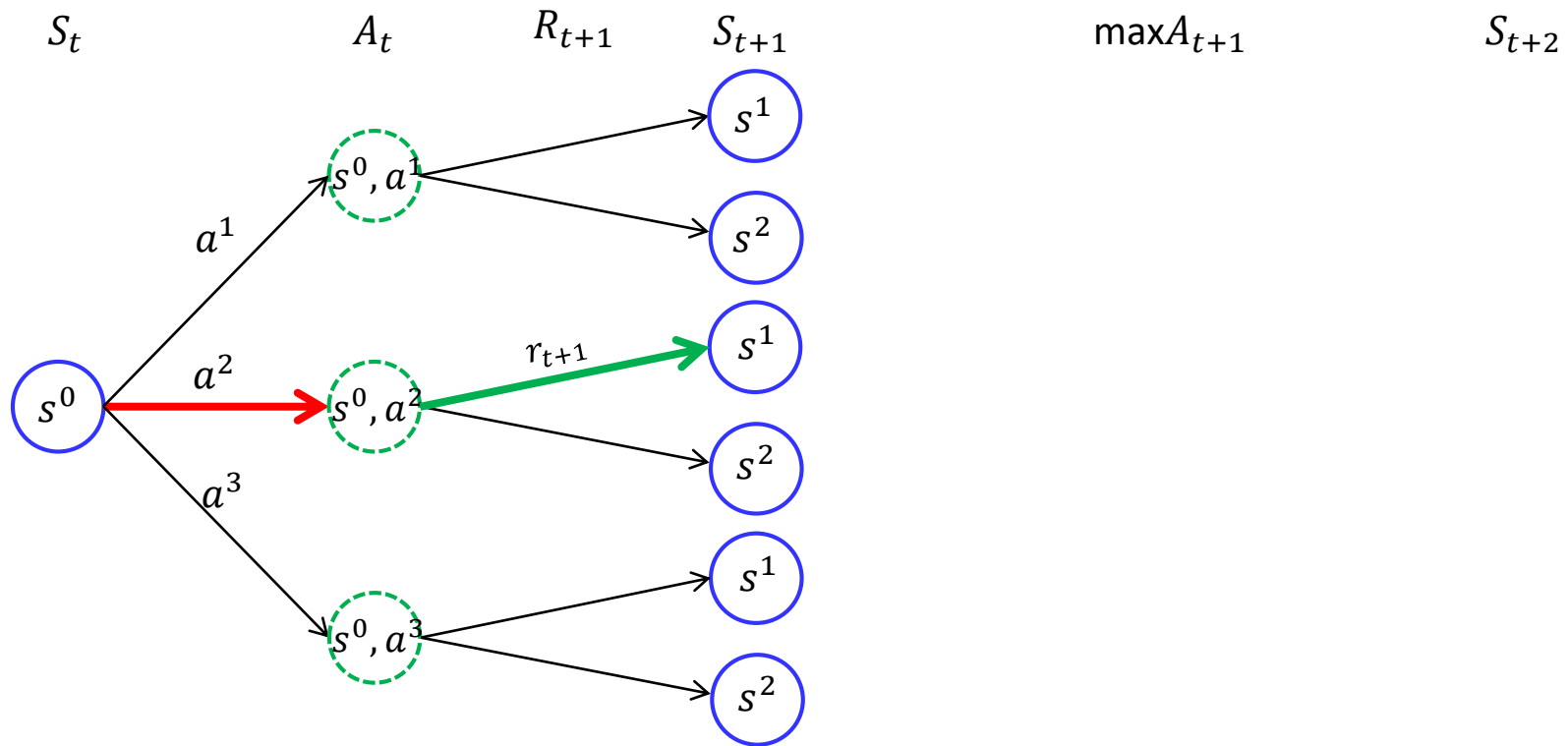
 s_t A_t R_{t+1} s_{t+1} $\max A_{t+1}$ s_{t+2} 

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

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

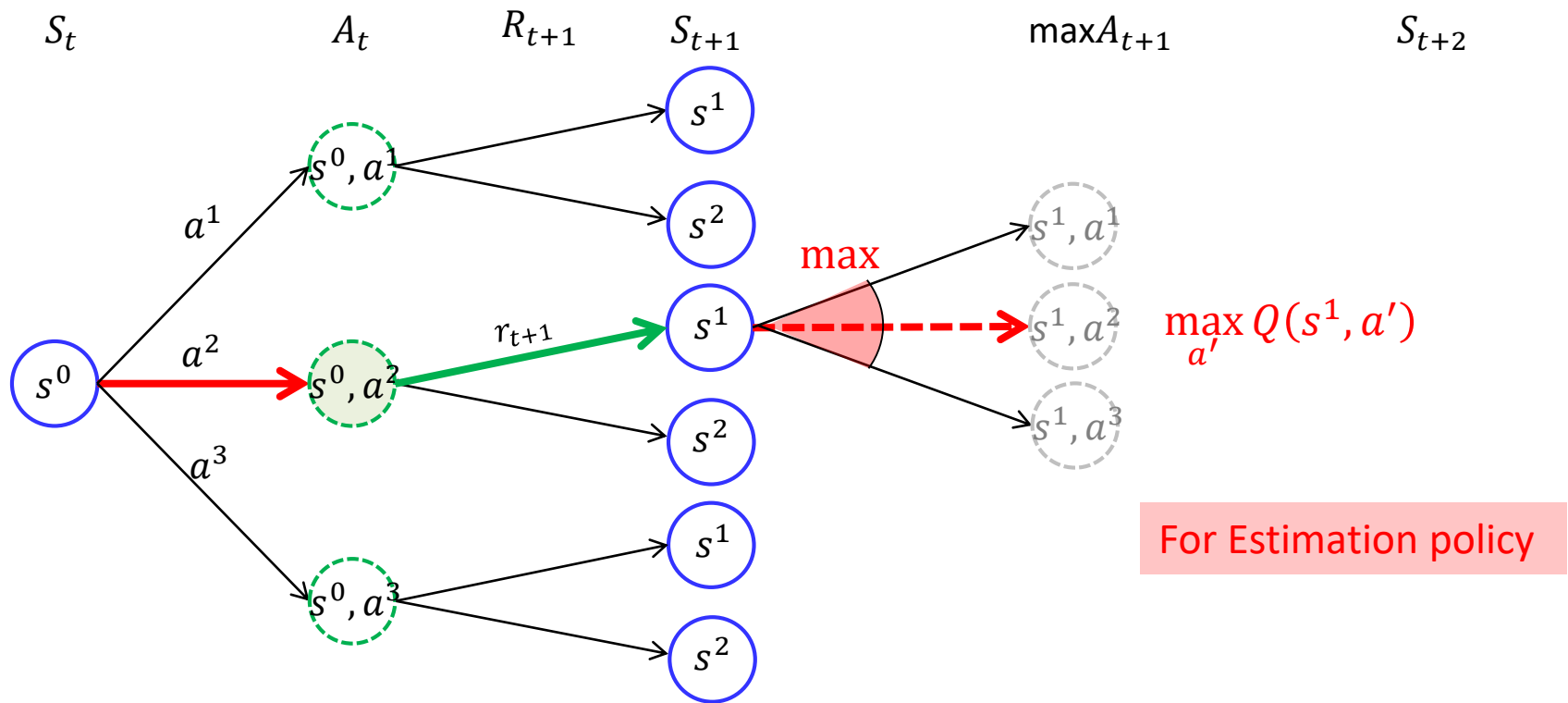
Assume a^2 is chosen

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

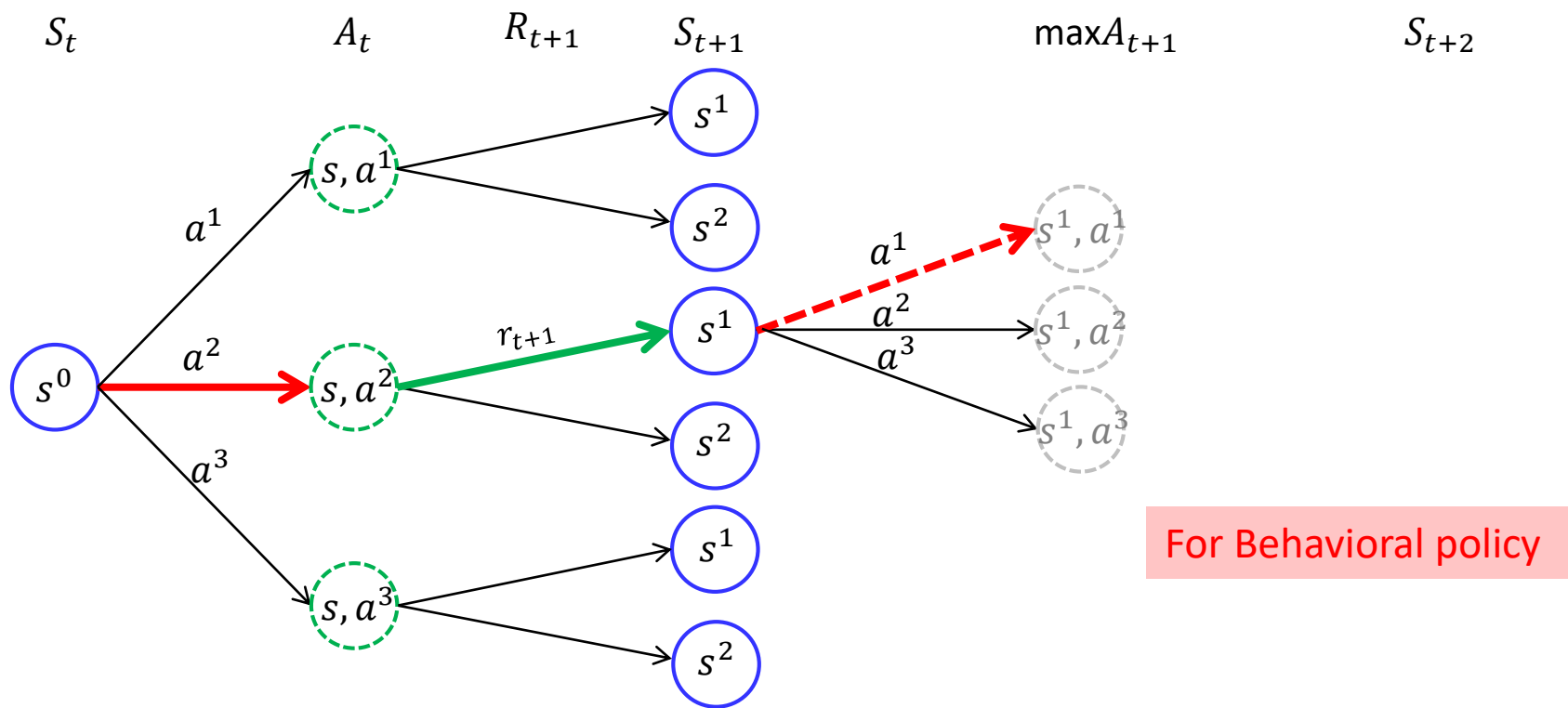


Update Q function with the $\max_{a'} Q(s^1, a')$

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

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

Q-Learning: Off-Policy TD Control

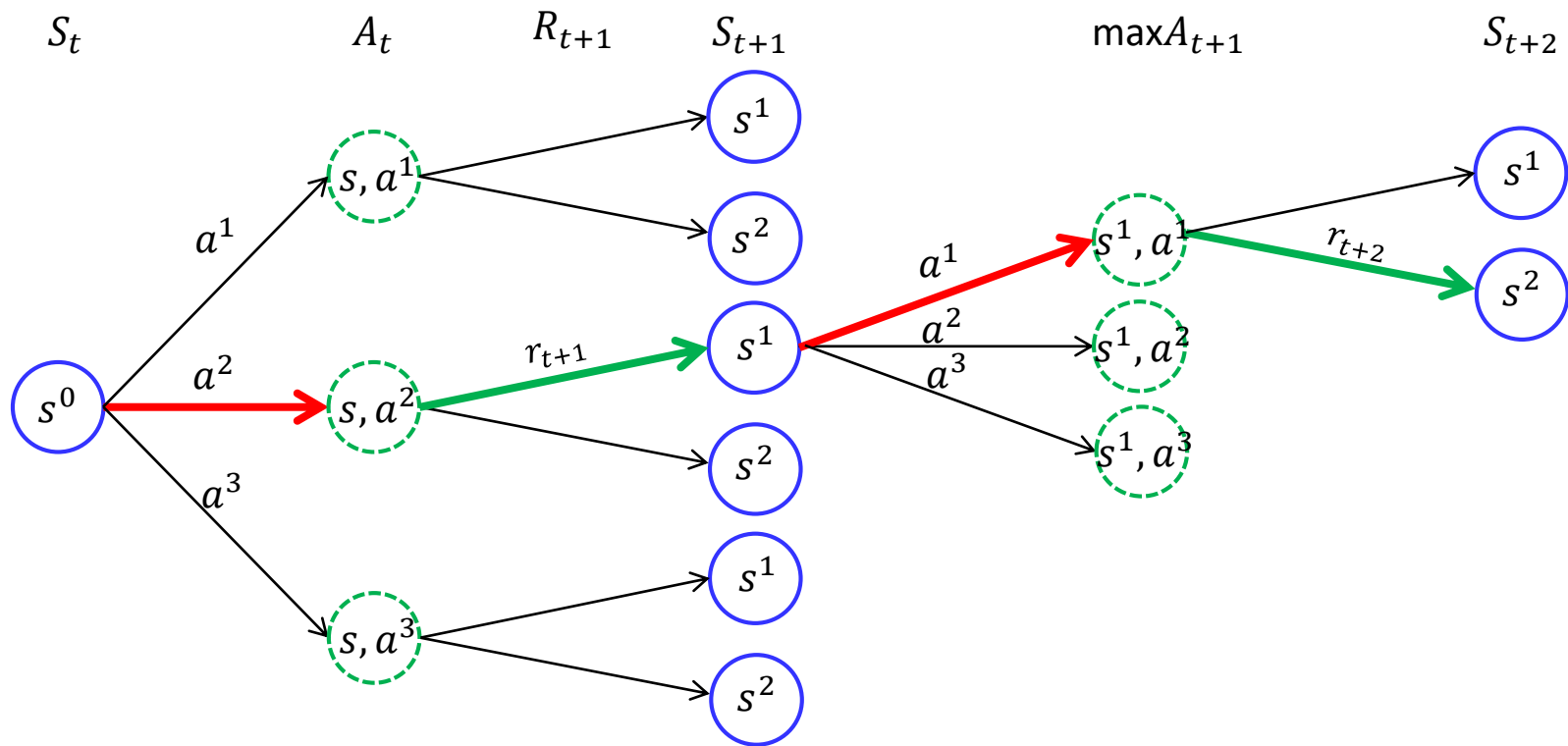


Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

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

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 $s_{t+2} = s^2$