
PATTERN RECOGNITION AND MACHINE LEARNING

CHAPTER 15: REINFORCEMENT LEARNING

Learning Objectives

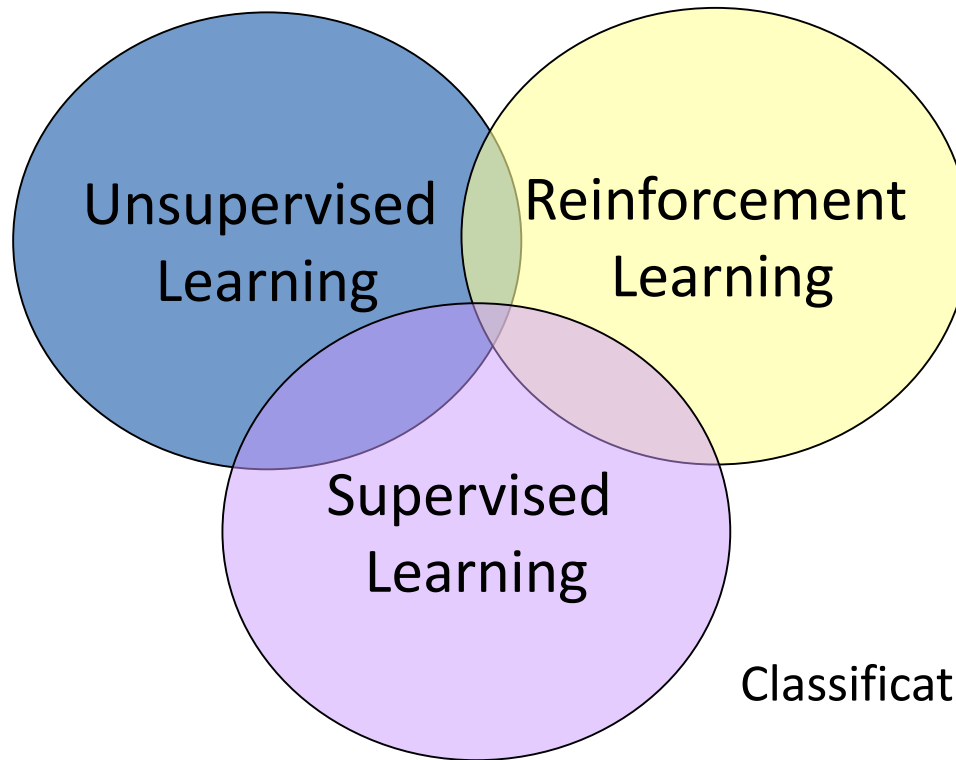
- 1、 What is reinforcement learning (RL)?
 - 2、 What are dynamic programming approaches to RL?
 - 3、 What are Monte Carlo approaches to RL?
 - 4、 What are Temporal Difference approaches to RL?
 - 5、 What is Bayesian Reinforcement Learning?
 - 6、 What are cost functions for deep reinforcement learning?
 - 7、 What is value approximation?
 - 8、 What is policy approximation?
-

Outlines

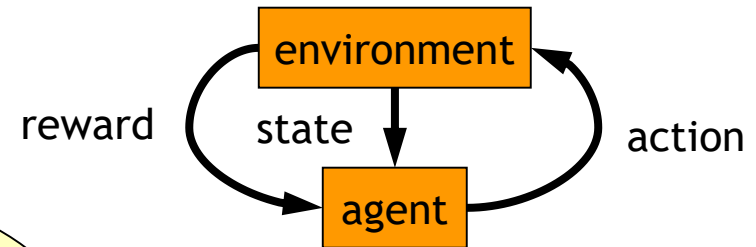
- Reinforcement Learning Introduction
 - Dynamic Programming Approaches
 - Monte Carlo Approaches
 - Temporal Difference Approaches
 - Bayesian Reinforcement Learning
 - Deep Reinforcement Learning
-

Machine Learning

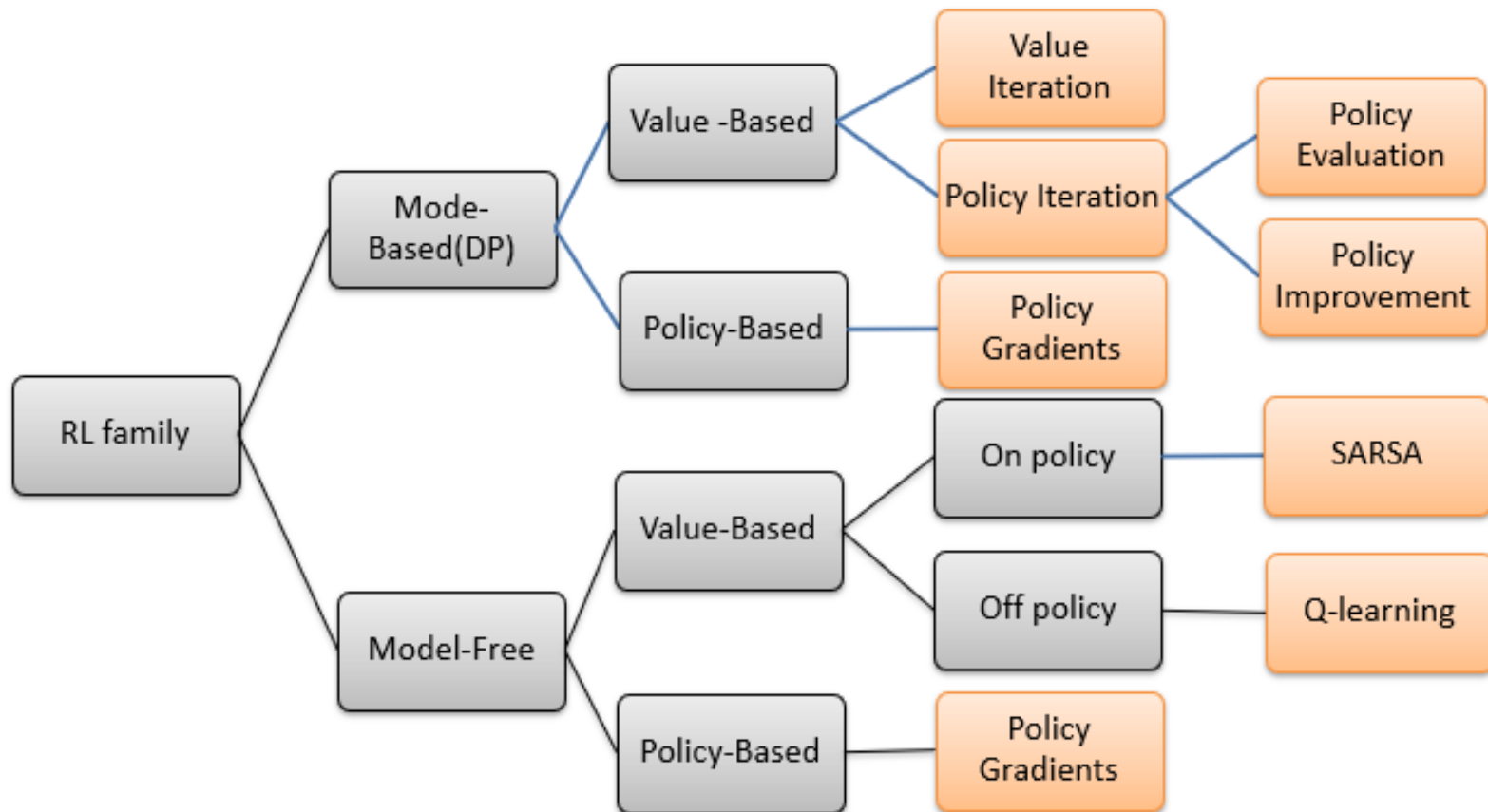
Clustering
Dimension reduction



Classification, Regression



Reinforcement Learning



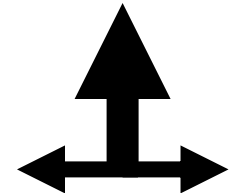
Reinforcement Learning Problem

| | | | |
|-------|----------|--|----|
| | | | +1 |
| | Obstacle | | -1 |
| START | | | |

actions: UP, DOWN, LEFT, RIGHT

Policy

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]

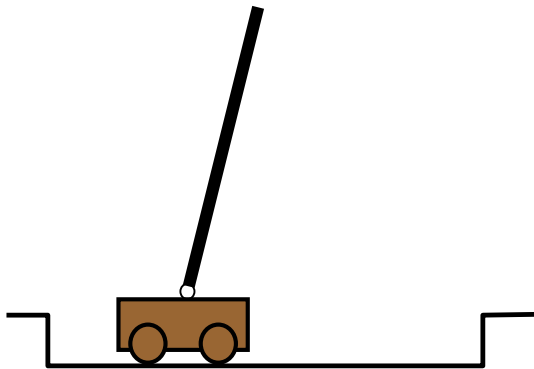
reward -0.04 for each step

what's the strategy to achieve max reward?

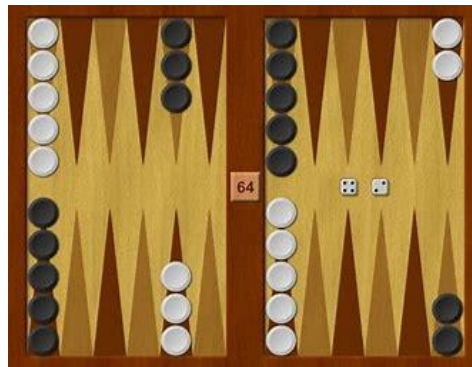
what if the actions were deterministic?

Reinforcement Learning Problems

Pole balancing



Games



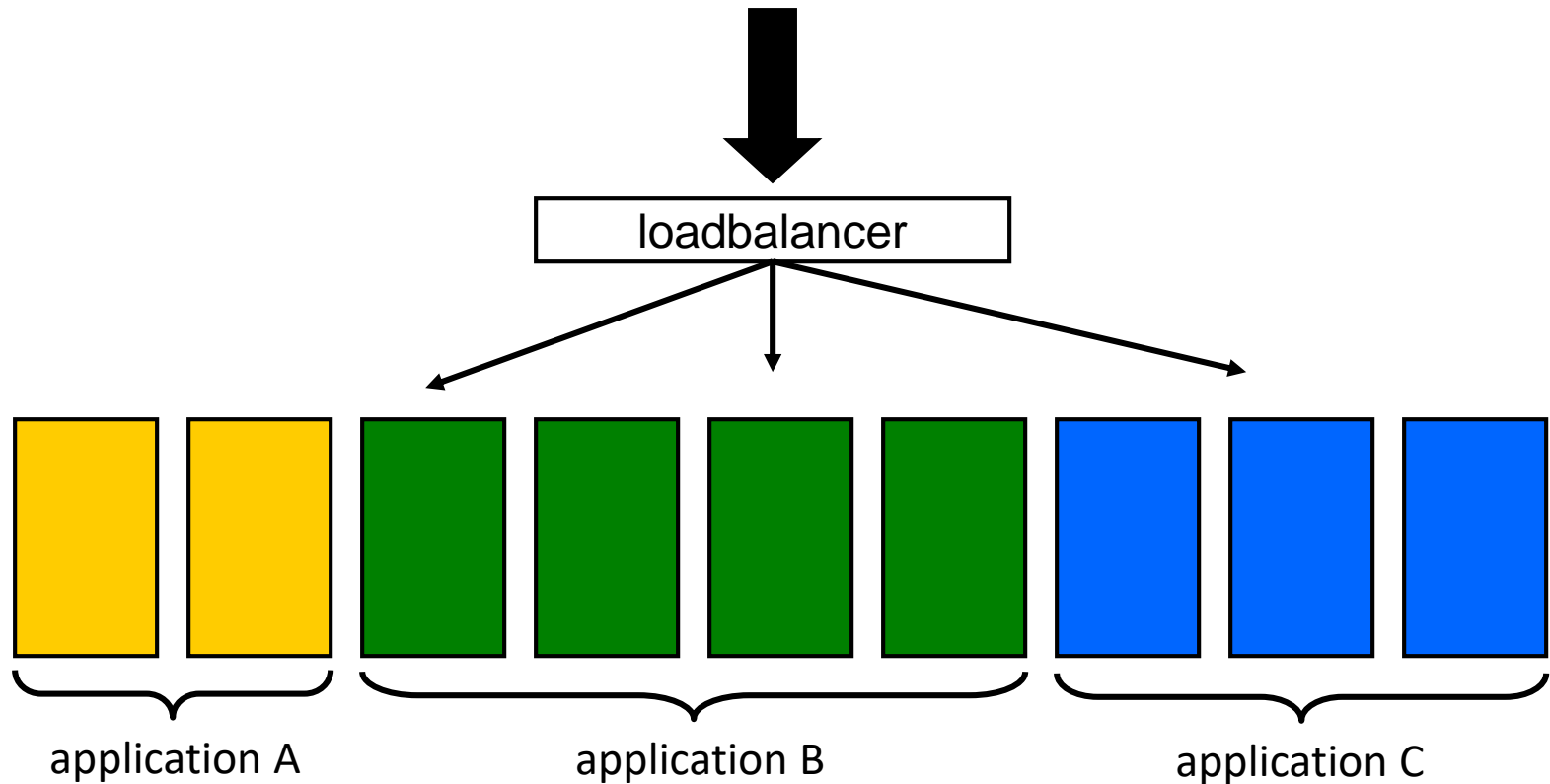
Drones



explore the environment and learn from experience

- ✓ not just blind search, try to be smart about it
- ✓ reward could be delayed

Reinforcement Learning Problem



Resource allocation among applications

Reinforcement Learning Framework



State



Action



Reward

$$x = \begin{bmatrix} A \\ J \end{bmatrix}$$

Policy

$$a = \pi(x)$$
$$a \sim \pi(\cdot | x)$$

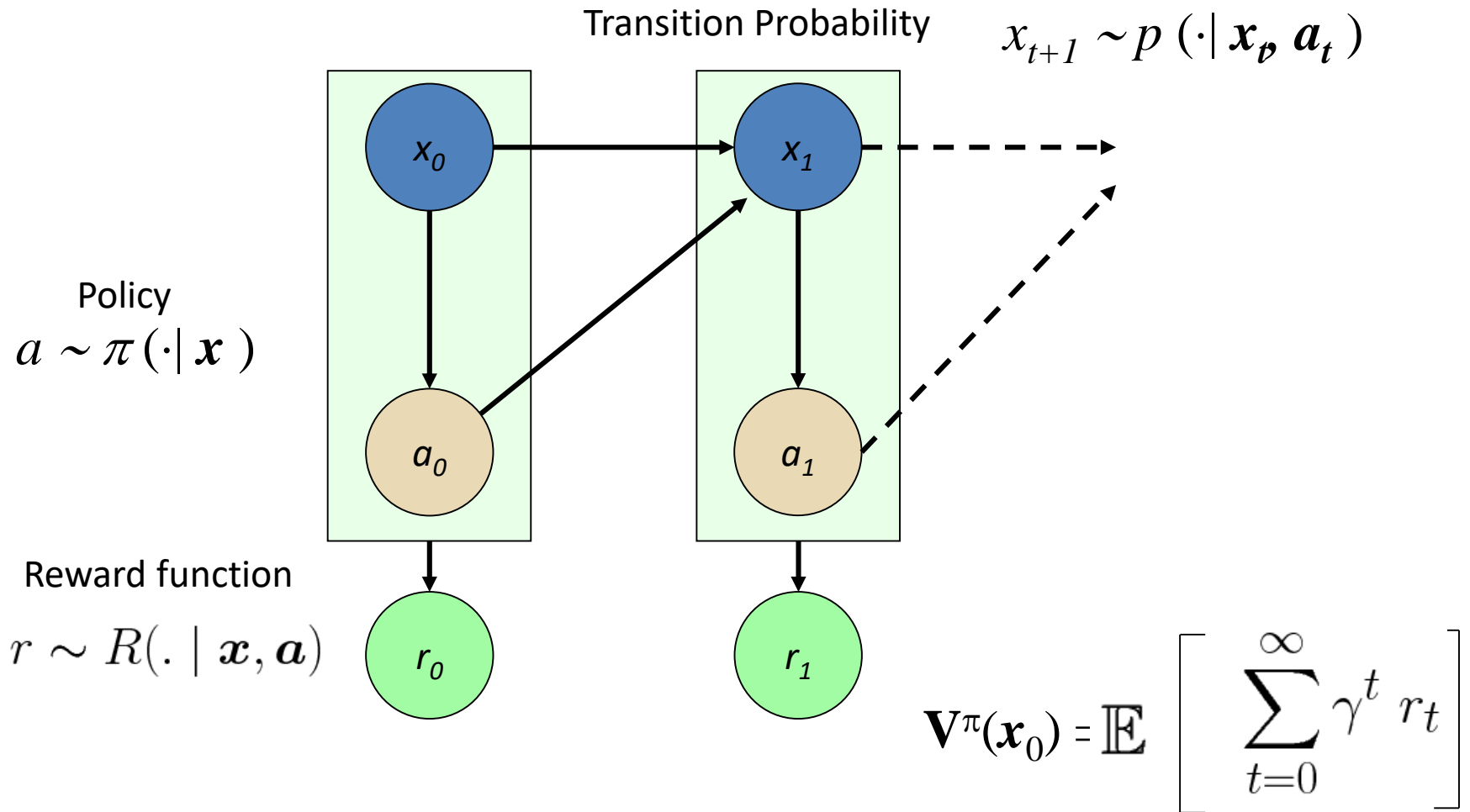
$$a = \text{stick}$$

Reward Function

$$r = R(x_t, a_t)$$
$$r \sim R(\cdot | x, a)$$

£££££

Markov Decision Process (MDP)



Value Function

$$\begin{aligned} V^\mu(\mathbf{x}_0) &= \mathbb{E}_{a_0, a_1, \dots; x_1, x_2, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{x}_t, \mu(\mathbf{x}_t)) \right] \\ &= \mathbb{E}_{a_0, a_1, \dots; x_1, x_2, \dots} \left[r(\mathbf{x}_0, \mu(\mathbf{x}_0)) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} r(\mathbf{x}_t, \mu(\mathbf{x}_t)) \right] \end{aligned}$$

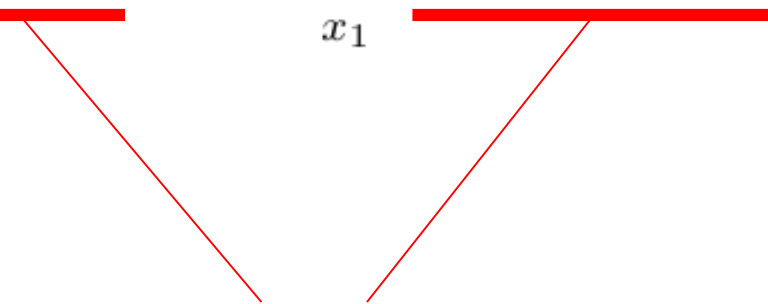
$$= \sum_{\mathbf{a}_0} \mu(\mathbf{a}_0 \mid \mathbf{x}_0) \left(r(\mathbf{x}_0, \mathbf{a}_0) + \sum_{x_1} p(x_1 \mid \mathbf{x}_0, \mathbf{a}_0) \gamma \mathbb{E}_{a_1, a_2, \dots; x_2, x_3, \dots} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(\mathbf{x}_t, \mu(\mathbf{x}_t)) \right] \right)$$

$$= \sum_{\mathbf{a}_0} \mu(\mathbf{a}_0 \mid \mathbf{x}_0) \left(r(\mathbf{x}_0, \mathbf{a}_0) + \gamma \sum_{x_1} p(x_1 \mid \mathbf{x}_0, \mathbf{a}_0) V^\mu(x_1) \right)$$

$$\mu = \pi$$

Optimal Policy

Assume one optimal action per state

$$V^*(\mathbf{x}_0) = \max_{\mathbf{a}_0} \left(\underbrace{r(\mathbf{x}_0, \mathbf{a}_0)} + \gamma \sum_{\mathbf{x}_1} \underbrace{p(\mathbf{x}_1 | \mathbf{x}_0, \mathbf{a}_0)} V^*(\mathbf{x}_1) \right)$$


Value Iteration

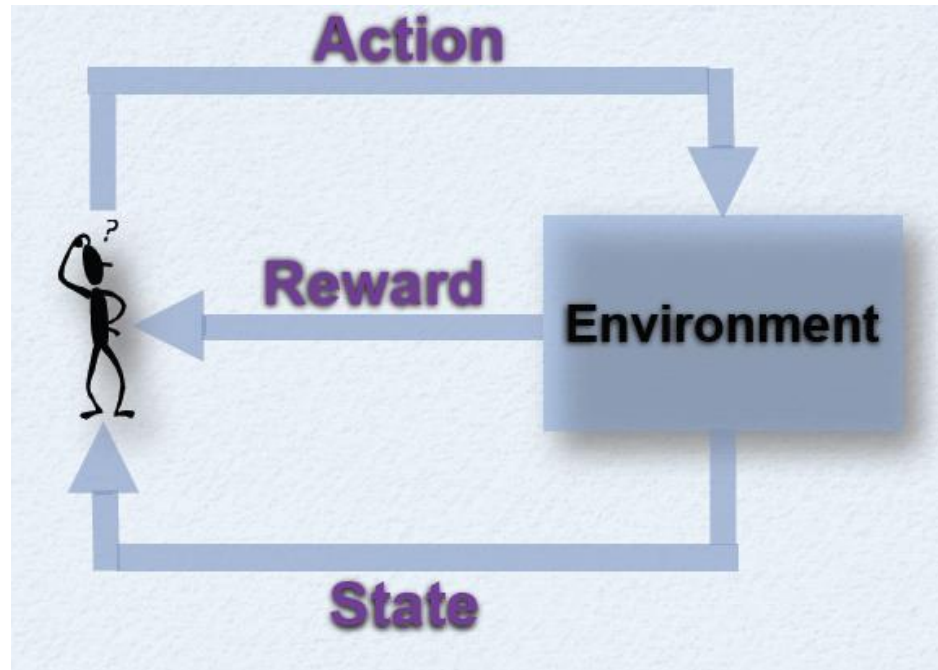
$$a_0^* \sim \pi^*(\cdot | \mathbf{x}_0)$$

$$a_1^* \sim \pi^*(\cdot | \mathbf{x}_1)$$

$$a_2^* \sim \pi^*(\cdot | \mathbf{x}_2)$$

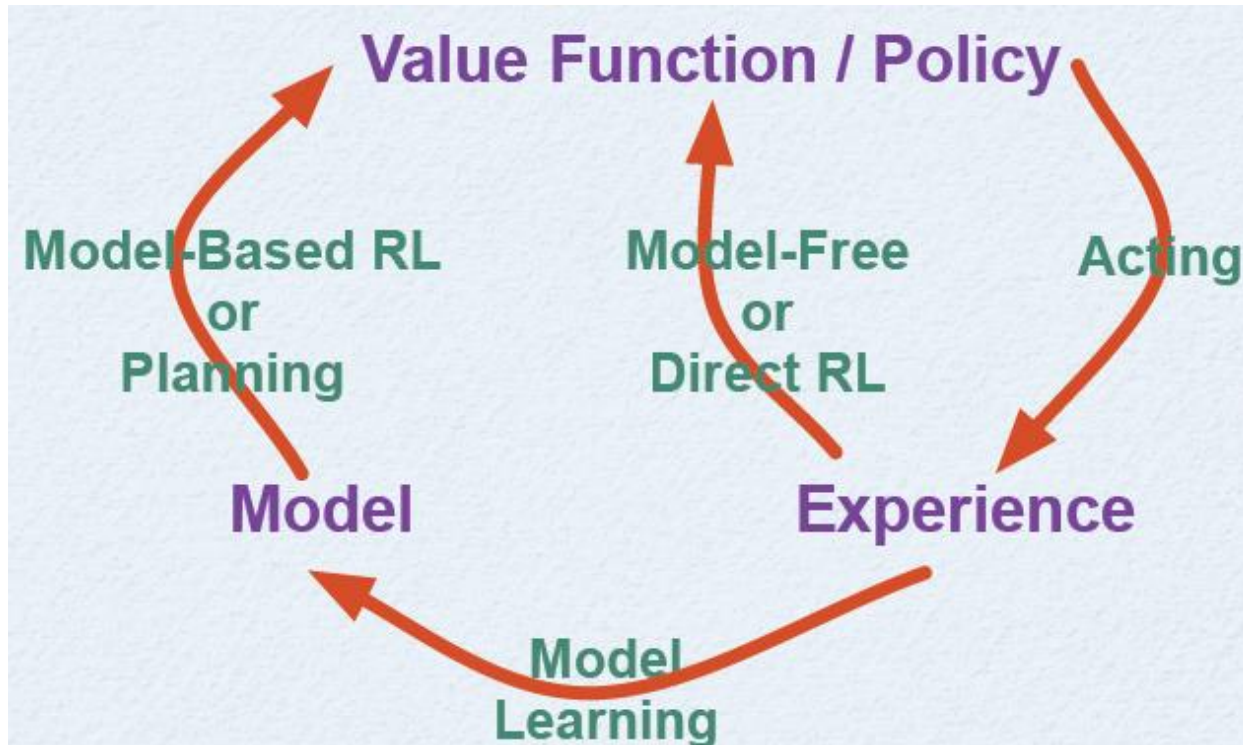
Unknown

Reinforcement Learning



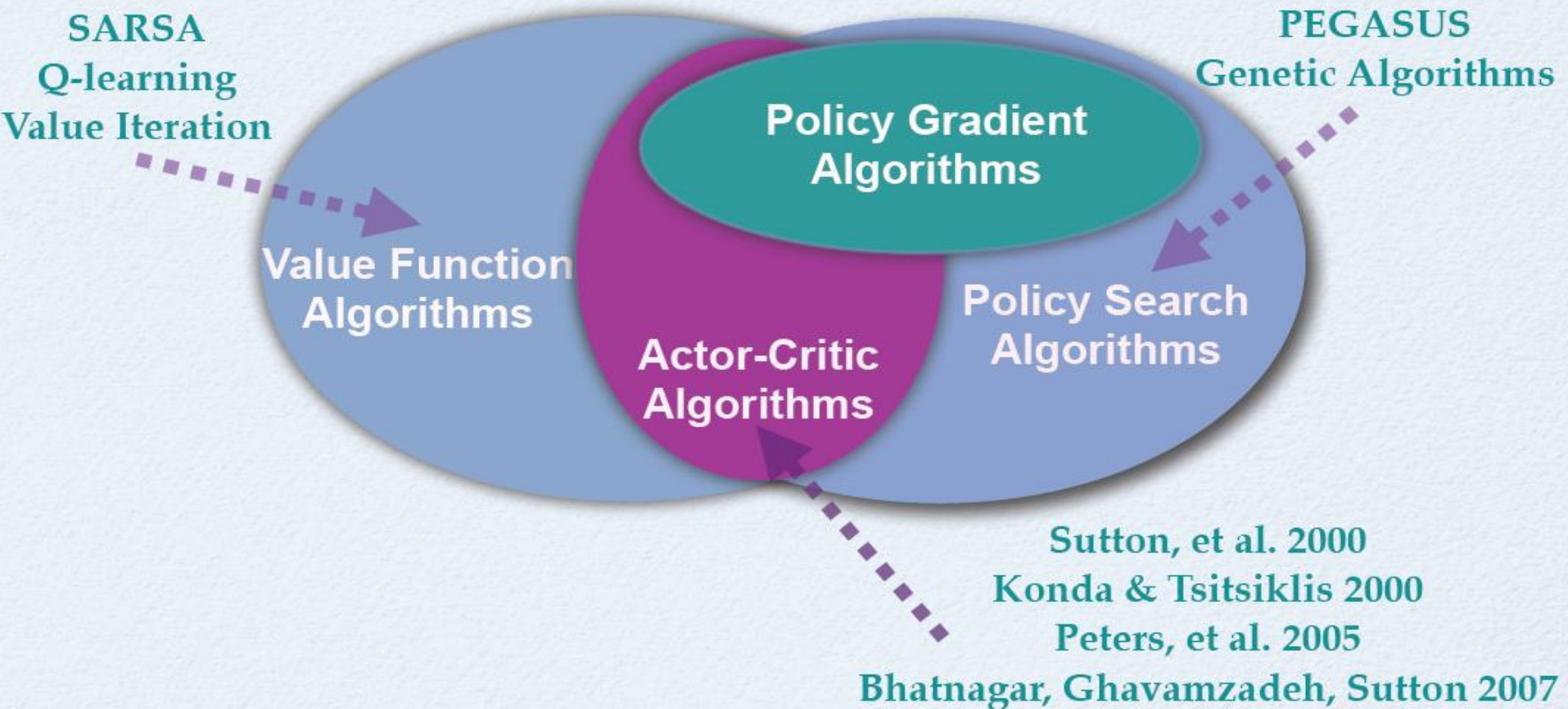
- ❑ **RL Problem:** Solve MDP when reward/transition models are unknown
 - ❑ **Basic Idea:** Use samples obtained from agent's interaction with environment
-

Reinforcement Learning



- **Model-Based:** Learn a model of the reward/transition dynamics and derive the value function/policy
 - **Model-Free:** Directly learn the value function/policy
-

Reinforcement Learning



Value-Based RL Solutions

□ Value Function Algorithms

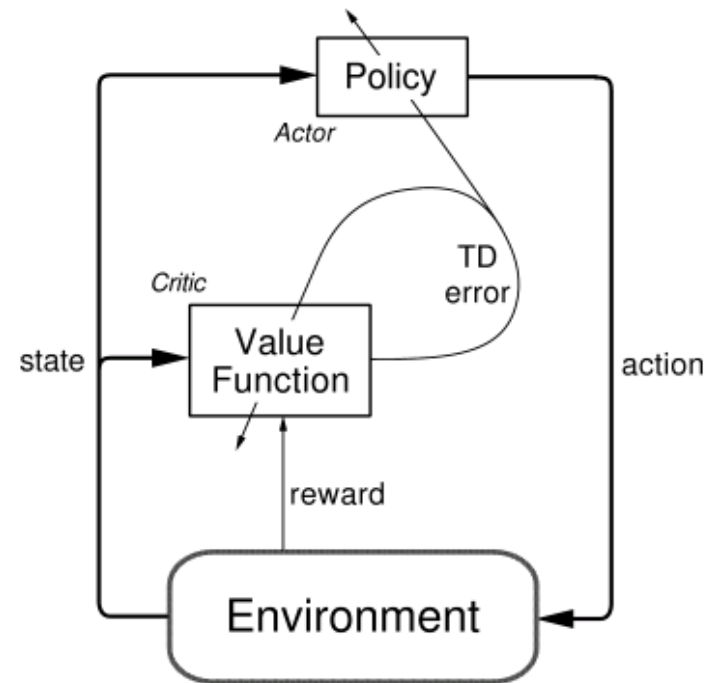
- ✓ Define a form for the value function
- ✓ Sample state-action-reward sequence
- ✓ Update value function
- ✓ Extract optimal policy

SARSA, Q-learning

Actor-Critic RL Solutions

□ Actor-Critic

- ✓ Define a policy structure (*actor*)
- ✓ Define a value function (*critic*)
- ✓ Sample state-action-reward
- ✓ Update both actor & critic



Policy-Based RL Solutions

□ Policy Search Algorithm

- ✓ Define a form for the policy
- ✓ Sample state-action-reward sequence
- ✓ Update policy

□ PEGASUS

(Policy Evaluation-of-Goodness
And Search Using Scenarios)



Online - Offline

□ Offline

- ✓ Use a simulator
- ✓ Policy fixed for each 'episode'
- ✓ Updates made at the end of episode

□ Online

- ✓ Directly interact with environment
 - ✓ Learning happens step-by-step
-

Model-Free Solutions

1. **Prediction:** Estimate $V(x)$ or $Q(x,a)$

2. **Control:** Extract policy

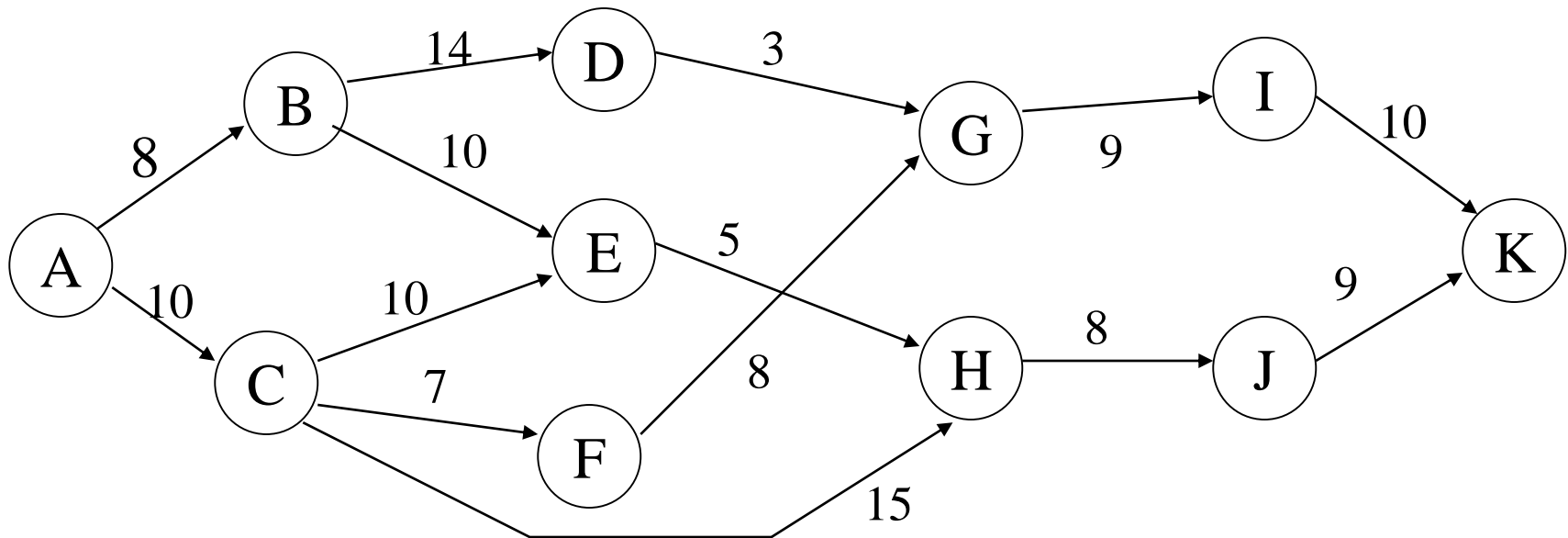
- On-Policy
- Off-Policy

Outlines

- Reinforcement Learning Introduction
 - Dynamic Programming Approaches
 - Monte Carlo Approaches
 - Temporal Difference Approaches
 - Bayesian Reinforcement Learning
 - Deep Reinforcement Learning
-

Dynamic Programming

Problem: How to determine the highway from A to K with the minimum total cost?



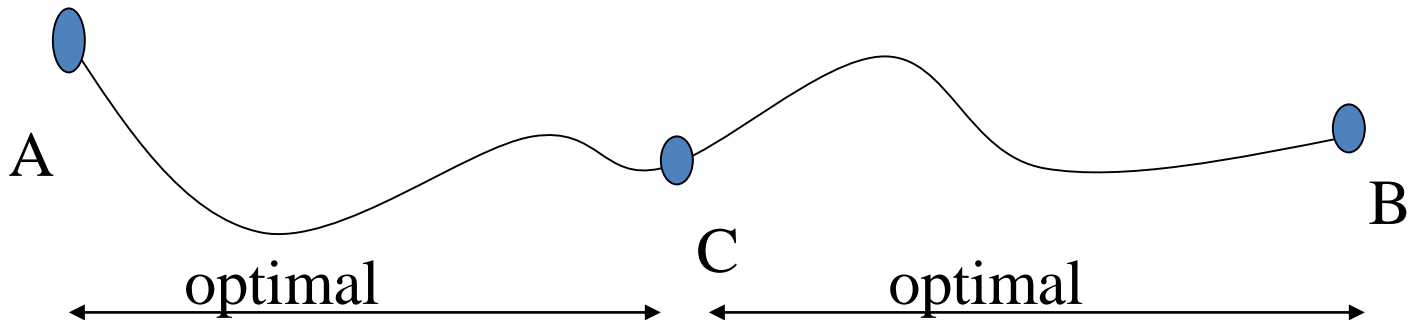
Dynamic Programming

General form:

if C belongs to an optimal path from A to B, then the sub-path A to C and C to B are also optimal

or

all sub-path of an optimal path is optimal



Dynamic Programming

□ main idea

- ✓ use value functions to search for good policies
- ✓ need a perfect model of the environment

□ two main components

policy evaluation: compute V^π from π



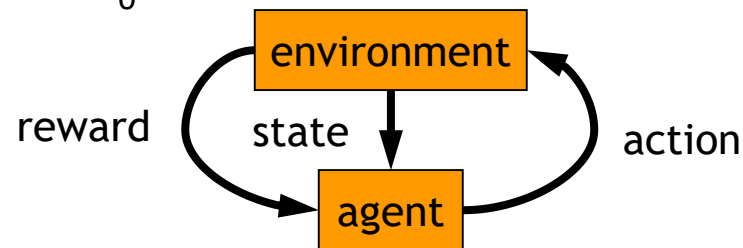
policy improvement: improve π based on V^π



- ✓ start with an arbitrary policy
 - ✓ repeat evaluation/improvement until convergence
-

Markov Decision Process

- ❑ **set of states:** S , **set of actions:** A , **initial state:** S_0
- ❑ **transition model:** $P(s,a,s')$
 $P([1,1], \text{up}, [1,2]) = 0.8$
- ❑ **reward function:** $r(s)$
 $r([4,3]) = +1$
- ❑ **goal:** maximize cumulative reward in the long run
- ❑ **policy:** mapping from S to A
 $\pi(s)$ or $\pi(s,a)$ (deterministic vs. stochastic)
- ❑ **reinforcement learning**
 - ✓ transitions and rewards usually not available
 - ✓ how to change the policy based on experience
 - ✓ how to explore the environment



Return from Rewards

- ❑ **episodic** (vs. continuing) tasks

“game over” after N steps

optimal policy depends on N; harder to analyze

- ❑ **additive rewards**

$$V(s_0, s_1, \dots) = r(s_0) + r(s_1) + r(s_2) + \dots$$

infinite value for continuing tasks

- ❑ **discounted rewards**

$$V(s_0, s_1, \dots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \dots$$

value bounded if rewards bounded

Value Functions

□ state value function: $V^\pi(s)$

expected return when starting in s and following π

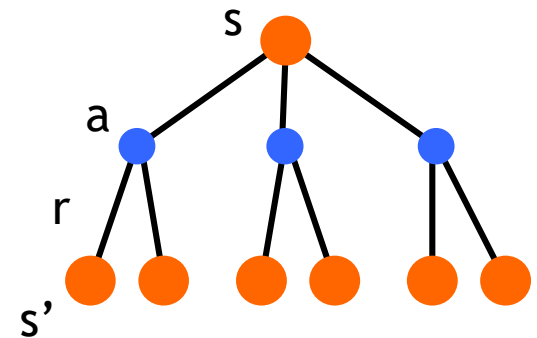
□ state-action value function: $Q^\pi(s,a)$

expected return when starting in s , performing a , and following π

□ useful for finding the optimal policy

can estimate from experience

pick the best action using $Q^\pi(s,a)$



□ Bellman equation

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [r_{ss'}^a + \gamma V^\pi(s')] = \sum_a \pi(s, a) Q^\pi(s, a)$$

Optimal Value Functions

□ there's a set of *optimal* policies

- ✓ V^π defines partial ordering on policies
- ✓ they share the same optimal value function $V^*(s) = \max_{\pi} V^\pi(s)$

□ Bellman optimality equation

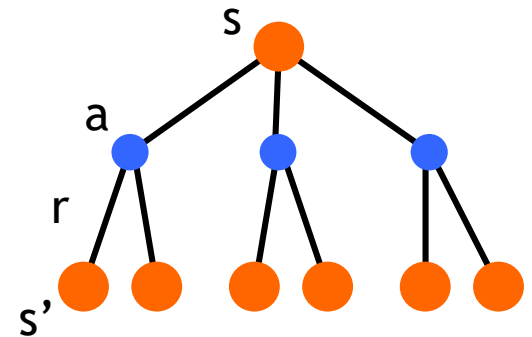
- ✓ system of n non-linear equations
- ✓ solve for $V^*(s)$
- ✓ easy to extract the optimal policy

$$V^*(s) = \max_a \sum_{s'} P_{ss'}^a \left[r_{ss'}^a + \gamma V^*(s') \right]$$

$Q(s, a)$

□ having $Q^*(s, a)$ makes it even simpler

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$



Policy Evaluation/Improvement

□ policy evaluation: $\pi \rightarrow V^\pi$

- ✓ Bellman equations define a system of n equations
- ✓ could solve, but will use iterative version

$$V_{k+1}(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [r_{ss'}^a + \gamma V_k(s')]$$

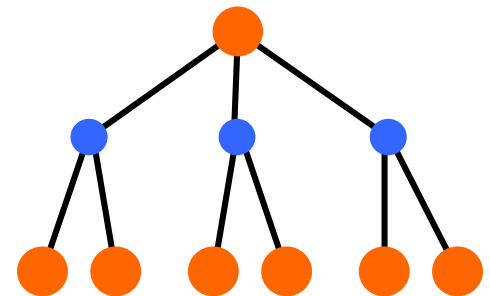
- ✓ start with an arbitrary value function V_0 , iterate until V_k converges

□ policy improvement: $V^\pi \rightarrow \pi'$

$$\pi'(s) = \arg \max_a Q^\pi(s, a)$$

$$= \arg \max_a \sum_{s'} P_{ss'}^a [r_{ss'}^a + \gamma V^\pi(s')]$$

π' either strictly better than π , or π' is optimal (if $\pi = \pi'$)

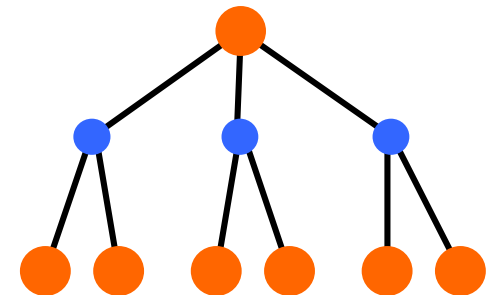


Policy/Value Iteration

□ Policy iteration

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

two nested iterations; too slow
don't need to converge to V^{π_k}
just move towards it



□ Value iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P_{ss'}^a [r_{ss'}^a + \gamma V_k(s')]$$

use Bellman optimality equation as an update
converges to V^*

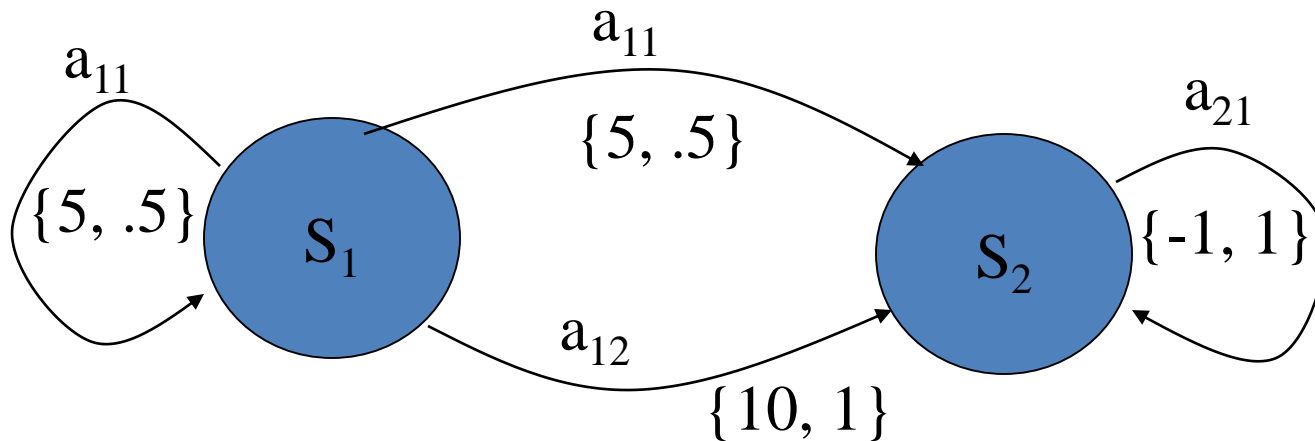
Example

Decision epochs: $T = \{1, 2, \dots, N\}$

State : $S = \{s_1, s_2\}$ Actions: $A_{s_1} = \{a_{11}, a_{12}\}, A_{s_2} = \{a_{21}\}$

Costs: $C_t(s_1, a_{11}) = 5, C_t(s_1, a_{12}) = 10, C_t(s_2, a_{21}) = -1$

Transition probabilities: $p_t(s_1 / s_1, a_{11}) = 0.5, p_t(s_2 / s_1, a_{11}) = 0.5, p_t(s_1 / s_1, a_{12}) = 0, p_t(s_2 / s_1, a_{12}) = 1, p_t(s_1 / s_2, a_{21}) = 0, p_t(s_2 / s_2, a_{21}) = 1$



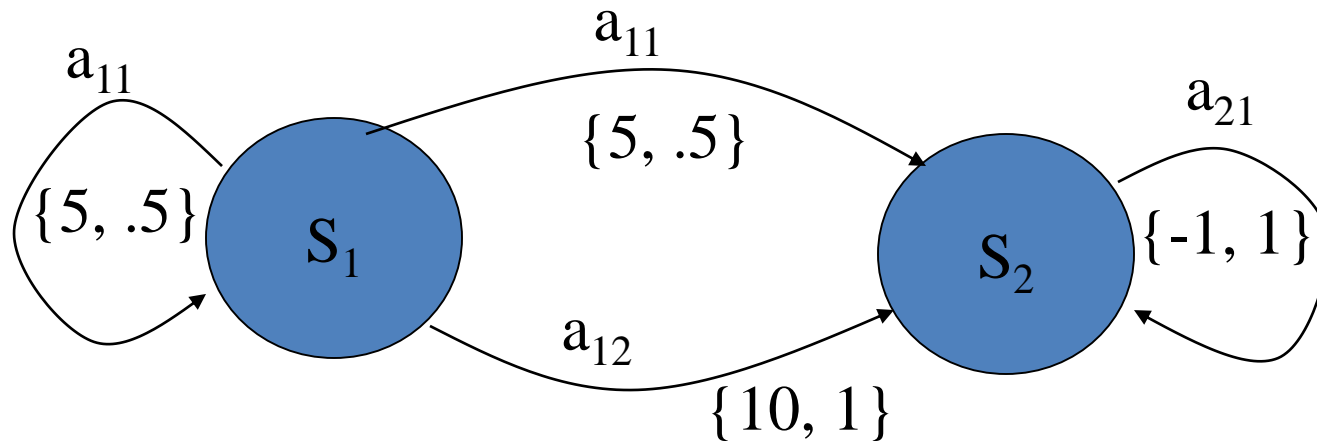
Example: Deterministic Markov Policy

Decision epoch 1:

$$d_1(s_1) = a_{11}, d_1(s_2) = a_{21}$$

Decision epoch 2:

$$d_2(s_1) = a_{12}, d_2(s_2) = a_{21}$$



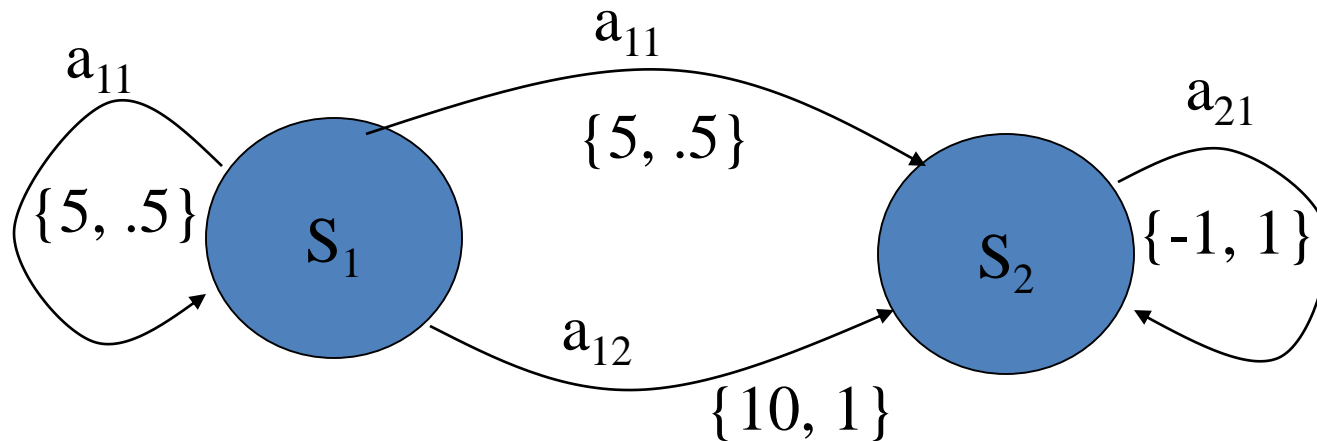
Example: Randomized Markov Policy

Decision epoch 1:

$$P_{1,s_1}(a_{11}) = 0.7, P_{1,s_1}(a_{12}) = 0.3; P_{1,s_2}(a_{21}) = 1$$

Decision epoch 2:

$$P_{2,s_1}(a_{11}) = 0.4, P_{2,s_1}(a_{12}) = 0.6; P_{2,s_2}(a_{21}) = 1$$



Example: Deterministic History-Dependent

Decision epoch 1:

$$d_1(s_1) = a_{11}$$

$$d_1(s_2) = a_{21}$$

Decision epoch 2:

history h

$d_2(h, s_1)$

$d_2(h, s_2)$

(s_1, a_{11})

a_{13}

a_{21}

(s_1, a_{12})

infeasible

a_{21}

(s_1, a_{13})

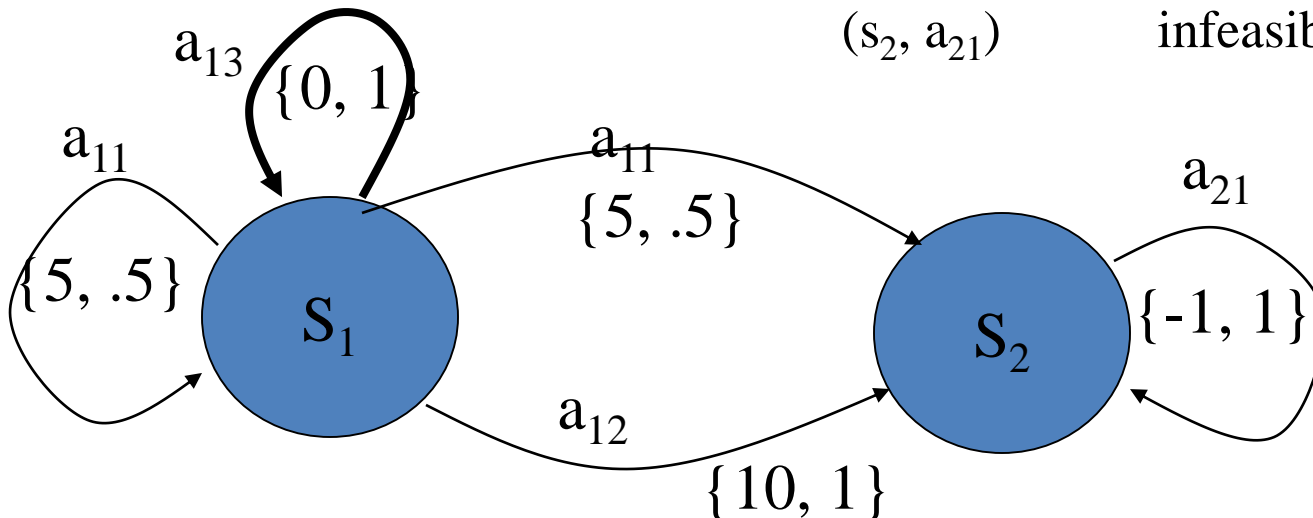
a_{11}

infeasible

(s_2, a_{21})

infeasible

a_{21}



Example: Randomized History-Dependent

Decision epoch 1:

$$P_{1,s_1}(a_{11}) = 0.6$$

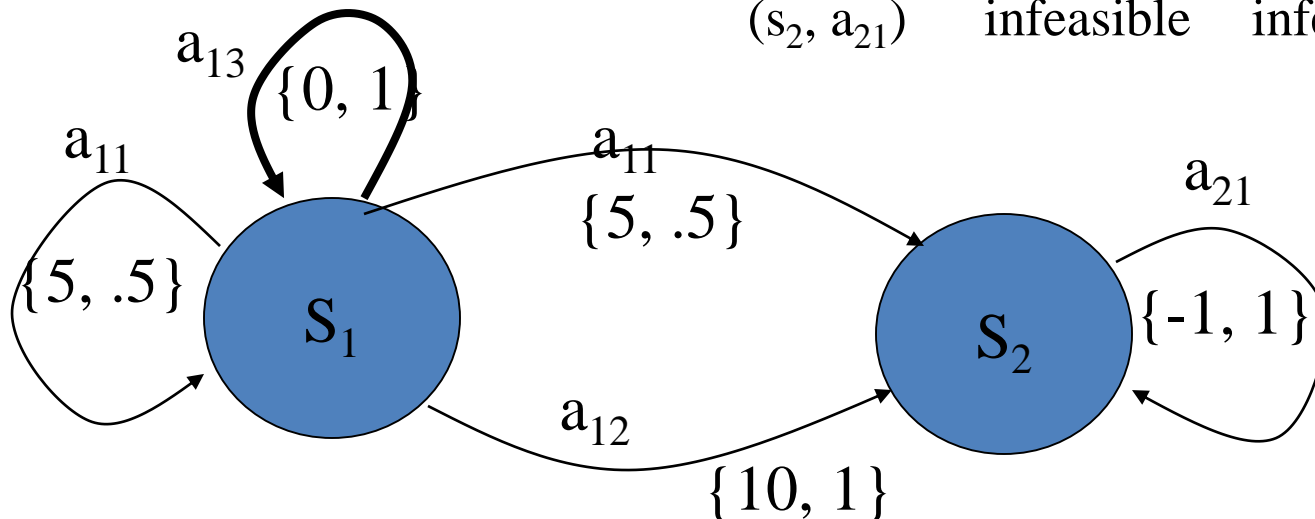
$$P_{1,s_1}(a_{12}) = 0.3$$

$$P_{1,s_1}(a_{13}) = 0.1$$

$$P_{1,s_2}(a_{21}) = 1$$

Decision epoch 2: at $s = s_1$

| history h | $P(a = a_{11})$ | $P(a = a_{12})$ | $P(a = a_{13})$ |
|-----------------|-----------------|-----------------|-----------------|
| (s_1, a_{11}) | 0.4 | 0.3 | 0.3 |
| (s_1, a_{12}) | infeasible | infeasible | infeasible |
| (s_1, a_{13}) | 0.8 | 0.1 | 0.1 |
| (s_2, a_{21}) | infeasible | infeasible | infeasible |



**at $s = s_2$,
select a_{21}**

Outlines

- Reinforcement Learning Introduction
 - Dynamic Programming Approaches
 - Monte Carlo Approaches
 - Temporal Difference Approaches
 - Bayesian Reinforcement Learning
 - Deep Reinforcement Learning
-

Monte Carlo Methods

- ❑ don't need full knowledge of environment
 - ✓ just experience, or
 - ✓ simulated experience

 - ❑ but similar to DP
 - ✓ policy evaluation, policy improvement

 - ❑ averaging sample returns
 - ✓ defined only for episodic tasks
-

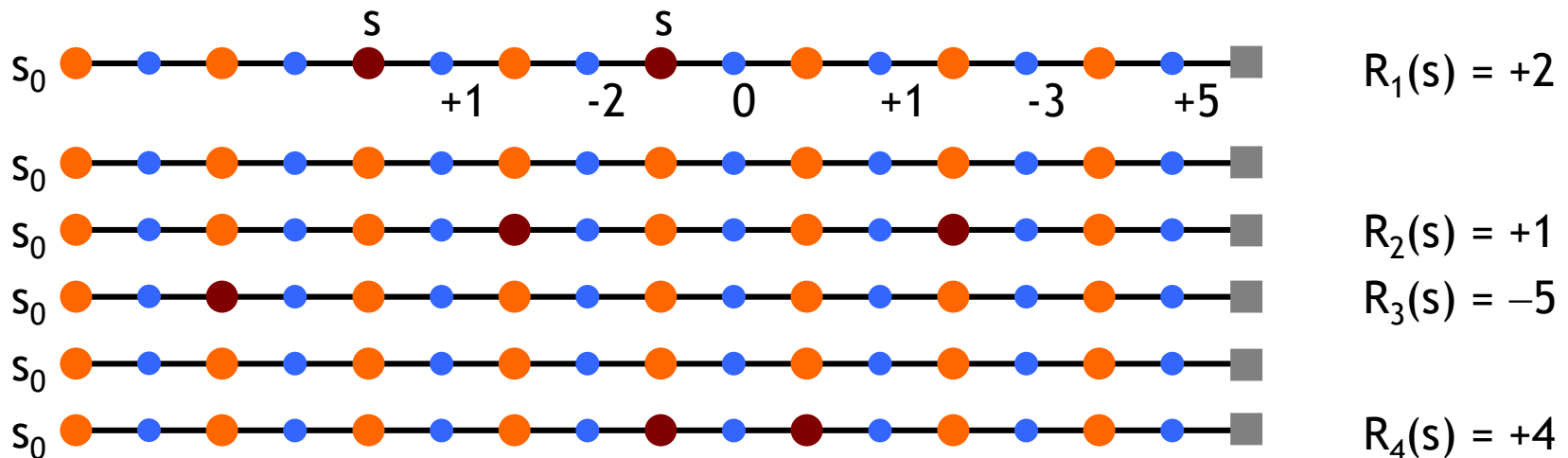
Monte Carlo Policy Evaluation

□ want to estimate $V^\pi(s)$

= expected return starting from s and following π
estimate as average of observed returns in state s

□ first-visit MC

average returns following the first visit to state s

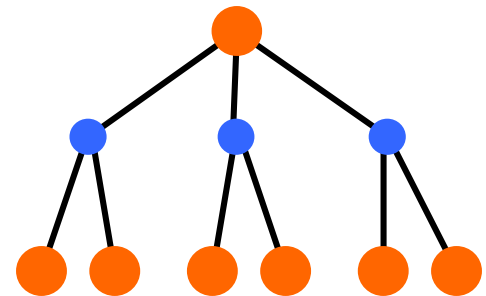


$$V^\pi(s) \approx (2 + 1 - 5 + 4) / 4 = 0.5$$

Monte Carlo Control

- V^π not enough for policy improvement
need exact model of environment

- estimate $Q^\pi(s,a)$
 $\pi'(s) = \arg \max_a Q^\pi(s, a)$



- MC control

$$u^{\pi_0} \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} Q^*$$

- non-stationary environment





- a problem $V(s) \leftarrow V(s) + \alpha [R - V(s)] = (1 - \alpha)V(s) + \alpha R$
greedy policy won't explore all actions

Maintaining Exploration

- ❑ deterministic/greedy policy won't explore all actions
don't know anything about the environment at the beginning
need to try all actions to find the optimal one
 - ❑ maintain exploration
use *soft* policies instead: $\pi(s,a) > 0$ (for all s,a)
 - ❑ ϵ -greedy policy
with probability $1-\epsilon$ perform the optimal/greedy action
with probability ϵ perform a random action

will keep exploring the environment
slowly move it towards greedy policy: $\epsilon \rightarrow 0$
-

Monte Carlo Predictions

| | Leave car park | Get out of city | Motorway | Enter Cambridge |
|--|---|---|--|---|
| State |  |  |  |  |
| Value | -90 | -83 | -55 | -11 |
| Reward | -13 | -15 | -61 | -11 |
| Updated | -100 | -87 | -72 | -11 |
| $V(\mathbf{x}_t) \leftarrow V(\mathbf{x}_t) + \alpha(R_t - V(\mathbf{x}_t))$ | | | | |

Simulated Experience

□ 5-card draw poker

s_0 : A♣, A♦, 6♠, A♥, 2♠

a_0 : discard 6♠, 2♠

s_1 : A♣, A♦, A♥, A♠, 9♠ + dealer takes 4 cards

return: +1 (probably)

□ DP

list all states, actions, compute $P(s,a,s')$

$$P([A♣, A♦, 6♠, A♥, 2♠], [6♠, 2♠], [A♠, 9♠, 4]) = 0.00192$$

□ MC

all you need are sample episodes

let MC play against a random policy, or itself, or another algorithm

Summary of Monte Carlo

- ❑ don't need a model of environment
 - ✓ averaging of sample returns
 - ✓ only for episodic tasks
 - ❑ learn from sample episodes or simulated experience
 - ❑ can concentrate on “important” states
 - don't need a full sweep
 - ❑ need to maintain exploration
 - use soft policies
-

Outlines

- Reinforcement Learning Introduction
 - Dynamic Programming Approaches
 - Monte Carlo Approaches
 - Temporal Difference Approaches
 - Bayesian Reinforcement Learning
 - Deep Reinforcement Learning
-

Temporal Difference Learning

- combines ideas from MC and DP
 - like MC: learn directly from experience (don't need a model)
 - like DP: learn from values of successors
 - works for continuous tasks, usually faster than MC

- constant-alpha MC:
 - have to wait until the end of episode to update

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$







target

- simplest TD
 - update after every step, based on the successor

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



Temporal Difference Predictions

| | Leave car park | Get out of city | Motorway | Enter Cambridge |
|---------|---|---|--|---|
| State |  |  |  |  |
| Value | -90 | -83 | -55 | -11 |
| Reward | -13 | -15 | -61 | -11 |
| Updated | -96 | -70 | -72 | -11 |

$$V(\mathbf{x}_t) \leftarrow V(\mathbf{x}_t) + \alpha(r_t + \gamma V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t))$$

Advantages of TD

- ❑ Don't need a model of reward/transitions
 - ❑ Online, fully incremental
 - ❑ Proved to converge given conditions on step-size
 - ❑ “Usually” faster than MC methods
-

MC vs. TD

- observed the following 8 episodes:

A – 0, B – 0

B – 1

B – 1

B – 1

B – 1

B – 1

B – 1

B – 0

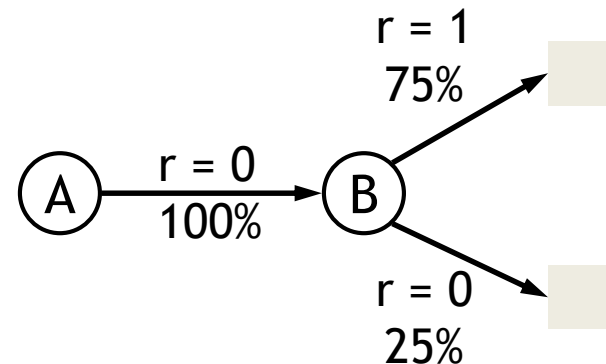
- MC and TD agree on $V(B) = 3/4$

- MC: $V(A) = 0$

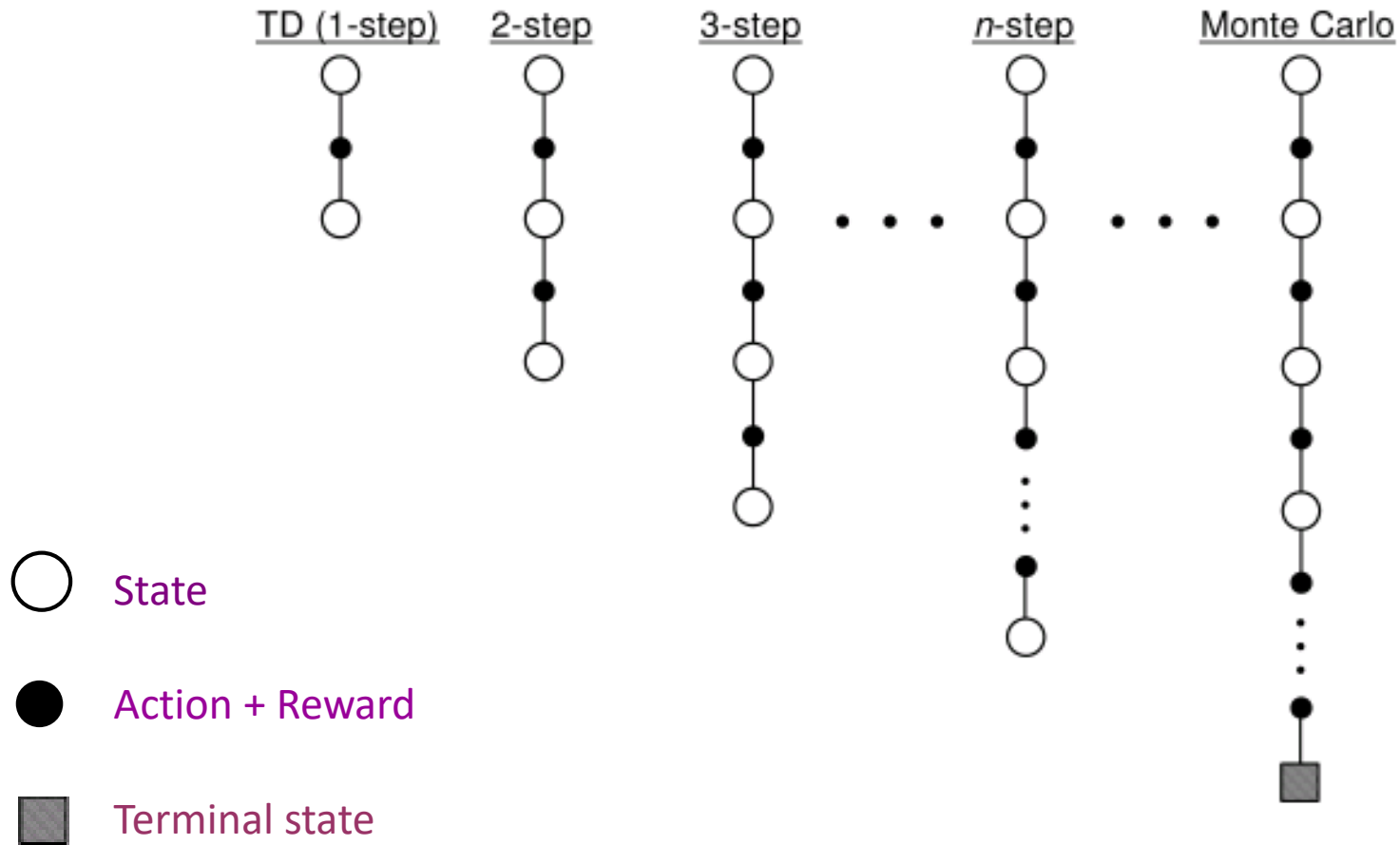
converges to values that minimize the error on training data

- TD: $V(A) = 3/4$

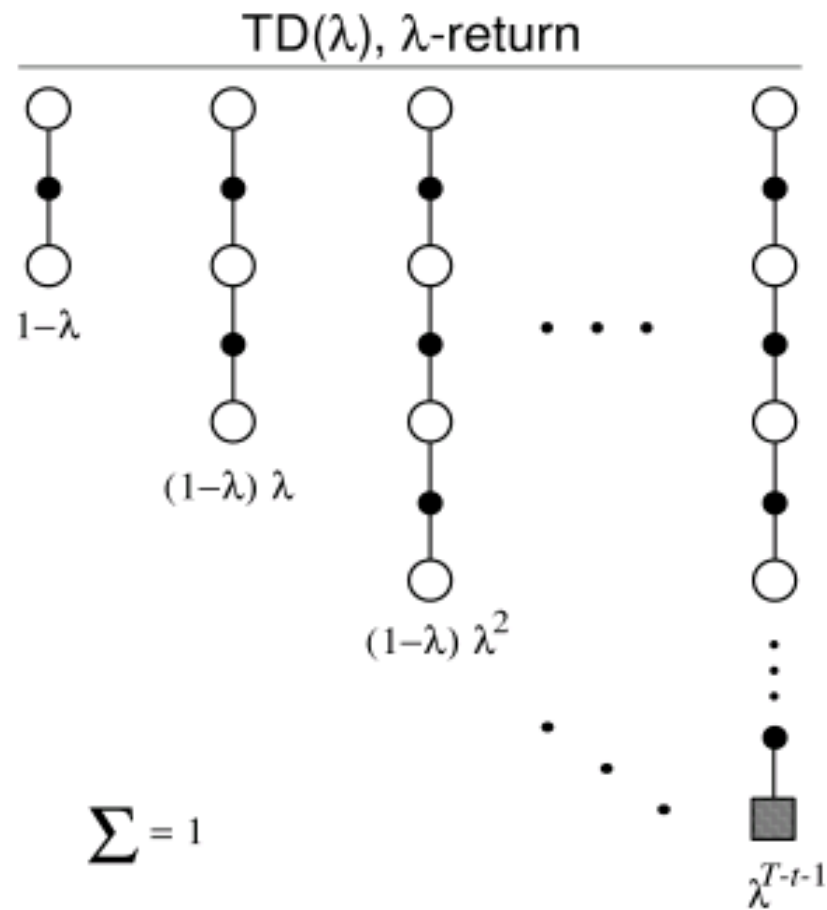
converges to ML estimate
of the Markov process



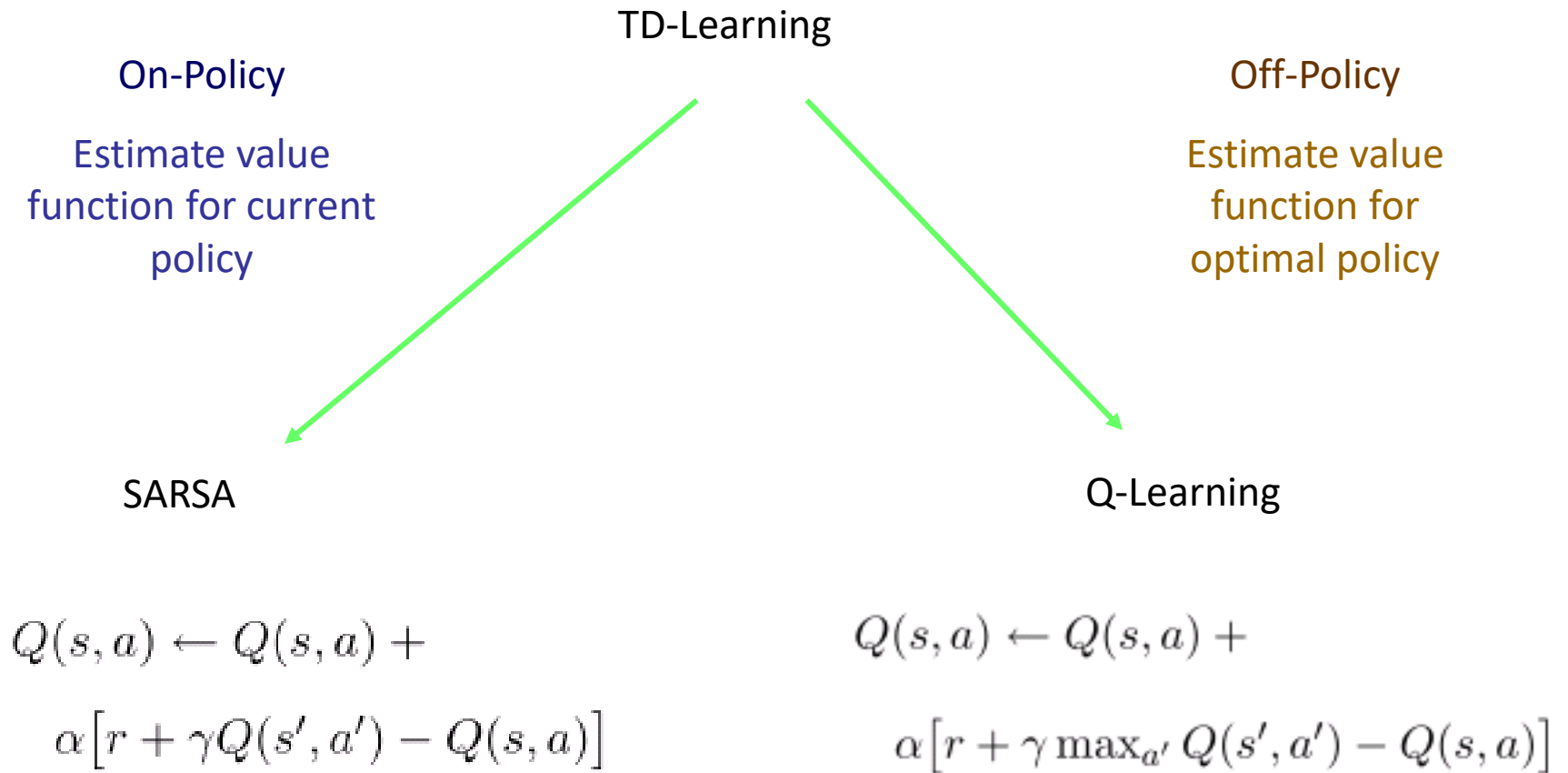
From TD to TD(λ)



From TD to TD(λ)

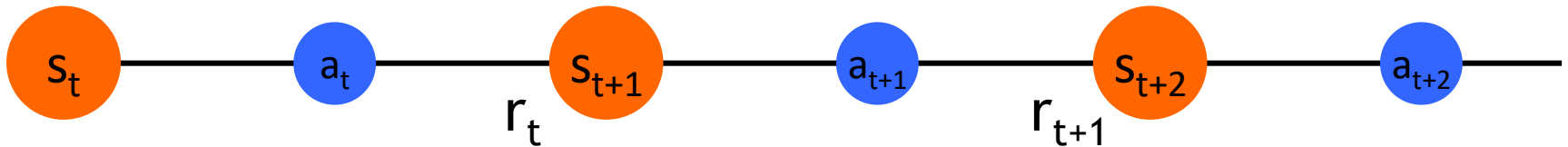


SARSA & Q-learning



SARSA

□ again, need $Q(s,a)$, not just $V(s)$



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

□ control

start with a random policy

update Q and π after each step

again, need ϵ -soft policies

Q-Learning

- before: on-policy algorithms
start with a random policy, iteratively improve
converge to optimal

- **Q-learning**: off-policy
use any policy to estimate Q

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

Q directly approximates Q^* (Bellman optimality eqn)
independent of the policy being followed
only requirement: keep updating each (s,a) pair

- **SARSA**

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

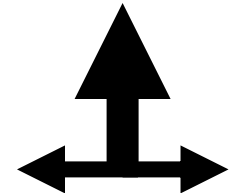
Robot in A Room

| | | | |
|-------|--|--|----|
| | | | +1 |
| | | | -1 |
| START | | | |

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

what is the solution?

Is This A Solution?

| | | | |
|---|---|---|----|
| → | → | → | +1 |
| ↑ | | | -1 |
| ↑ | | | |

- ❑ only if actions deterministic
not in this case (actions are stochastic)
 - ❑ solution/policy
mapping from each state to an action
-

Optimal Policy

| | | | |
|---|---|---|----|
| → | → | → | +1 |
| ↑ | | ↑ | -1 |
| ↑ | ← | ← | ← |

Reward for Each Step: -2

| | | | |
|---|---|---|----|
| → | → | → | +1 |
| ↑ | | → | -1 |
| → | → | → | ↑ |

Reward for each step: -0.1

| | | | |
|---|---|---|----|
| → | → | → | +1 |
| ↑ | | ↑ | -1 |
| ↑ | → | ↑ | ← |

Reward for each step: -0.04

| | | | |
|---|---|---|----|
| → | → | → | +1 |
| ↑ | | ↑ | -1 |
| ↑ | ← | ← | ← |

Reward for each step: -0.01

| | | | |
|---|---|---|----|
| → | → | → | +1 |
| ↑ | | ← | -1 |
| ↑ | ← | ← | ↓ |

Reward for each step: +0.01

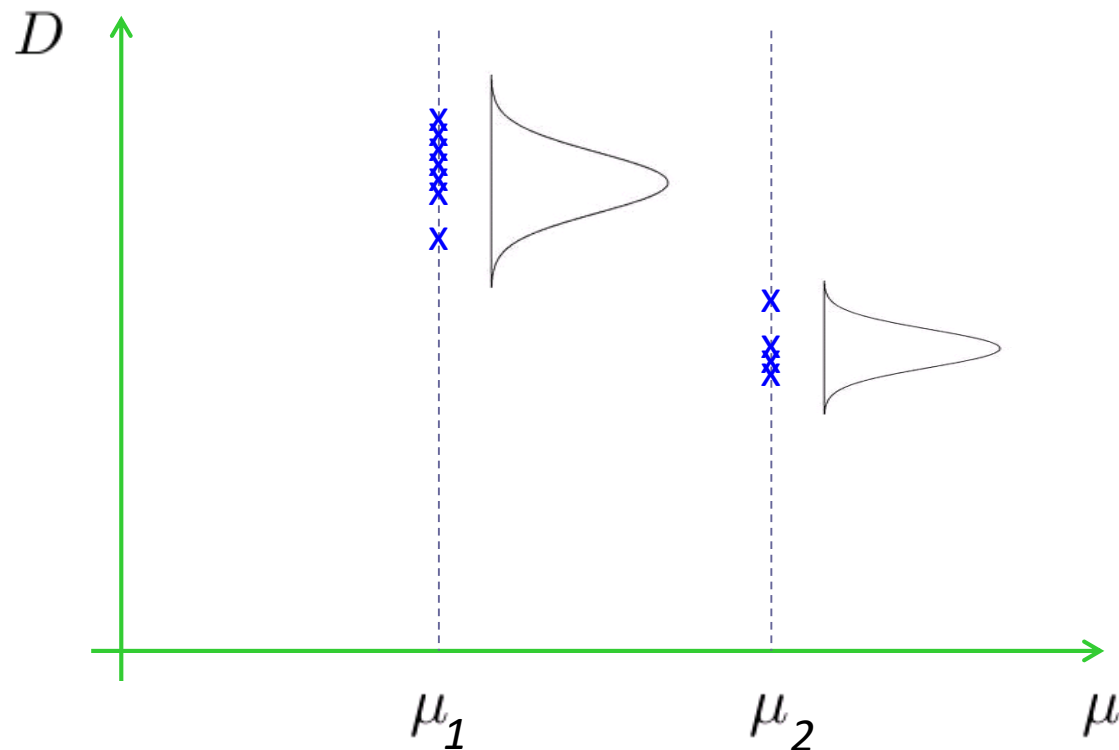
| | | | |
|---|---|---|----|
| ↓ | ← | ← | +1 |
| ↓ | | ← | -1 |
| ← | ← | ← | ↓ |

Outlines

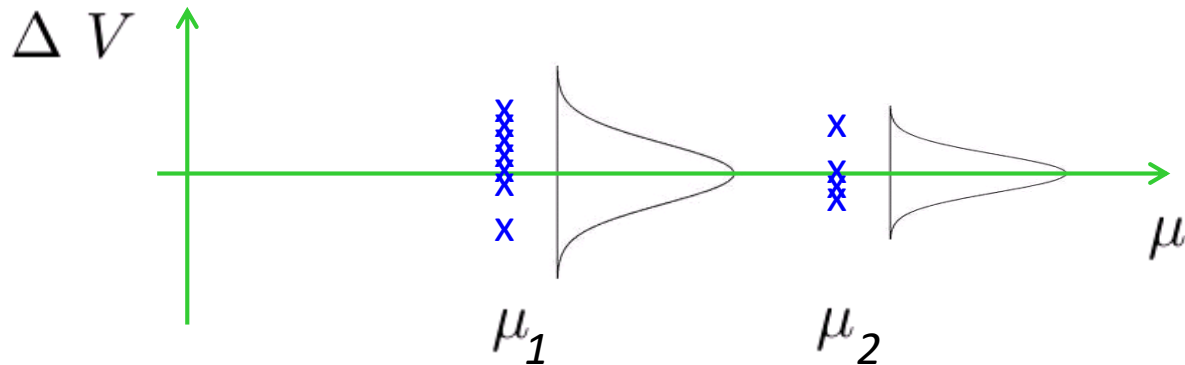
- Reinforcement Learning Introduction
 - Dynamic Programming Approaches
 - Monte Carlo Approaches
 - Temporal Difference Approaches
 - Bayesian Reinforcement Learning
 - Deep Reinforcement Learning
-

Gaussian Process Temporal Difference

$$D^\mu(\mathbf{x}_0) = \sum_{t=0}^{\infty} \gamma^t r(\mathbf{x}_t, \mu(\mathbf{x}_t)) \quad V^\mu(\mathbf{x}_0) = \mathbb{E}_\mu[D^\mu(\mathbf{x}_0)]$$



Gaussian Process Temporal Difference



$$\Delta V^\mu(\mathbf{x}_0) = D^\mu(\mathbf{x}_0) - V^\mu(\mathbf{x}_0)$$

$$\Delta V \sim \mathcal{N}(0, \sigma^2)$$

Gaussian Process Temporal Difference

$$D^\mu(\mathbf{x}_0) = r_0 + \gamma D^\mu(\mathbf{x}_1)$$

$$\begin{aligned} D^\mu(\mathbf{x}_0) &= V^\mu(\mathbf{x}_0) + D^\mu(\mathbf{x}_0) - V^\mu(\mathbf{x}_0) \\ &= V^\mu(\mathbf{x}_0) + \Delta V^\mu(\mathbf{x}_0) \end{aligned}$$

$$\begin{aligned} R(\mathbf{x}_i) &= V(\mathbf{x}_i) - \gamma V(\mathbf{x}_{i+1}) + N(\mathbf{x}_i, \mathbf{x}_{i+1}) \\ N(\mathbf{x}_i, \mathbf{x}_{i+1}) &\stackrel{\text{def}}{=} \Delta V(\mathbf{x}_i) - \gamma \Delta V(\mathbf{x}_{i+1}) \end{aligned}$$

Gaussian Process Temporal Difference

$$\begin{aligned} R(\mathbf{x}_i) &= V(\mathbf{x}_i) - \gamma V(\mathbf{x}_{i+1}) + N(\mathbf{x}_i, \mathbf{x}_{i+1}) \\ N(\mathbf{x}_i, \mathbf{x}_{i+1}) &\stackrel{\text{def}}{=} \Delta V(\mathbf{x}_i) - \gamma \Delta V(\mathbf{x}_{i+1}) \end{aligned}$$

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix}$$

$$R_t = \mathbf{H}_{t+1} V_{t+1} + N_t, \text{ with } N_t \sim \mathcal{N}\{0, \sigma^2 \mathbf{H}_{t+1} \mathbf{H}_{t+1}^\top\}$$

Gaussian Process Temporal Difference

General noise covariance:

$$\text{Cov}[N_t] = \Sigma_t$$

Joint distribution:

$$\begin{bmatrix} R_{t-1} \\ V(x) \end{bmatrix} \sim \mathcal{N} \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \Sigma_t & \mathbf{H}_t \mathbf{k}_t(x) \\ \mathbf{k}_t(x)^\top \mathbf{H}_t^\top & k(x, x) \end{bmatrix} \right\}$$

Condition on R_{t-1} :

$$\mathbb{E}[V(x) | R_{t-1} = \mathbf{r}_{t-1}] = \mathbf{k}_t(x)^\top \alpha_t$$

$$\text{Cov}[V(x), V(x') | R_{t-1} = \mathbf{r}_{t-1}] = k(x, x') - \mathbf{k}_t(x)^\top \mathbf{C}_t \mathbf{k}_t(x')$$

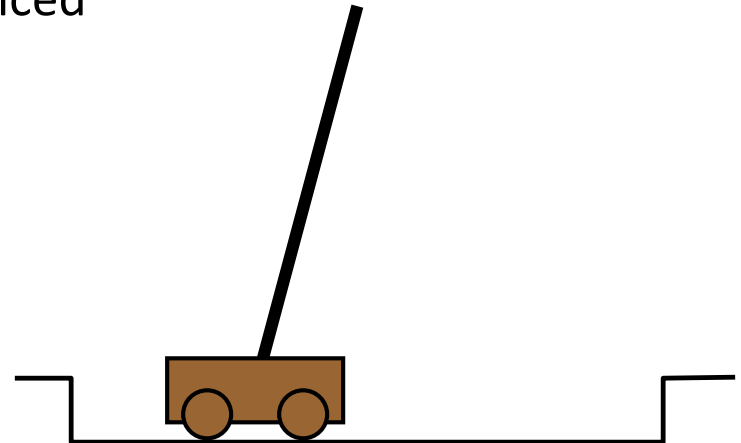
$$\alpha_t = \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \Sigma_t \right)^{-1} \mathbf{r}_{t-1}, \quad \mathbf{C}_t = \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \Sigma_t \right)^{-1} \mathbf{H}_t$$

Outlines

- Reinforcement Learning Introduction
 - Dynamic Programming Approaches
 - Monte Carlo Approaches
 - Temporal Difference Approaches
 - Bayesian Reinforcement Learning
 - Deep Reinforcement Learning
-

State Representation

- ❑ pole-balancing
move car left/right to keep the pole balanced
- ❑ state representation
position and velocity of car
angle and angular velocity of pole
- ❑ what about *Markov property*?
would need more info
noise in sensors, temperature, bending of pole
- ❑ solution
coarse discretization of state variables
left, center, right
totally non-Markov, but still works



Function Approximation

- represent V_t as a parameterized regression function

- ✓ linear regression, decision tree, neural net, ...

- ✓ linear regression:

$$V_t(s) = \vec{\theta}_t^T \vec{\phi}_s = \sum_{i=1}^n \theta_t(i) \phi_s(i)$$

- update parameters instead of entries in a table

- ✓ better generalization

fewer parameters and updates affect “similar” states as well

- TD update

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

$$V(s_t) \mapsto r_{t+1} + \gamma V(s_{t+1})$$

$f(x)$

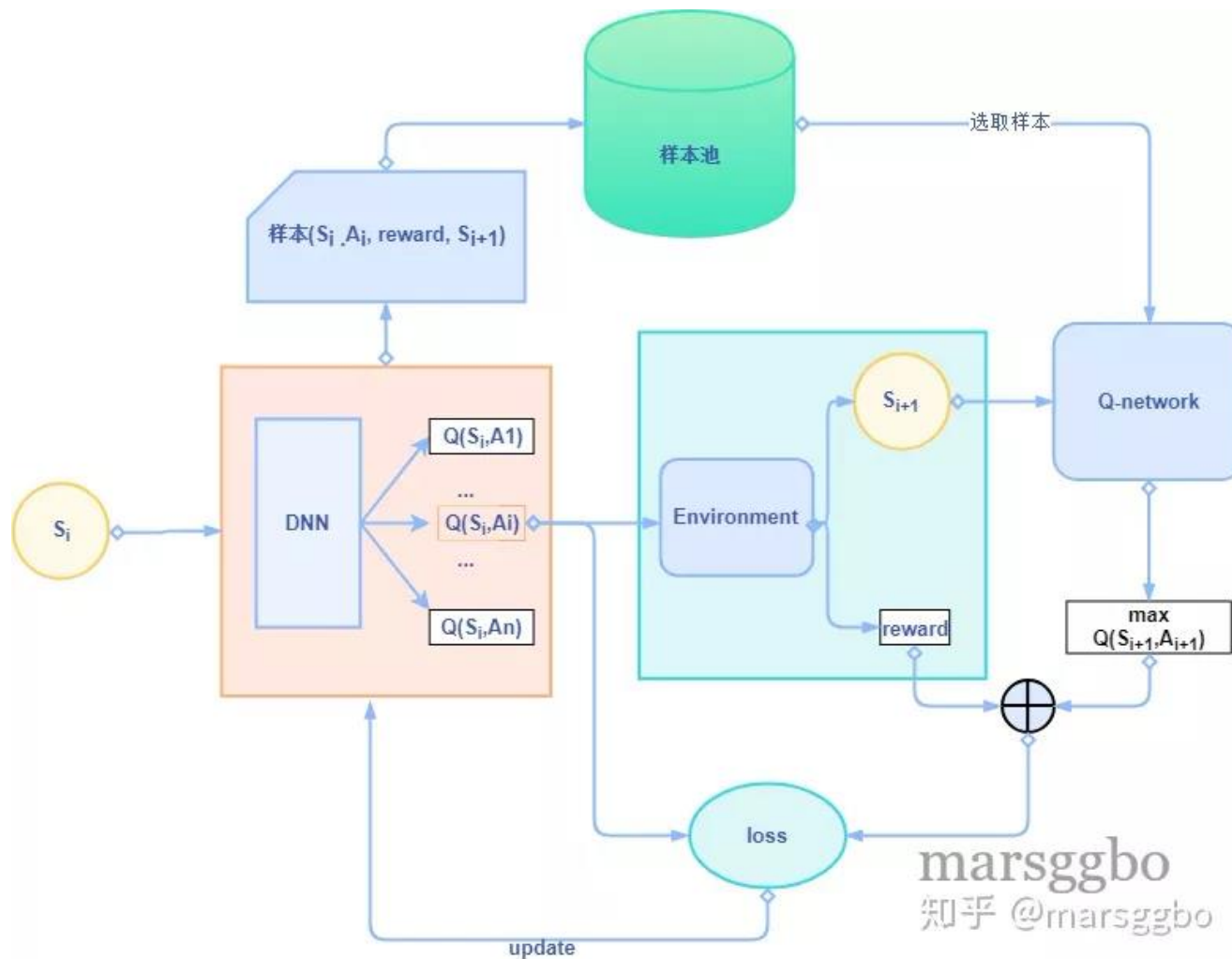
y : label

- ✓ treat as one data point for regression

- ✓ want a method that can learn on-line (update after each step)

- ✓ TD + NN

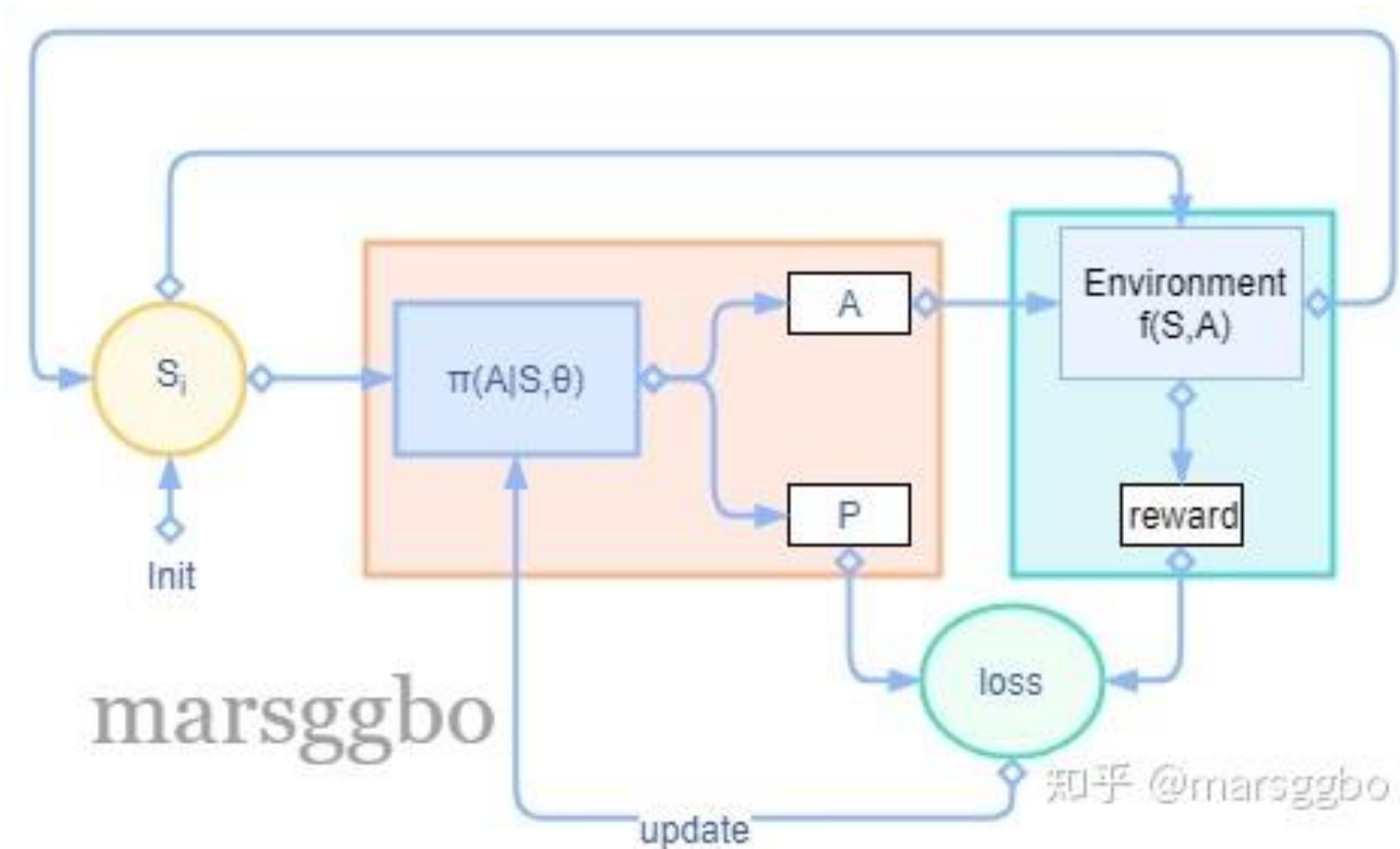
Deep Q-Network Diagram



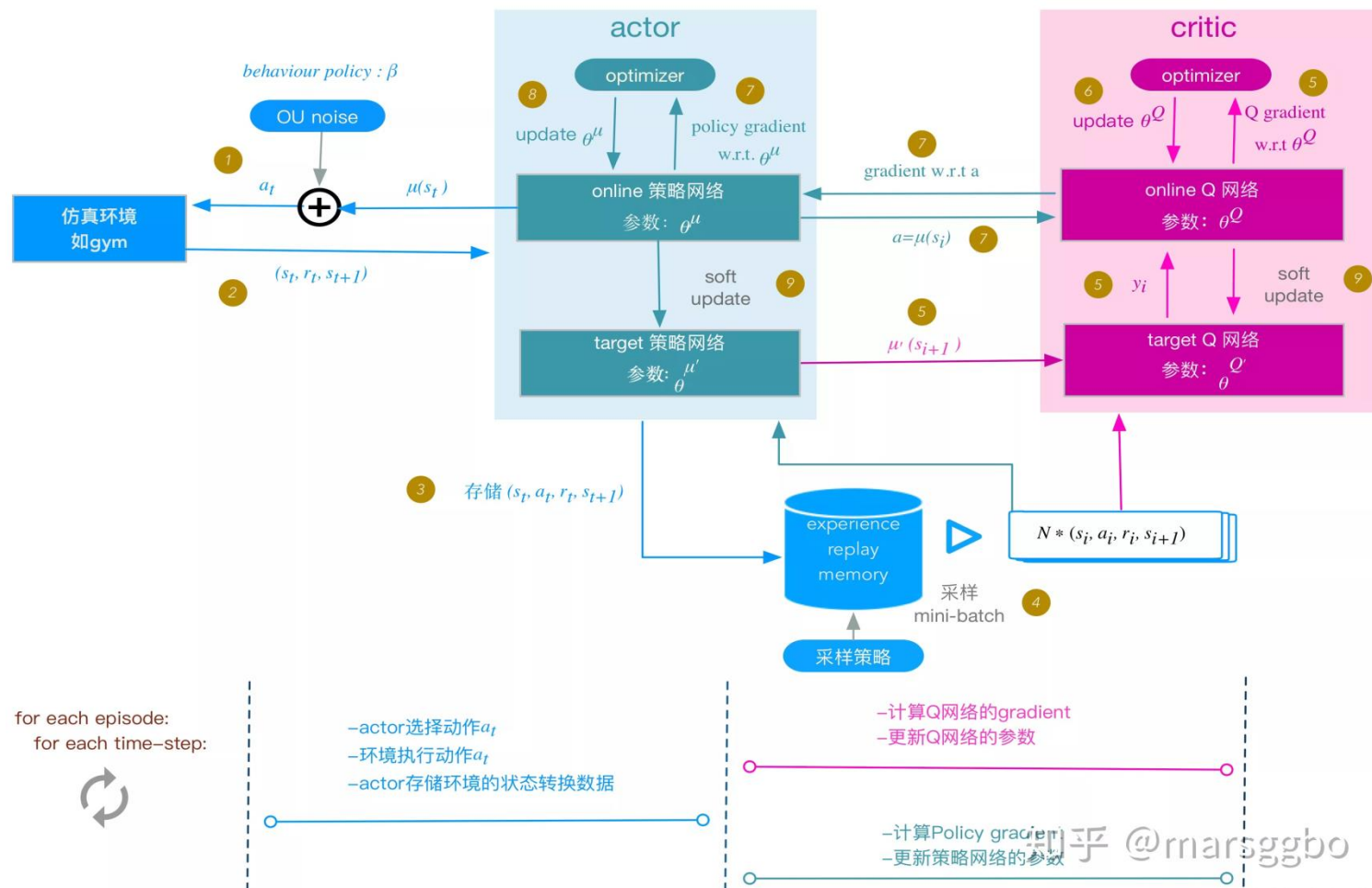
Policy Approximation

- represent π_t as a parameterized classification function
 - ✓ softmax regression, neural net, ...
 - ✓ cost function:
$$L(\theta) = - \sum \log p(x) f(x) = - \sum \log \pi(A|S, \theta) f(S, A)$$
 - update parameters instead of entries in a table
 - ✓ better generalization
 - fewer parameters and updates affect “similar” states-actions as well
 - Policy gradient for update
 - $$g(\theta) = \nabla_{\theta} L(\theta) = - \sum \nabla_{\theta} \log \pi(A|S, \theta) f(S, A)$$
 - ✓ treat as one episode for classification
 - ✓ want a method that can learn off-line (update after each episode)
 - ✓ Monte Carlo + NN
-

Policy Gradient Diagram



Actor-Critic Diagram



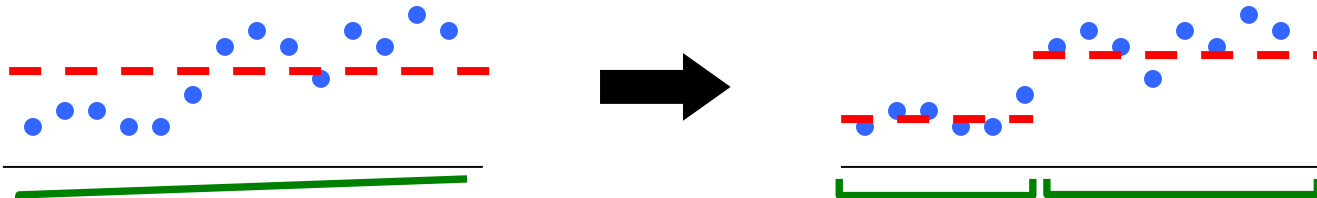
Splitting and Aggregation

- want to discretize the state space
 - learn the best discretization during training

- **splitting of state space**

start with a single state

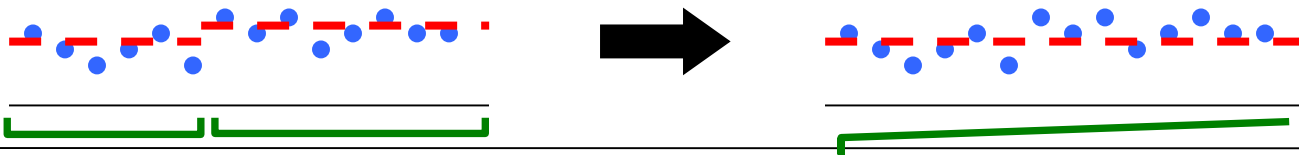
split a state when different *parts of that state* have different values



- **state aggregation**

start with many states

merge states with similar values



Designing Rewards

- ❑ **robot in a maze**
episodic task, not discounted, +1 when out, 0 for each step
- ❑ **chess**
GOOD: +1 for winning, -1 losing
BAD: +0.25 for taking opponent's pieces
high reward even when lose
- ❑ **rewards**
rewards indicate what we want to accomplish
NOT how we want to accomplish it
- ❑ **shaping**
positive reward often very “far away”
rewards for achieving sub-goals (domain knowledge)
also: adjust initial policy or initial value function



Summary

❑ **Reinforcement learning**

use when need to make decisions in uncertain environment

❑ **solution methods**

- ✓ dynamic programming
need complete model
- ✓ Monte Carlo
- ✓ time-difference learning (SARSA, Q-learning)
- ✓ Approximation functions (Deep Q-learning, PG, actor-critic)

❑ **most work**

simple algorithms

need to design features, state representation, rewards

Summary

- Reinforcement Learning Introduction
 - Dynamic Programming Approaches
 - Monte Carlo Approaches
 - Temporal Difference Approaches
 - Bayesian Reinforcement Learning
 - Deep Reinforcement Learning
-