PATTERN RECOGNITION AND MACHINE LEARNING

CHAPTER 16: 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?

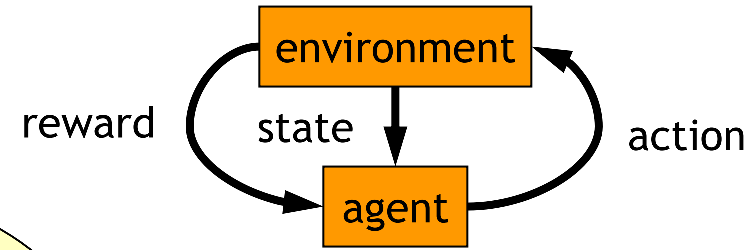
7、What is value approximation?

8、What is policy approximation?

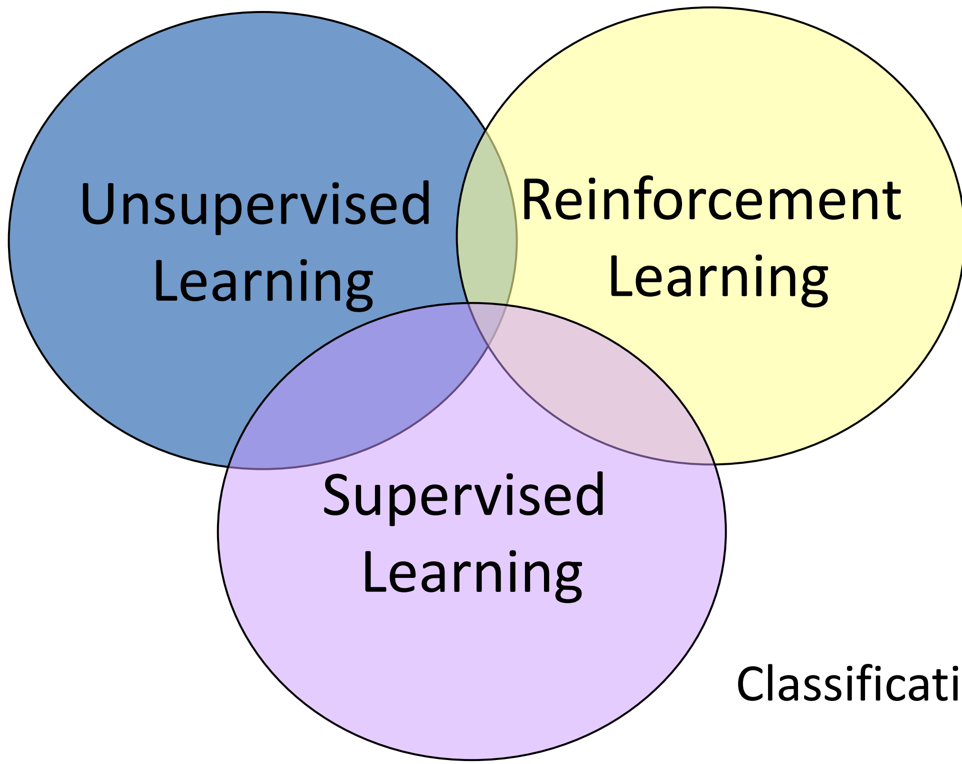
Outlines

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

Machine Learning

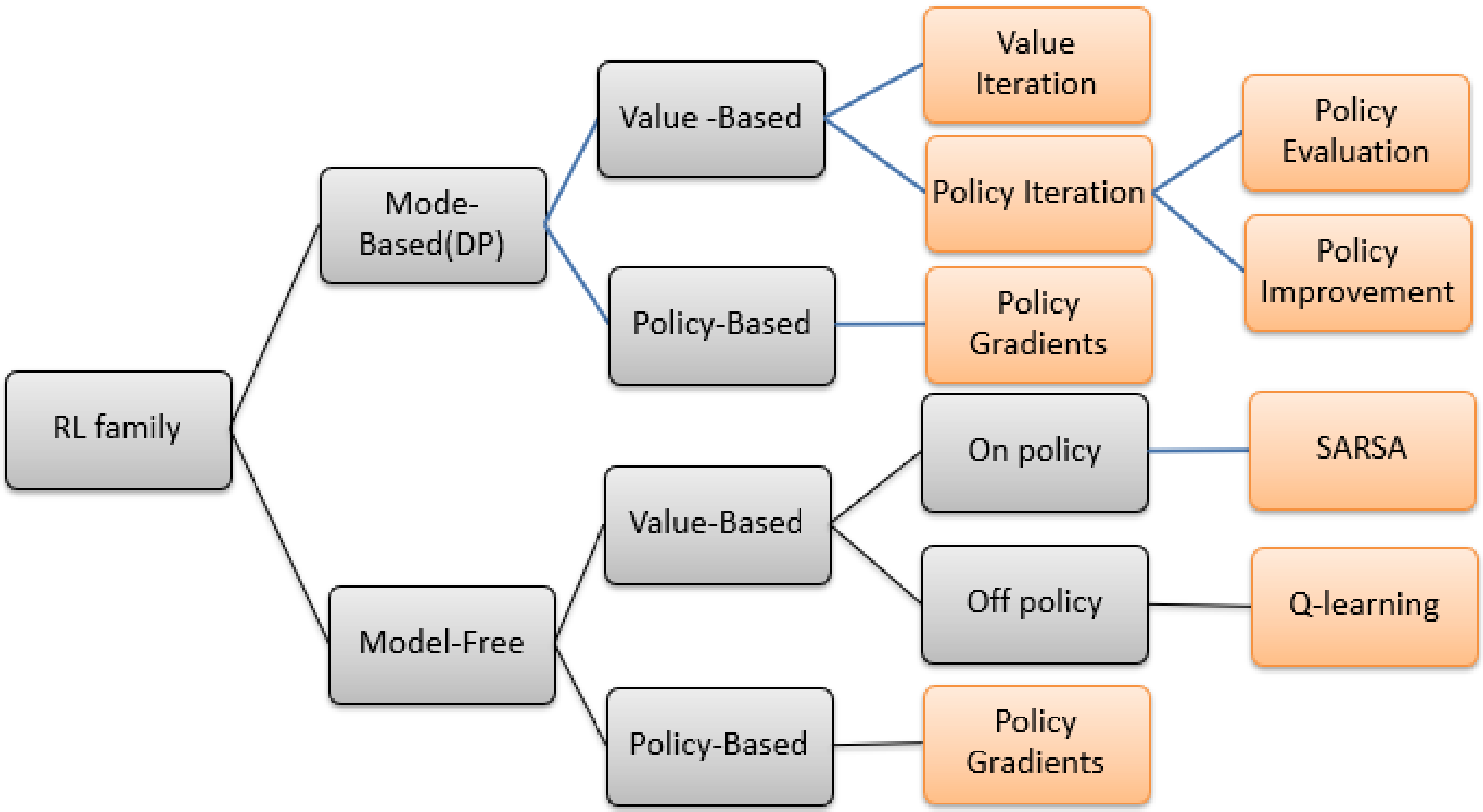


Clustering
Dimension reduction

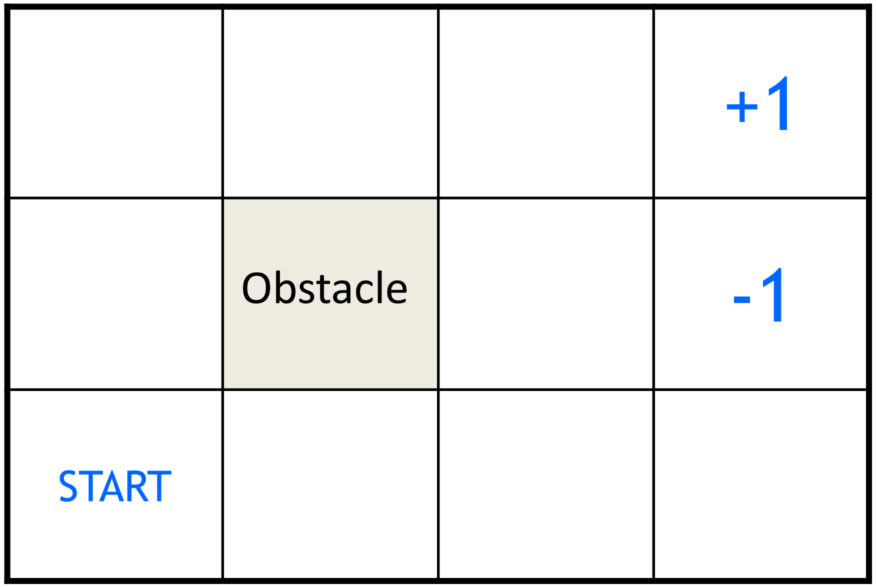


Classification, Regression

Reinforcement Learning



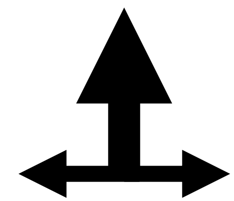
Reinforcement Learning Problem



actions: UP, DOWN, LEFT, RIGHT

Policy

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



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

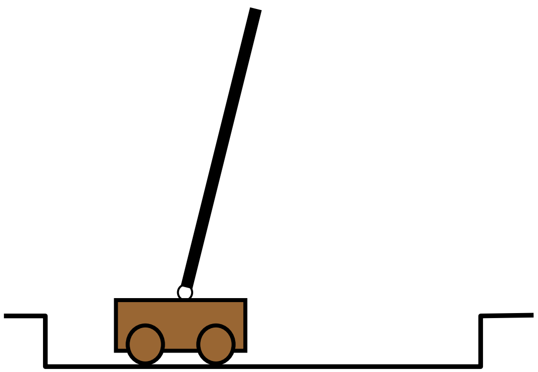
reward -0.04 for each step

what's the strategy to achieve max reward?

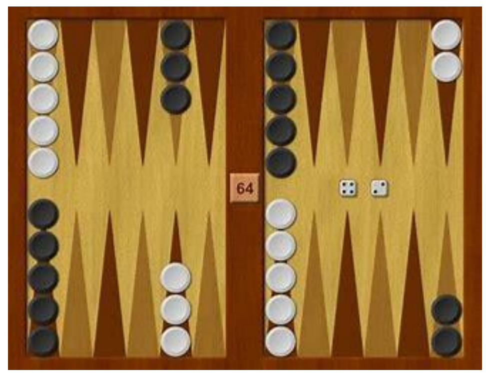
what if the actions were deterministic?

Reinforcement Learning Problems

Pole balancing



Games

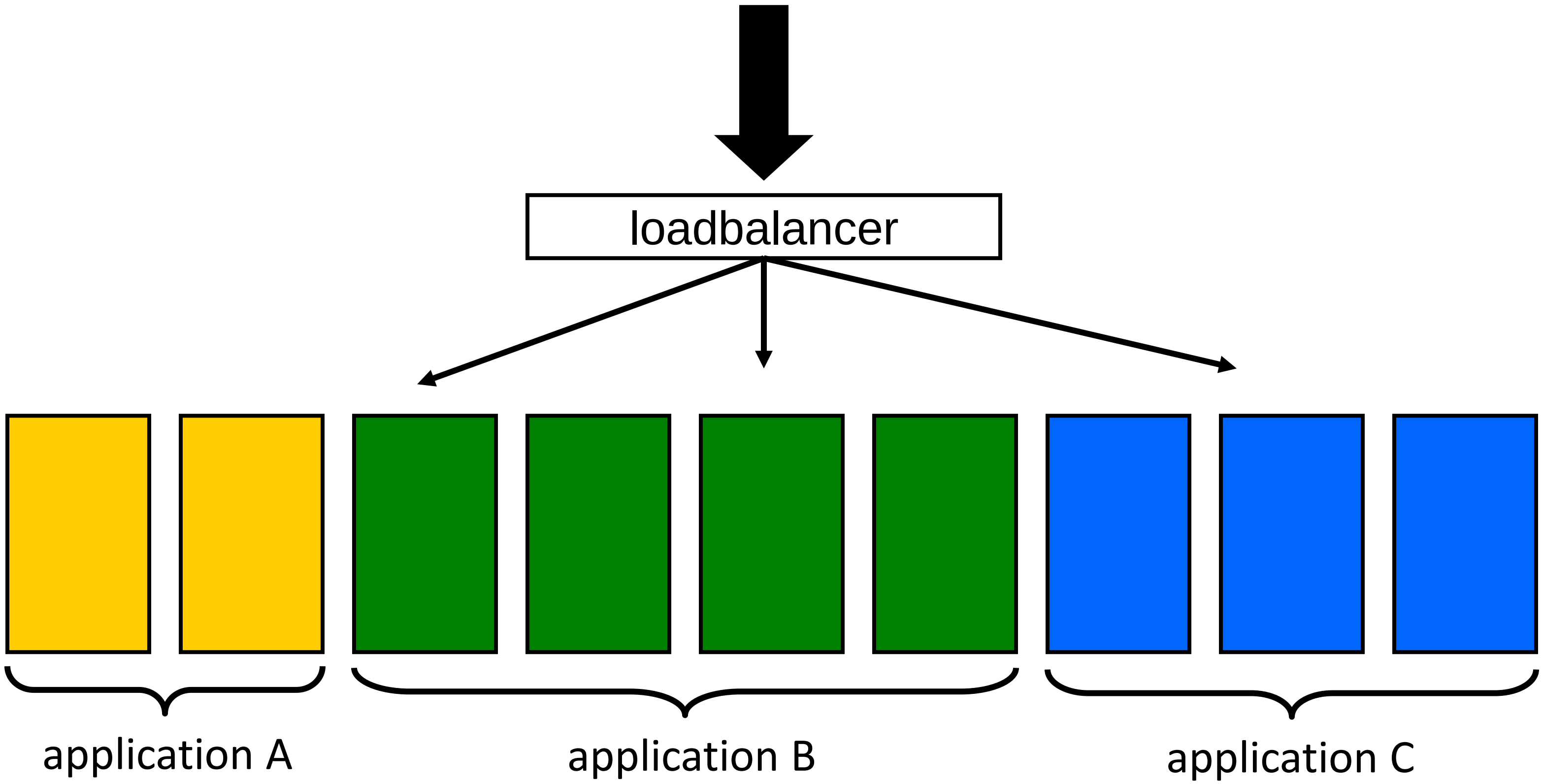


Drones

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

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

$$\boldsymbol{a} = \text{stick}$$

££££££

Policy

$$\boldsymbol{a} = \pi(x)$$
$$\boldsymbol{a} \sim \pi(\cdot | x)$$

Reward Function

$$r = R(\boldsymbol{x}_t, \boldsymbol{a}_t)$$
$$r \sim R(. \mid \boldsymbol{x}, \boldsymbol{a})$$

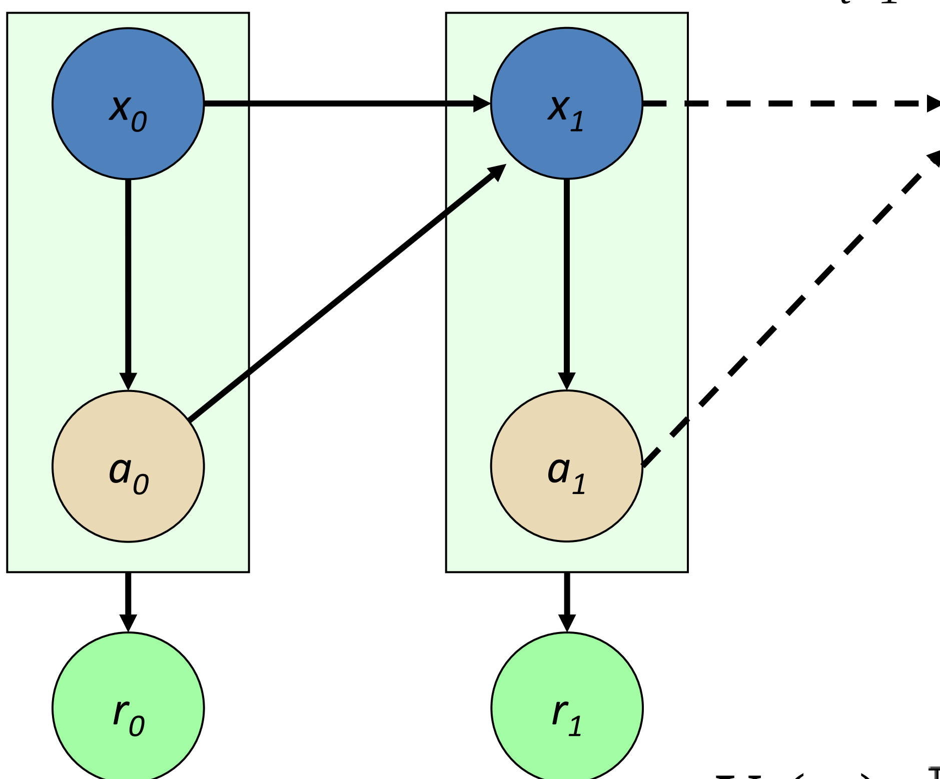
Markov Decision Process (MDP)

Transition Probability

$x_{t+1} \sim p\,(\cdot|\, \boldsymbol{x}_t, \boldsymbol{a}_t\,)$

Policy

$a \sim \pi\,(\cdot|\, \boldsymbol{x}\,)$



Reward function

$r \sim R(.\mid \boldsymbol{x}, \boldsymbol{a})$

$$\mathbf{V}^{\pi}(\mathbf{x}_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t\, r_t\right]$$

Value Function

$$V^{\mu}(\boldsymbol{x}_0) \;=\; \mathbb{E}_{a_0,a_1,\ldots;x_1,x_2,\ldots}\left[\,\sum_{t=0}^{\infty}\gamma^t\; r(\boldsymbol{x}_t,\mu(\boldsymbol{x}_t))\,\right]$$

$$=\; \mathbb{E}_{a_0,a_1,\ldots;x_1,x_2,\ldots}\left[\; r(\boldsymbol{x}_0,\mu(\boldsymbol{x}_0)) \;+\; \gamma\sum_{t=1}^{\infty}\gamma^{t-1}\; r(\boldsymbol{x}_t,\mu(\boldsymbol{x}_t))\;\right]$$

$$=\; \sum_{\boldsymbol{a}_0}\mu(\boldsymbol{a}_0 \mid \boldsymbol{x}_0)\;\left(r(\boldsymbol{x}_0,\boldsymbol{a}_0) \;+\; \sum_{x_1}p(\boldsymbol{x}_1 \mid \boldsymbol{x}_0,\boldsymbol{a}_0)\;\gamma\;\mathbb{E}_{a_1,a_2,\ldots;x_2,x_3,\ldots}\left[\;\sum_{t=1}^{\infty}\gamma^{t-1}\; r(\boldsymbol{x}_t,\mu(\boldsymbol{x}_t))\right]\right)$$

$$=\; \sum_{\boldsymbol{a}_0}\mu(\boldsymbol{a}_0 \mid \boldsymbol{x}_0)\;\left(r(\boldsymbol{x}_0,\boldsymbol{a}_0) \;+\; \gamma\sum_{x_1}\; p(\boldsymbol{x}_1 \mid \boldsymbol{x}_0,\boldsymbol{a}_0)\; V^{\mu}(\boldsymbol{x}_1)\right)$$

$\mu = \pi$

Optimal Policy

Assume one optimal action per state

$$V^*(\boldsymbol{x}_0) = \max_{\boldsymbol{a}_0} \left(\underline{r(\boldsymbol{x}_0, \boldsymbol{a}_0)} + \gamma \sum_{x_1} \underline{p(\boldsymbol{x}_1 \mid \boldsymbol{x}_0, \boldsymbol{a}_0)} V^*(\boldsymbol{x}_1) \right)$$

Value Iteration

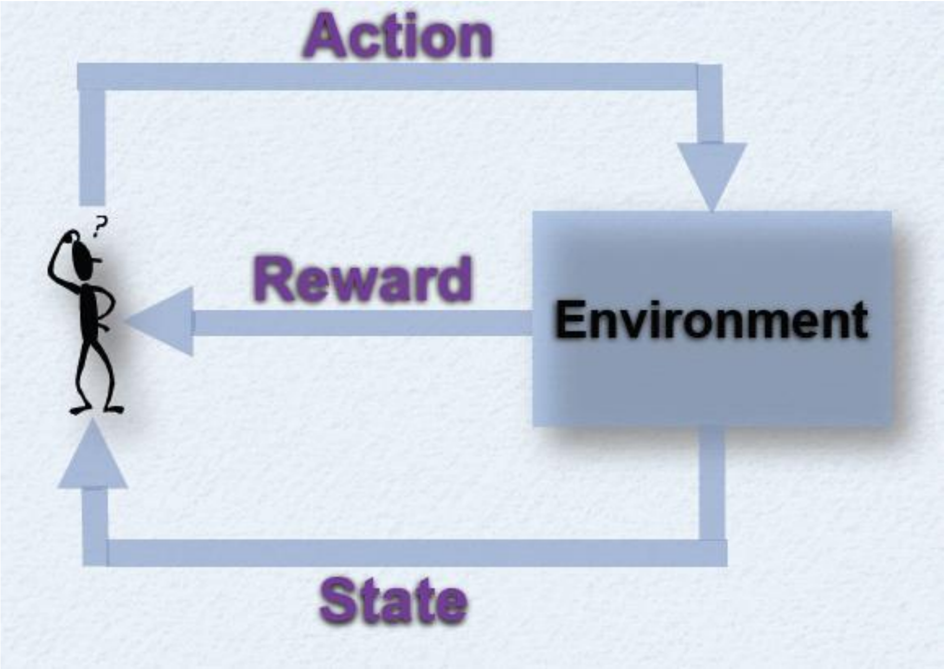
$$a_0^* \sim \pi^*(\cdot \mid \boldsymbol{x}_0)$$

$$a_1^* \sim \pi^*(\cdot \mid \boldsymbol{x}_1)$$

$$a_2^* \sim \pi^*(\cdot \mid \boldsymbol{x}_2)$$

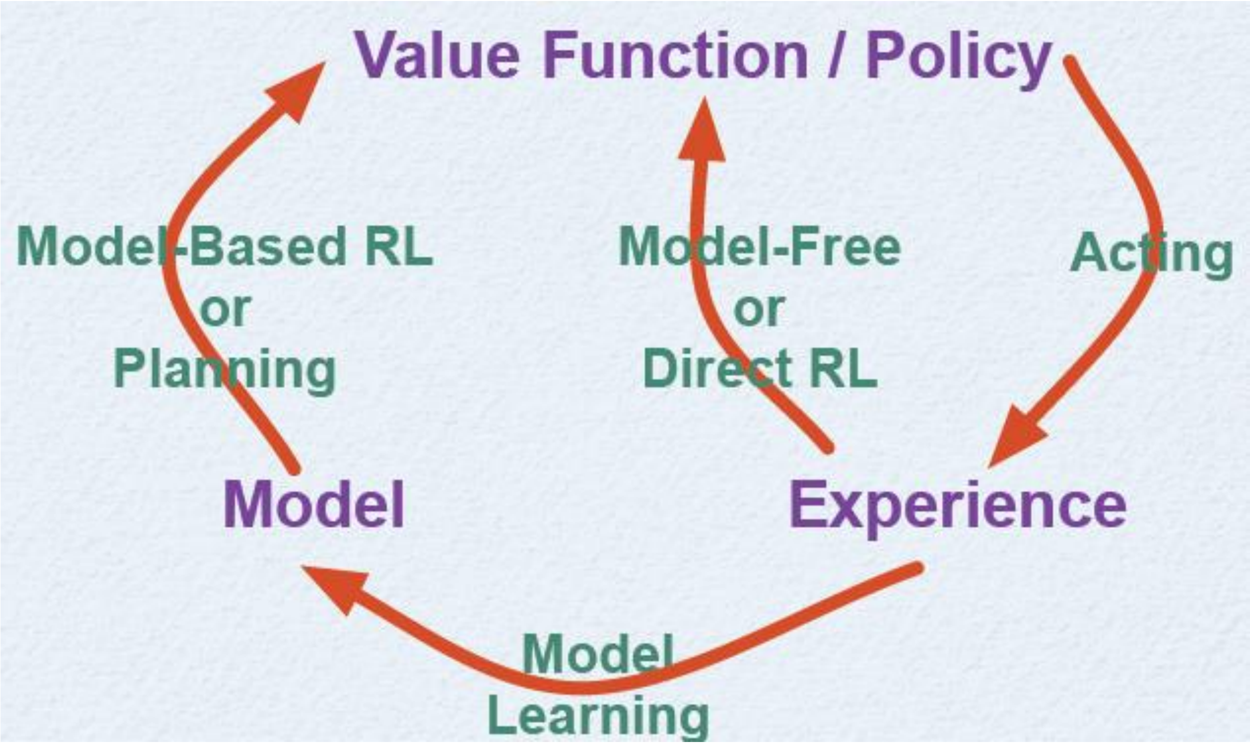
Unknown

Reinforcement Learning



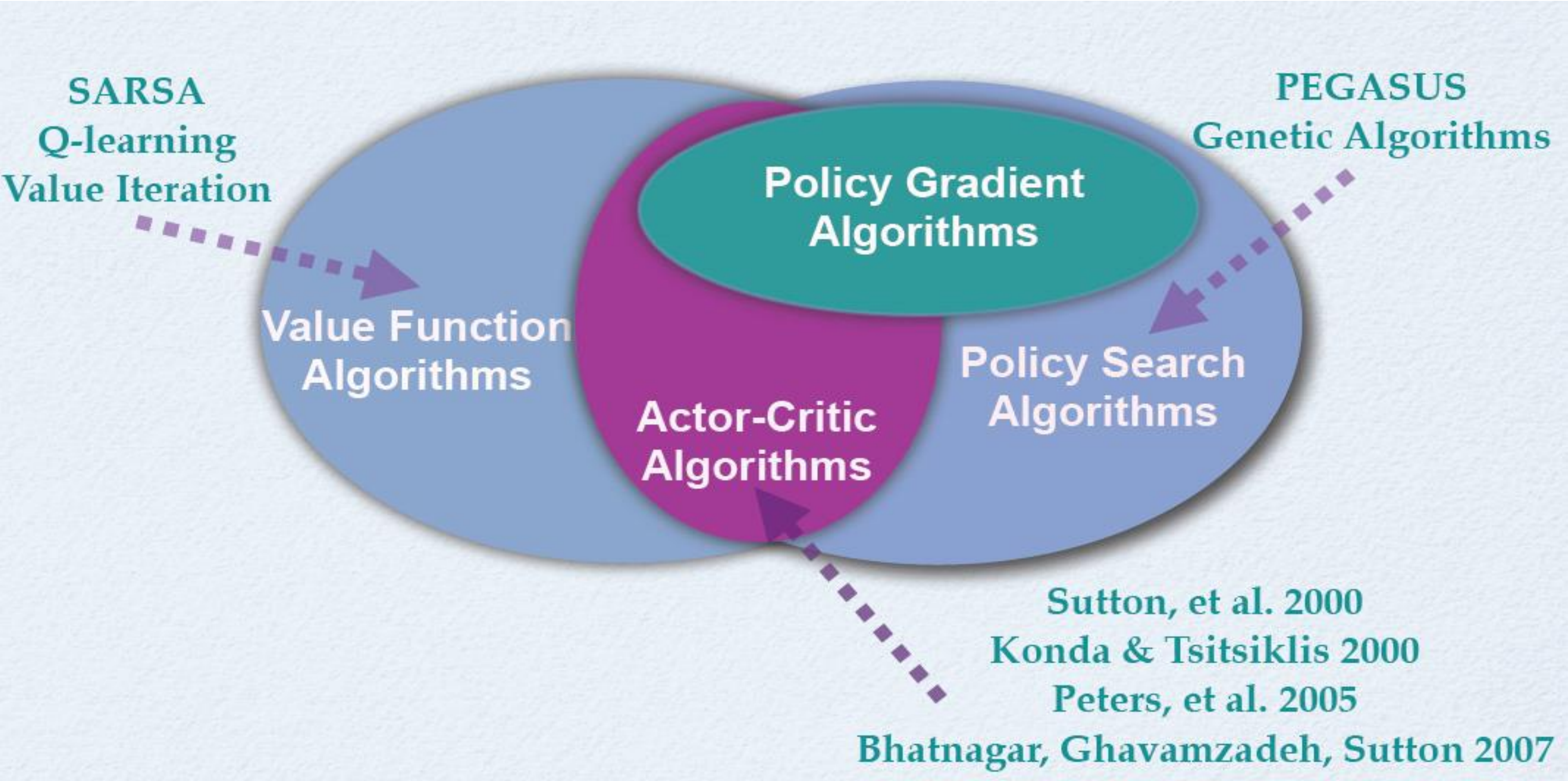
- **RL Problem:** Solve MDP when reward/transition models are unknown
- **Basic Idea:** Use samples obtained from agent's interaction with environment

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



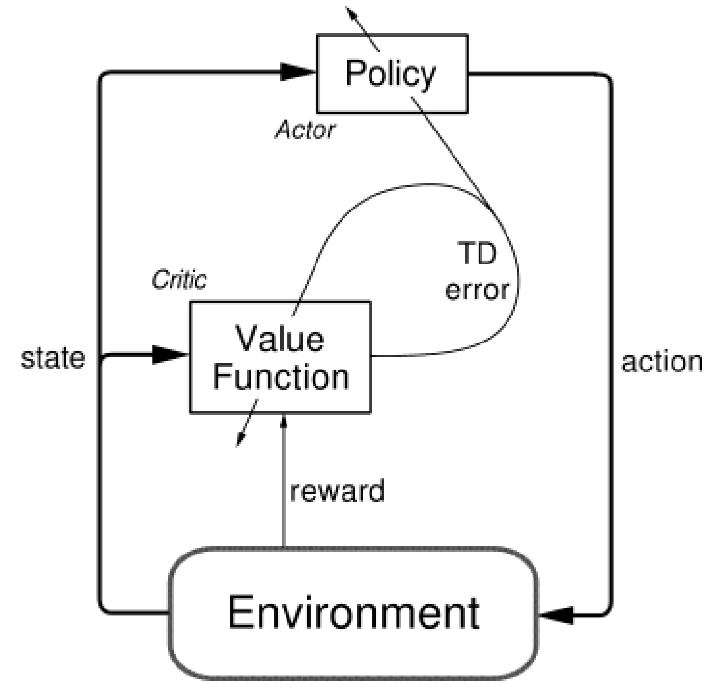
Value-Based RL Solutions

- 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)

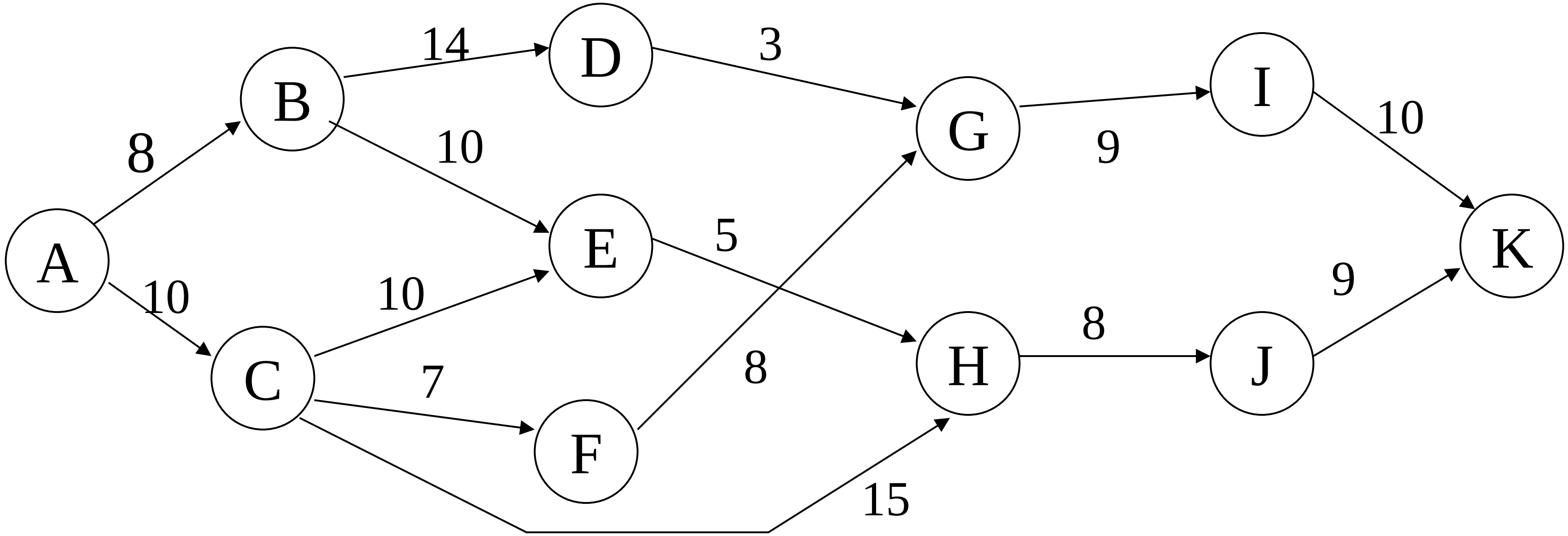
2. **Control:** Extract policy
   - On-Policy
   - Off-Policy

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?



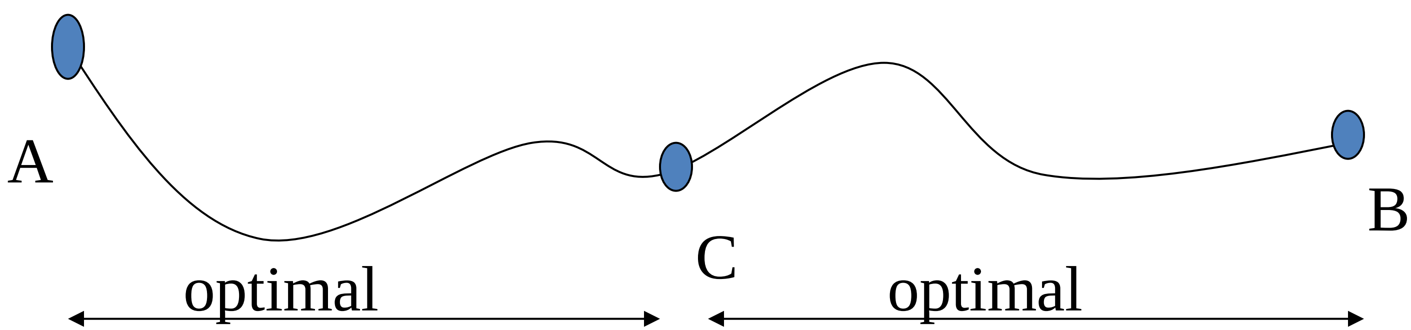
Dynamic Programming

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

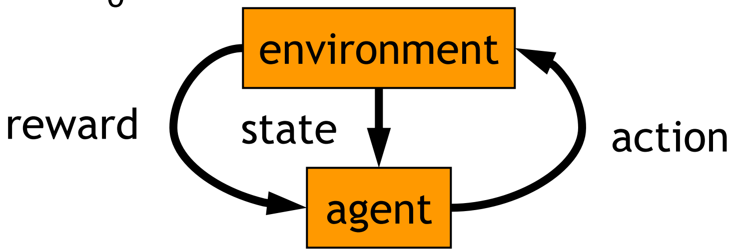
policy evaluation: compute V^{π} from π

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], 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
  π(s) or π(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**
  $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  infinite value for continuing tasks

- **discounted rewards**
  $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  value bounded if rewards bounded

Value Functions

- **state value function**: $V^{\pi}(s)$

  expected return when starting in *s* and following π

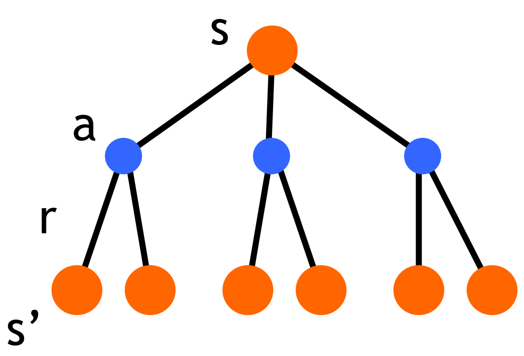
- **state-action value function**: $Q^{\pi}(s,a)$

  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 $Q^{\pi}(s,a)$



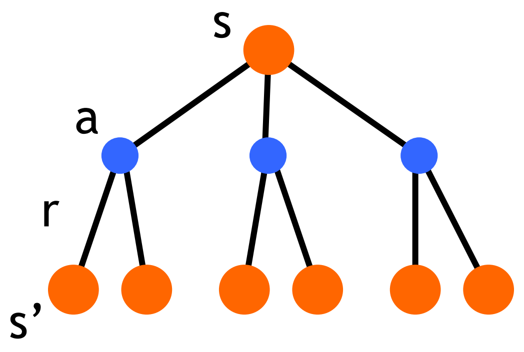
- **Bellman equation**

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{\pi}(s') \right] = \sum_{a} \pi(s,a) Q^{\pi}(s,a)$$

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

$$V^*(s) = \max_a \overbrace{\sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^*(s') \right]}^{Q(s,a)}$$



- **having Q*(s,a) makes it even simpler**

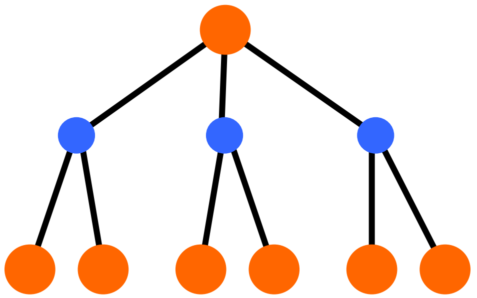
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Policy Evaluation/Improvement

- **policy evaluation**: π -> V^{π}
  - ✓ Bellman equations define a system of n equations
  - ✓ could solve, but will use iterative version

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{k'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

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



- **policy improvement**: V^{π} -> π'

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$
$$= \arg\max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^{\pi}(s') \right]$$

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

Policy/Value Iteration

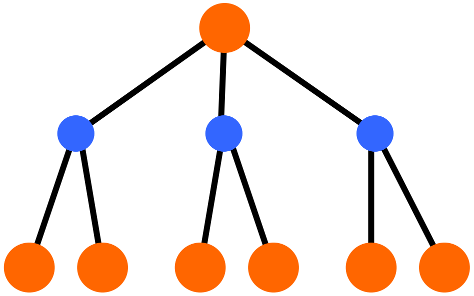
Policy iteration

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E V^*$$

two nested iterations; too slow

don't need to converge to V^{π_k}

just move towards it



Value iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

use Bellman optimality equation as an update

converges to V*

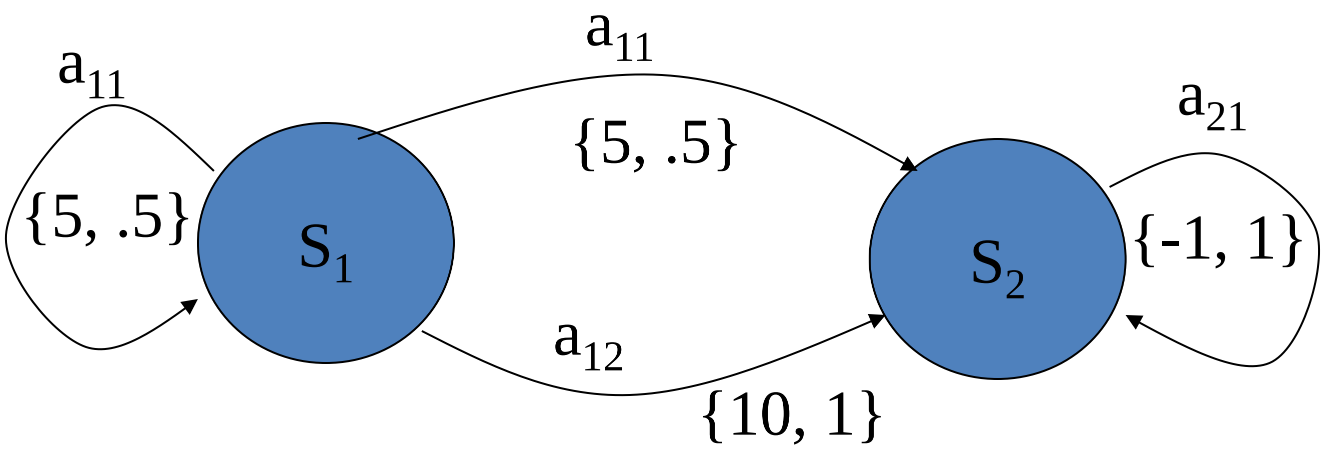
Example

Decision epochs: T = {1, 2, …, N}

State : $S = \{s_1, s_2\}$ Actions: $A_{s1} = \{a_{11}, a_{12}\}$, $A_{s2} = \{a_{21}\}$

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

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



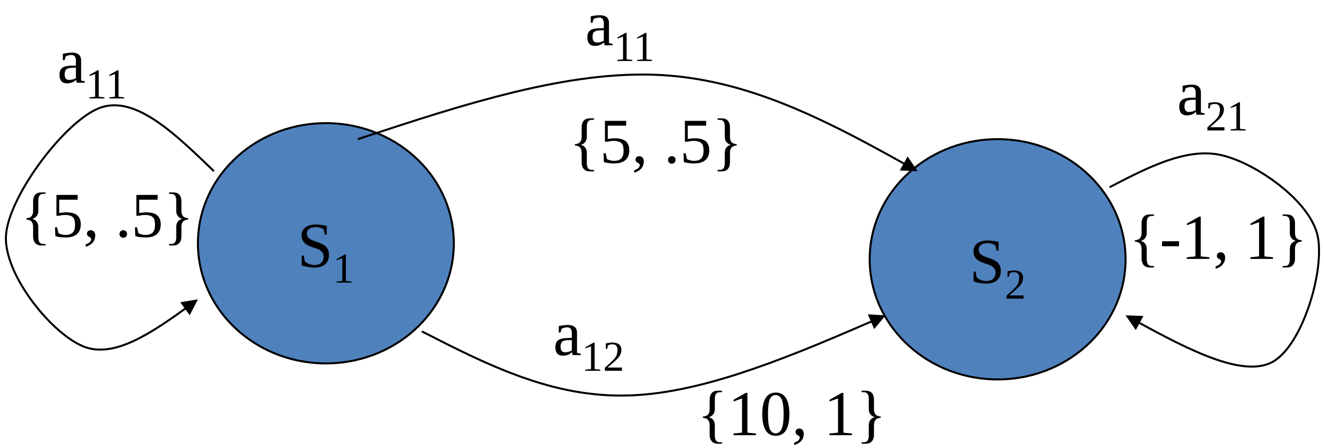
Example: Deterministic Markov Policy

Decision epoch 1:

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

Decision epoch 2:

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



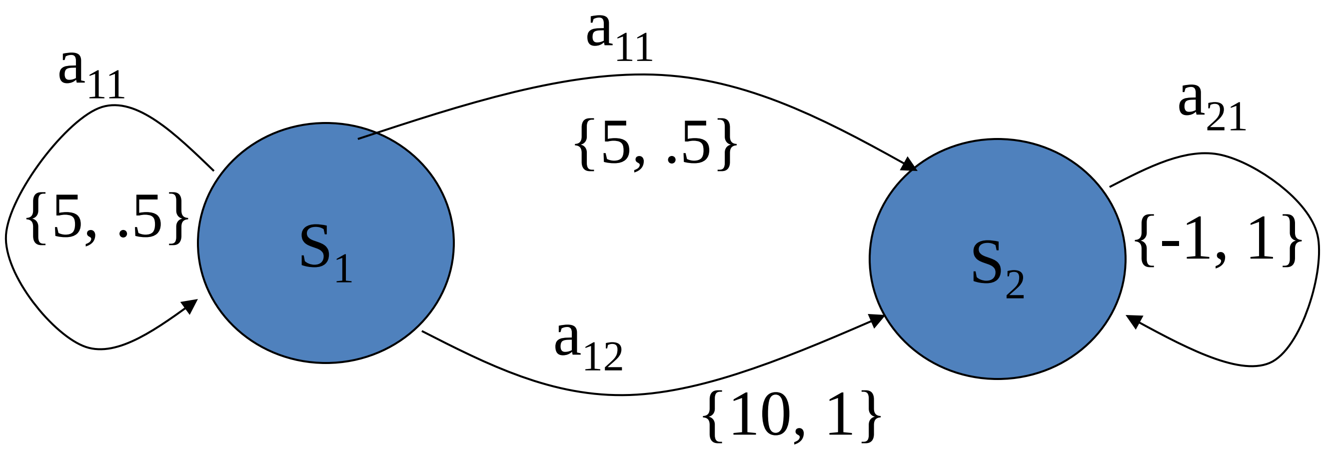
Example: Randomized Markov Policy

Decision epoch 1:

$P_{1,\, s1}(a_{11}) = 0.7$, $P_{1,\, s1}(a_{12}) = 0.3$; $P_{1,\, s2}(a_{21}) = 1$

Decision epoch 2:

$P_{2,\, s1}(a_{11}) = 0.4$, $P_{2,\, s1}(a_{12}) = 0.6$; $P_{2,\, s2}(a_{21}) = 1$



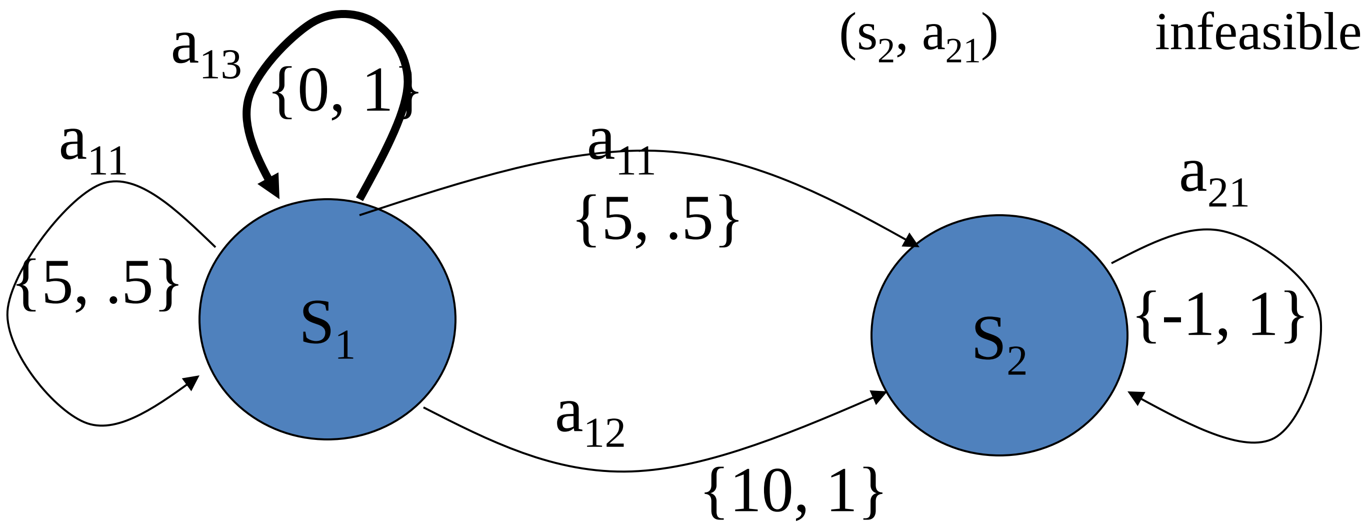
Example: Deterministic History-Dependent

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:

$P_{1,\,s1}(a_{11}) = 0.6$

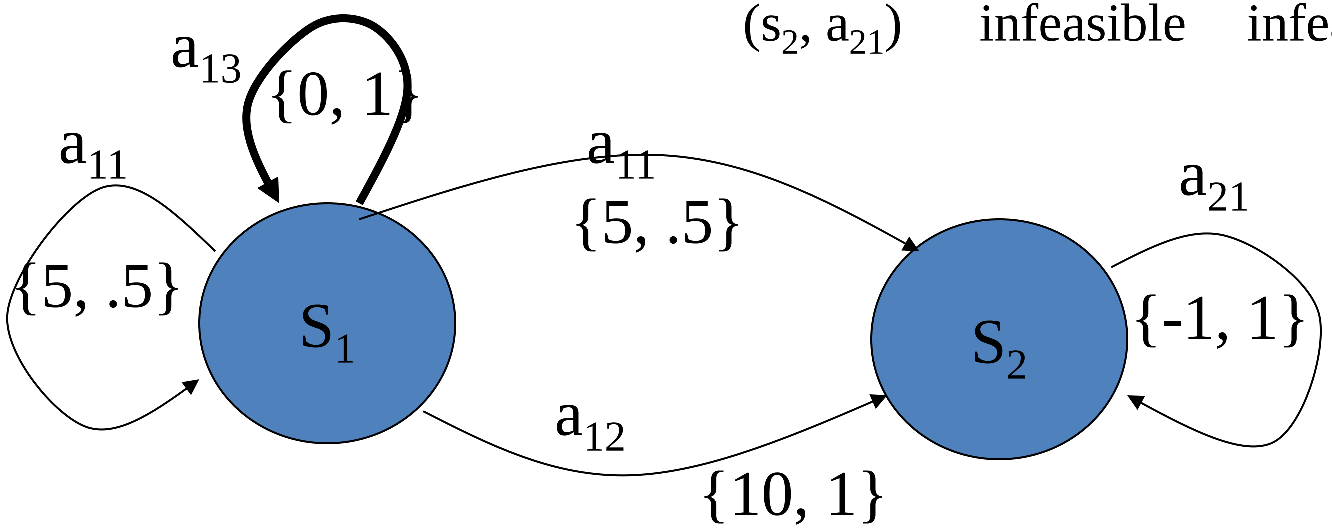
$P_{1,\,s1}(a_{12}) = 0.3$

$P_{1,\,s1}(a_{13}) = 0.1$

$P_{1,\,s2}(a_{21}) = 1$

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

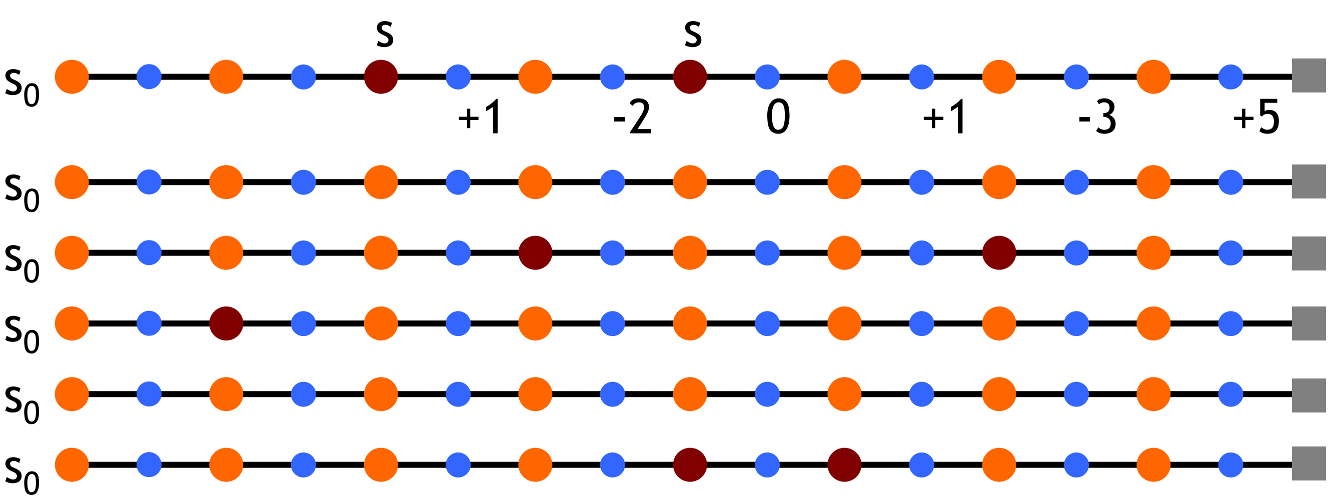
- want to estimate $V^{\pi}(s)$

  = 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



$R_1(s) = +2$

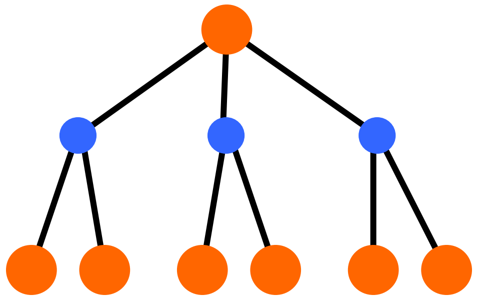
$R_2(s) = +1$

$R_3(s) = -5$

$R_4(s) = +4$

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

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

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

- 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-ε perform the optimal/greedy action
  with probability ε perform a random action

  will keep exploring the environment
  slowly move it towards greedy policy: ε -> 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

$$V(\boldsymbol{x}_t) \leftarrow V(\boldsymbol{x}_t) + \alpha(R_t - V(\boldsymbol{x}_t))$$

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:

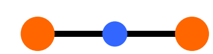
  have to wait until the end of episode to update

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

target

- simplest TD

  update after every step, based on the successor



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

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

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

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

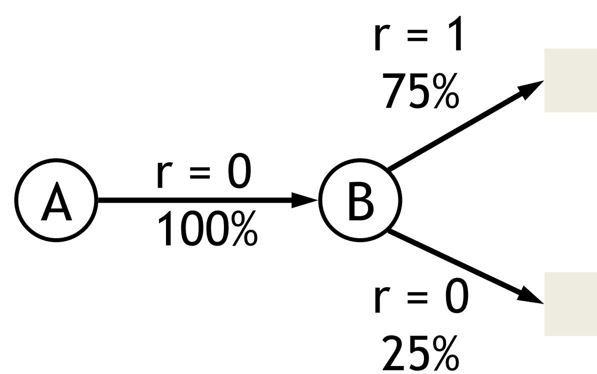
- observed the following 8 episodes:

A – 0, B – 0	B – 1	B – 1	B - 1
B – 1	B – 1	B – 1	B – 0

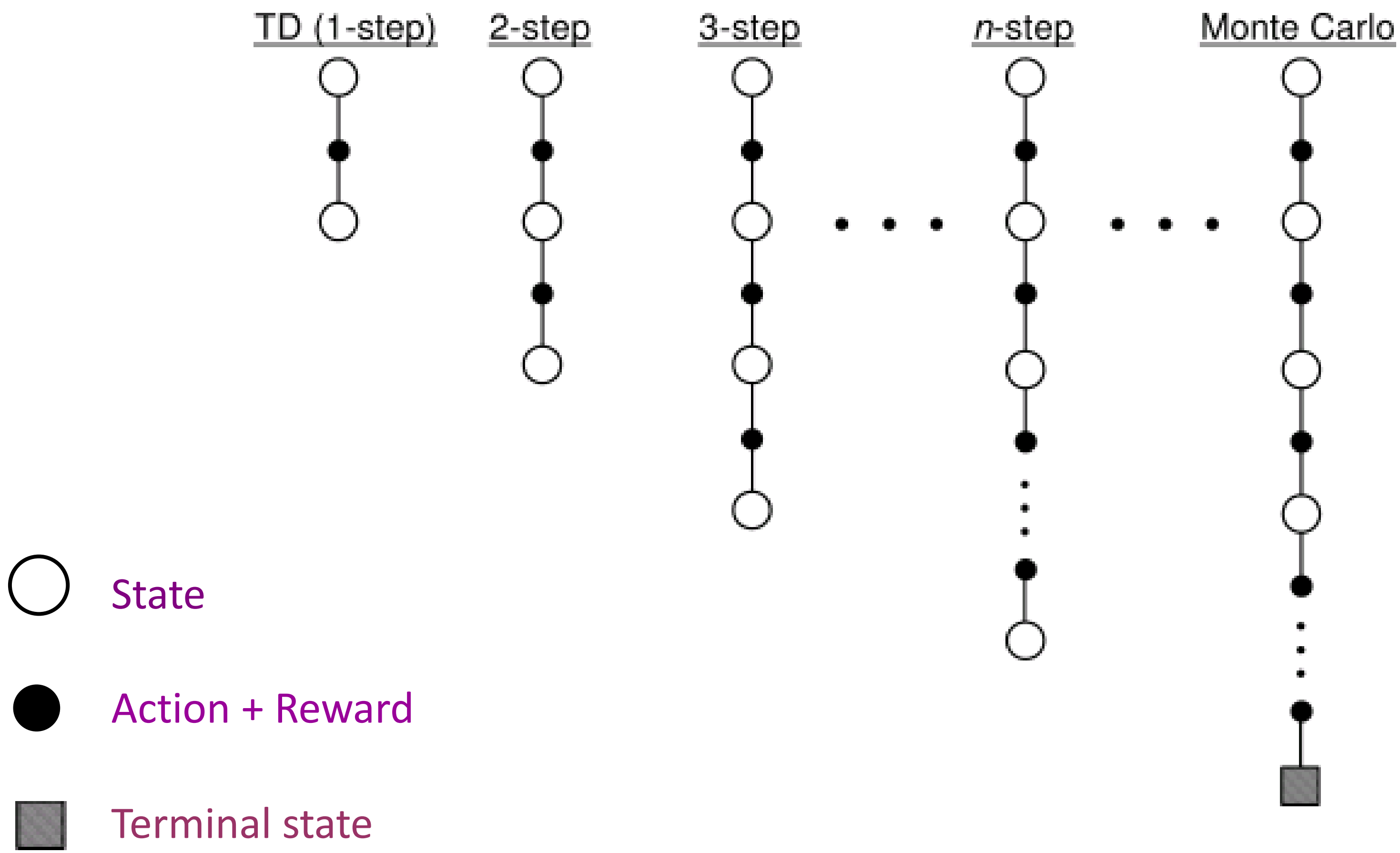
- MC and TD agree on V(B) = 3/4

- MC: V(A) = 0
  converges to values that minimize the error on training data

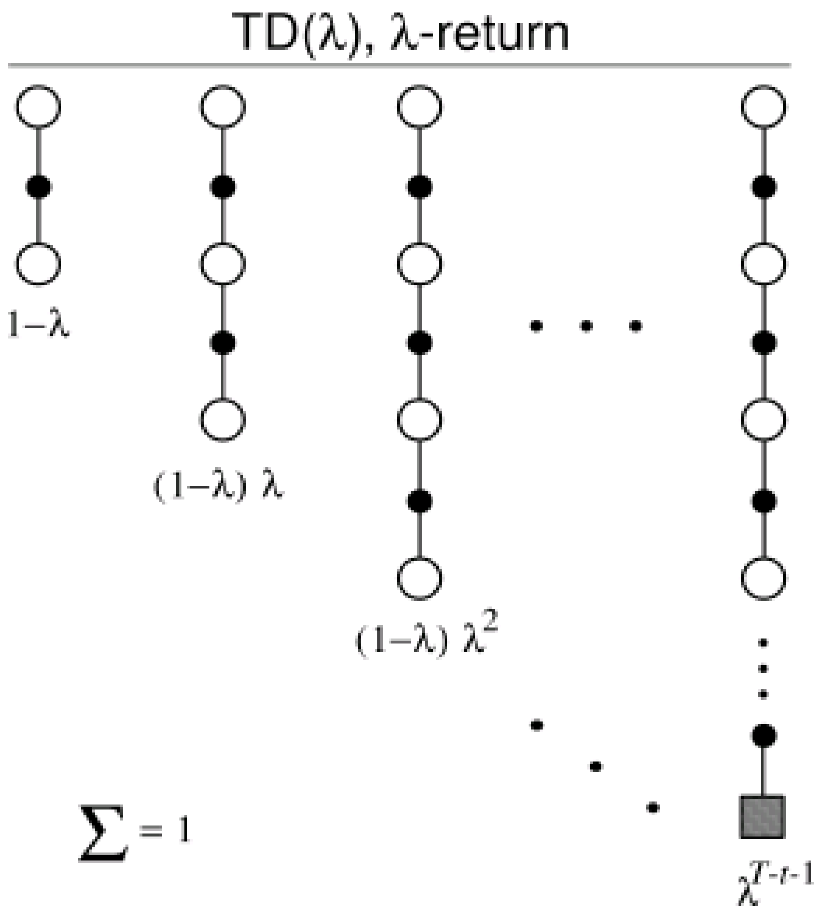
- TD: V(A) = 3/4
  converges to ML estimate
  of the Markov process



From TD to TD(λ)



From TD to TD(λ)



State

Action + Reward

Terminal state

SARSA & Q-learning

TD-Learning

On-Policy

Estimate value function for current policy

Off-Policy

Estimate value function for optimal policy

SARSA

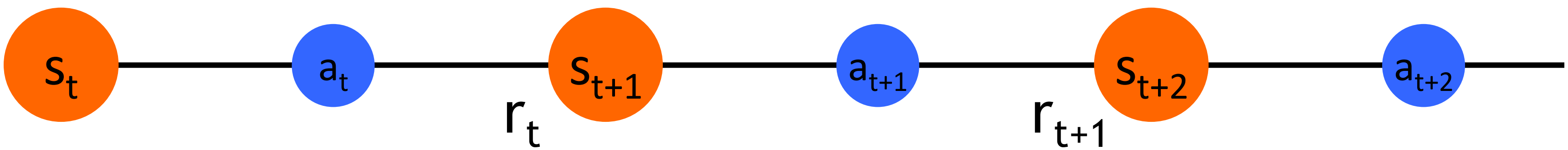
Q-Learning

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

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

SARSA

- again, need Q(s,a), not just V(s)



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

- 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

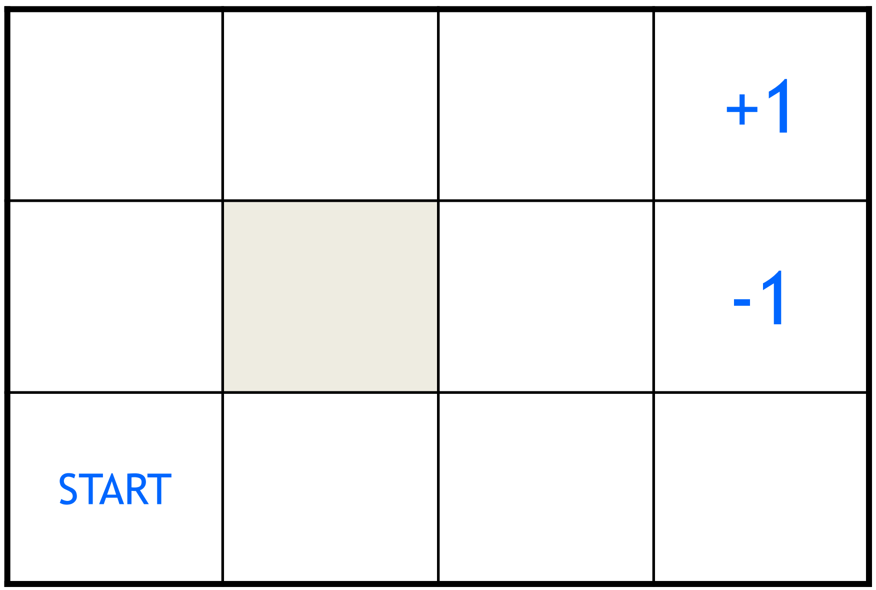
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

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

- **SARSA**

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

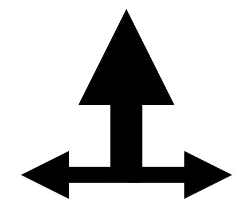
Robot in A Room



actions: UP, DOWN, LEFT, RIGHT

UP

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

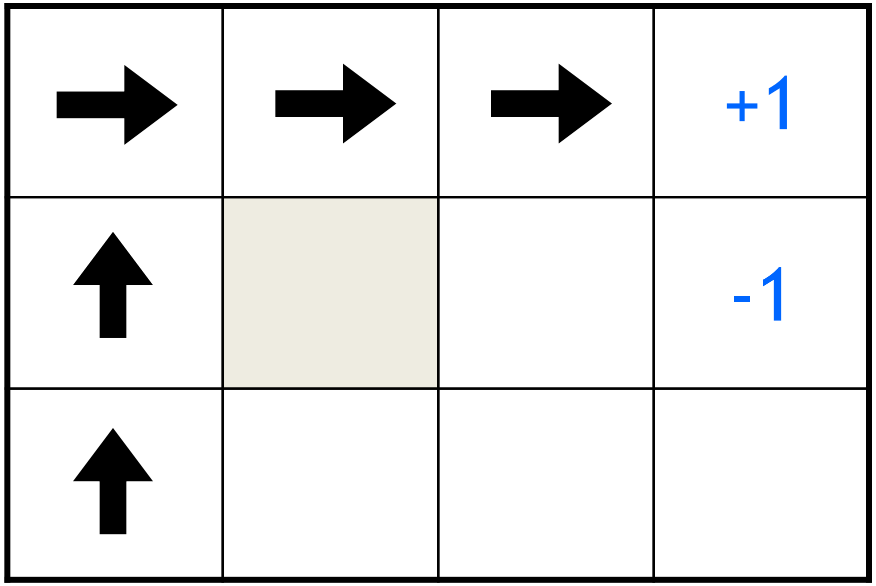


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

what is the solution?

Is This A Solution?



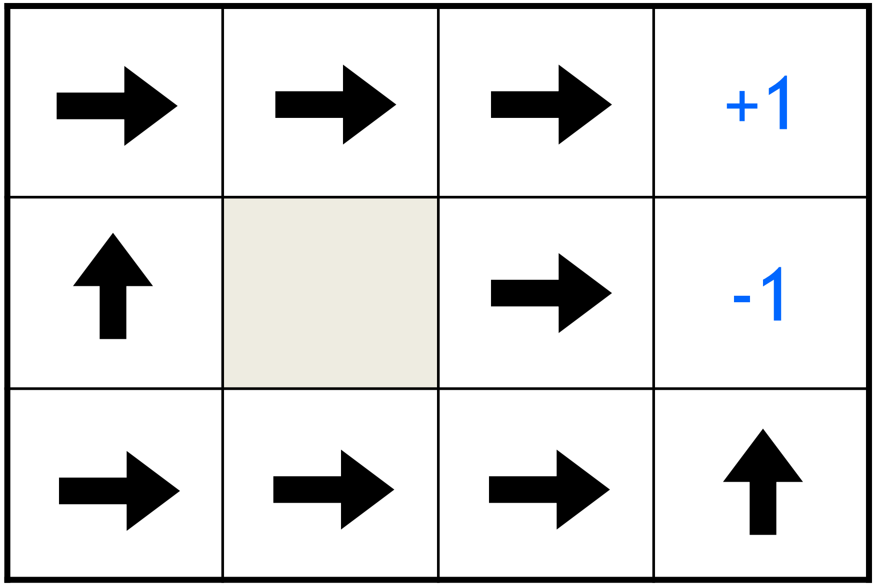
- only if actions deterministic
  not in this case (actions are stochastic)

- solution/policy
  mapping from each state to an action

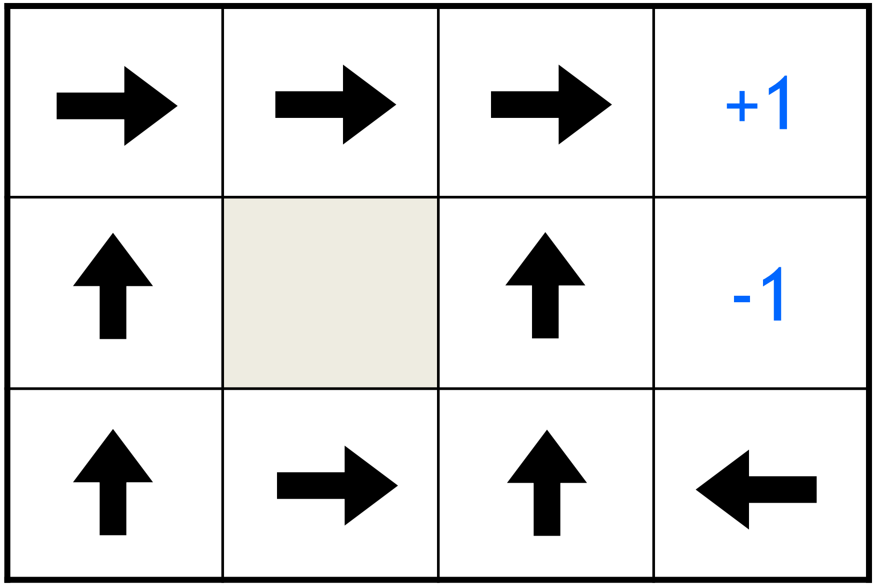
Optimal Policy

→	→	→	+1
↑		↑	-1
↑	←	←	←

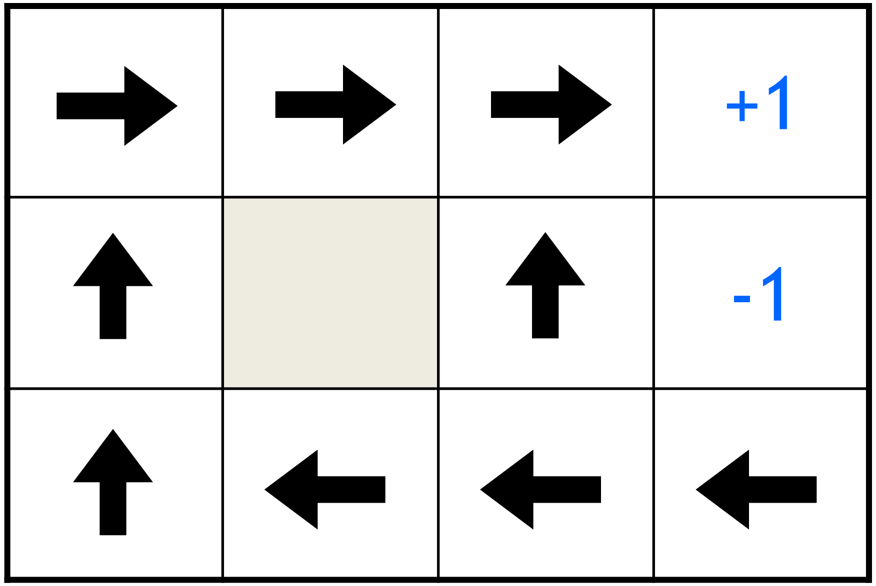
Reward for Each Step: -2



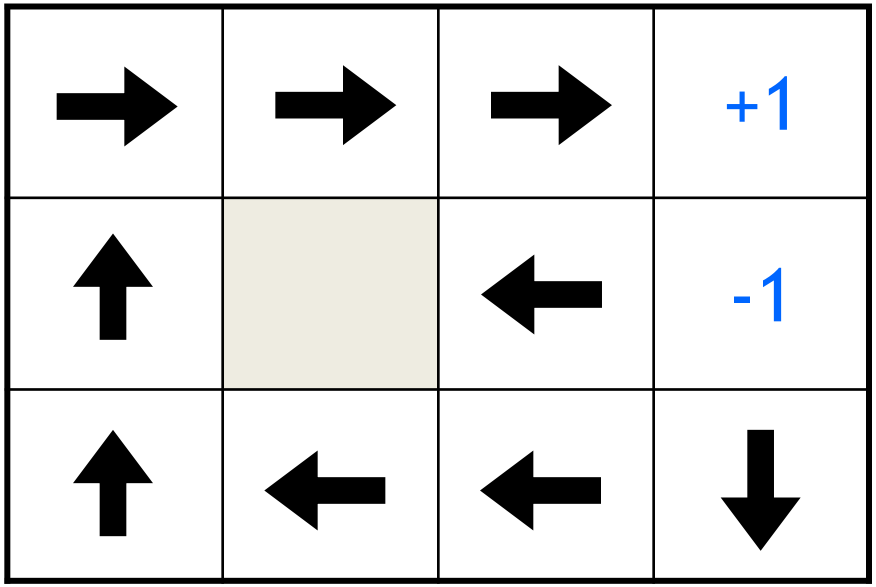
Reward for each step: -0.1



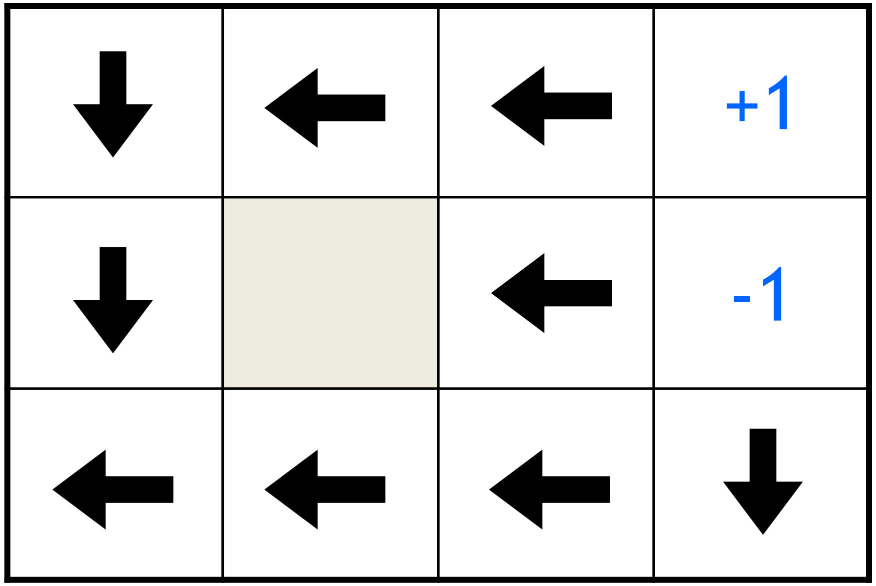
Reward for each step: -0.04



Reward for each step: -0.01



Reward for each step: +0.01

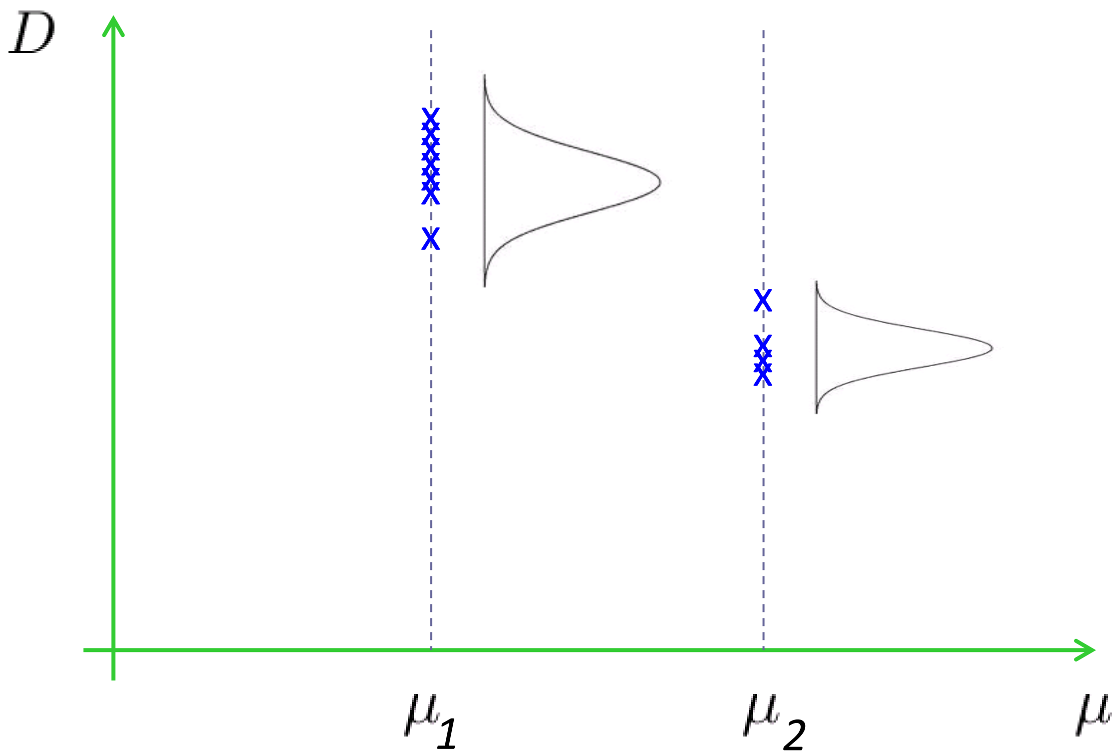


Outlines

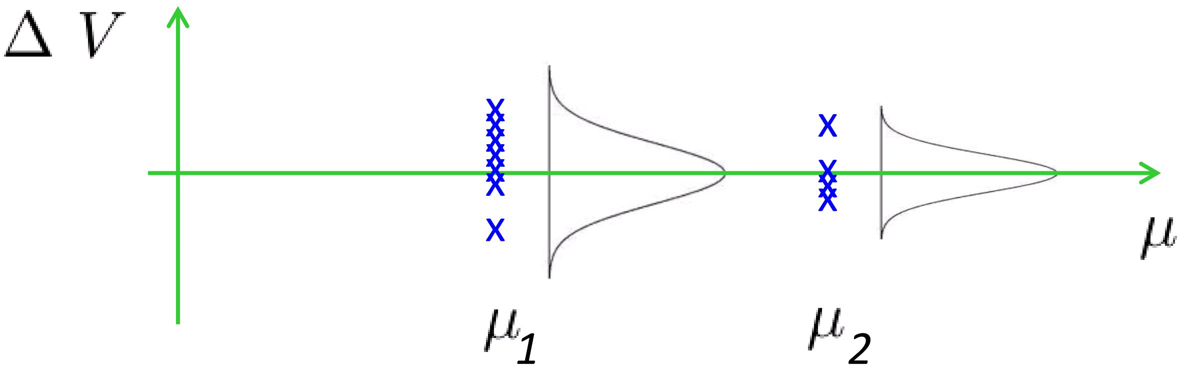
- Reinforcement Learning Introduction
- Dynamic Programming Approaches
- Monte Carlo Approaches
- Temporal Difference Approaches
- Bayesian Reinforcement Learning
- Deep Reinforcement Learning

Gaussian Process Temporal Difference

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



Gaussian Process Temporal Difference



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

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

Gaussian Process Temporal Difference

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

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

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

Gaussian Process Temporal Difference

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

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

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

Gaussian Process Temporal Difference

General noise covariance: $\boxed{\mathrm{Cov}[N_t] = \mathbf{\Sigma}_t}$

Joint distribution:

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

Condition on R_{t-1}:

$$\boxed{\begin{aligned} \mathbf{E}[V(x)|R_{t-1} = \mathbf{r}_{t-1}] &= \mathbf{k}_t(x)^\top \boldsymbol{\alpha}_t \\ \mathrm{Cov}[V(x), V(x')|R_{t-1} = \mathbf{r}_{t-1}] &= k(x,x') - \mathbf{k}_t(x)^\top \mathbf{C}_t \mathbf{k}_t(x') \end{aligned}}$$

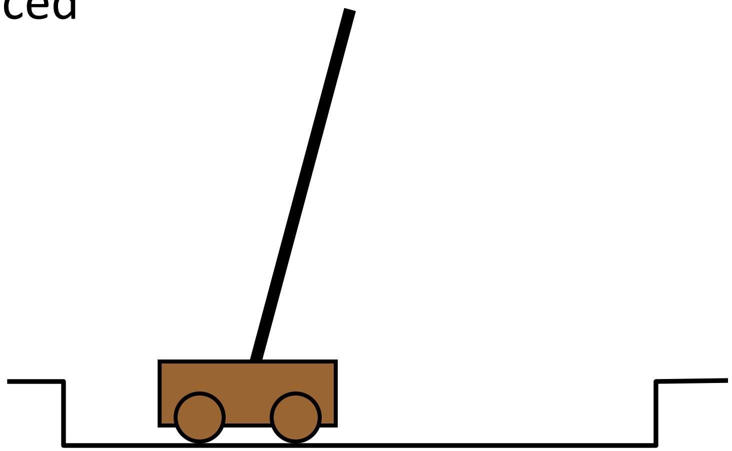
$$\boldsymbol{\alpha}_t = \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \right)^{-1} \mathbf{r}_{t-1}, \quad \mathbf{C}_t = \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \right)^{-1} \mathbf{H}_t$$

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

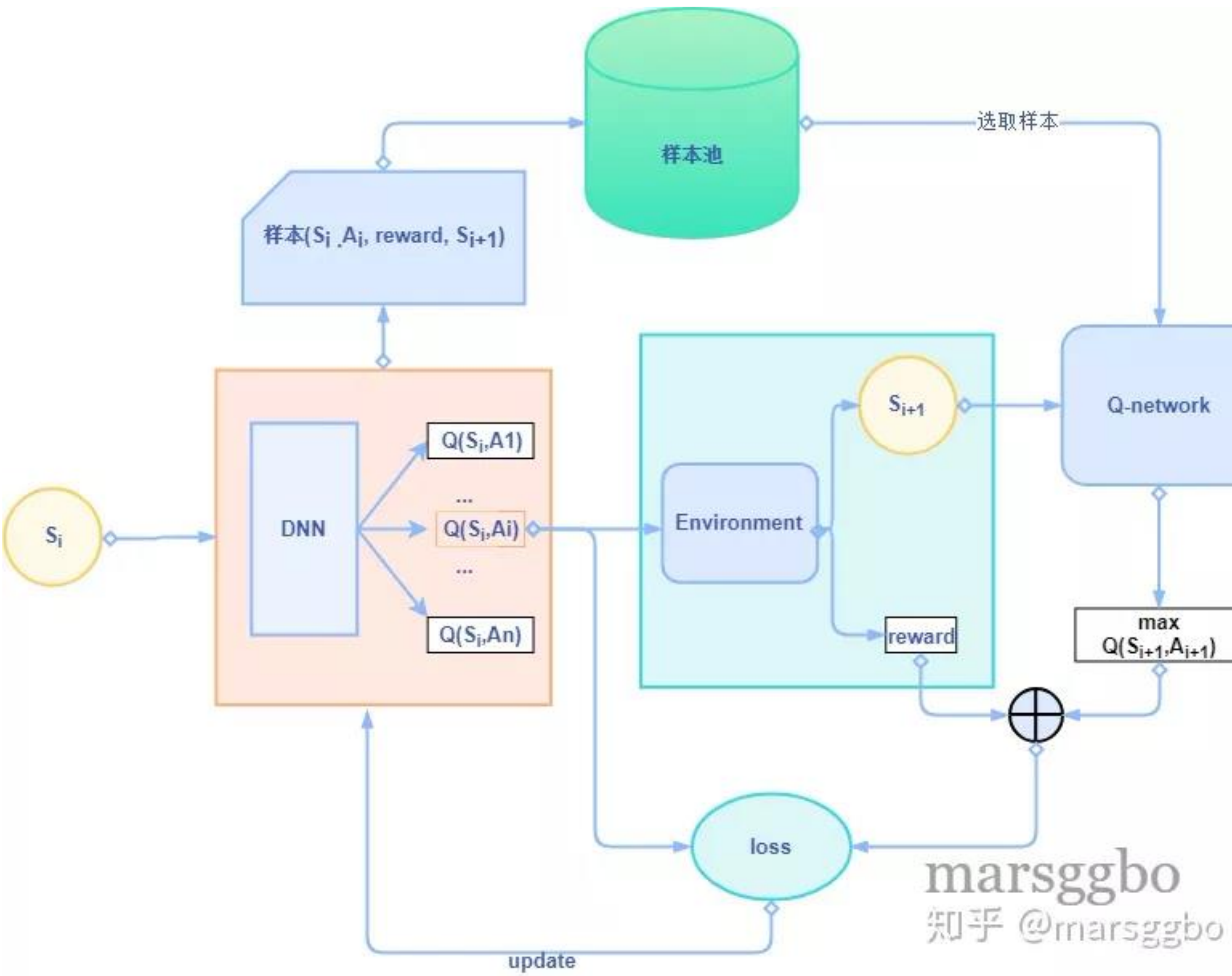
    fewer parameters and updates affect "similar" states as well
- TD update

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

$$\underbrace{V(s_t)}_{\text{f(x)}} \mapsto \underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{y: label}}$$

  - ✓ 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

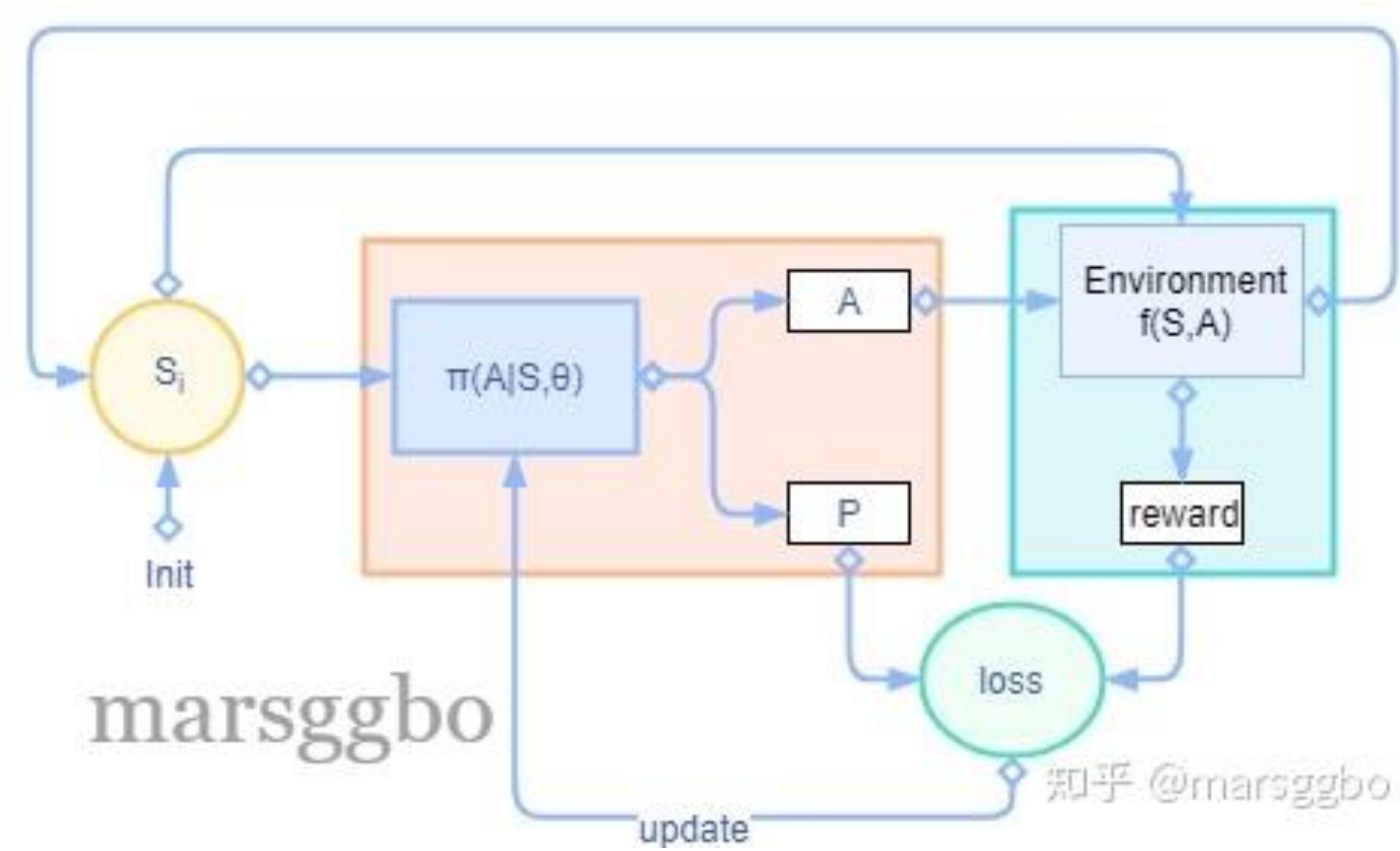
- represent π_t as a parameterized classification function
  - ✓ softmax regression, neural net, …
  - ✓ cost function: $L(\theta) = -\sum \log p(x)\, f(x) = -\sum \log \pi(A|S,\theta)\, f(S,A)$
- update parameters instead of entries in a table
  - ✓ better generalization

    fewer parameters and updates affect "similar" states-actions as well
- Policy gradient for update

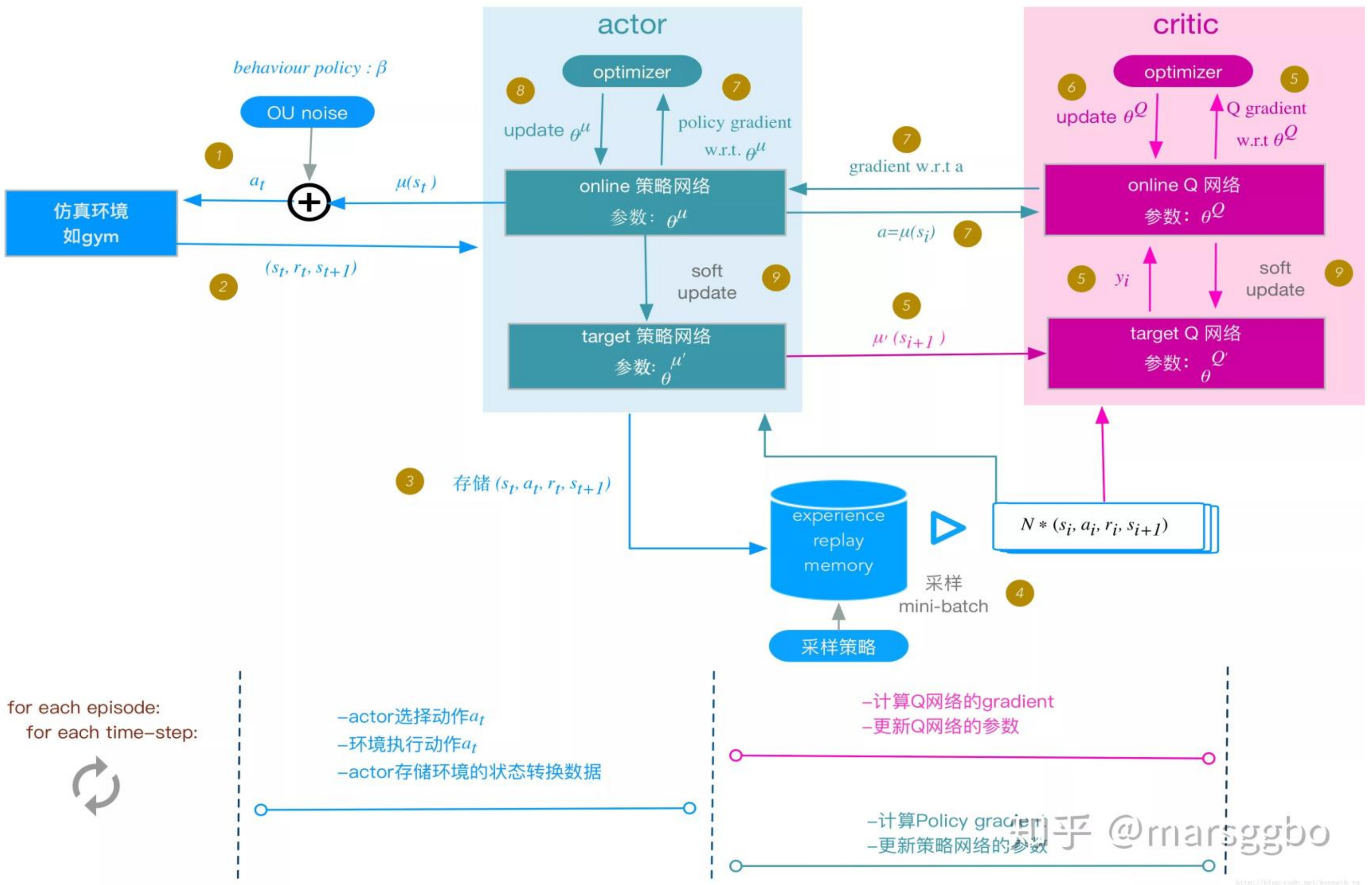
  $$g(\theta) = \nabla_\theta L(\theta) = -\sum \nabla_\theta \log \pi(A|S,\theta)\, f(S,A)$$

  - ✓ treat as one episode for classification
  - ✓ want a method that can learn off-line (update after each episode)
  - ✓ Monte Carlo + NN

Policy Gradient Diagram

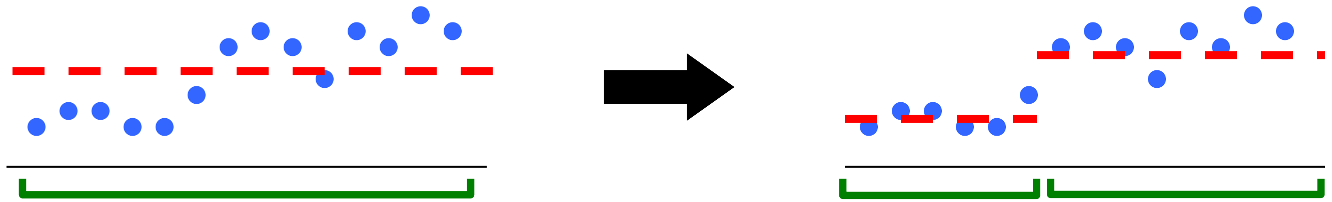


Actor-Critic Diagram

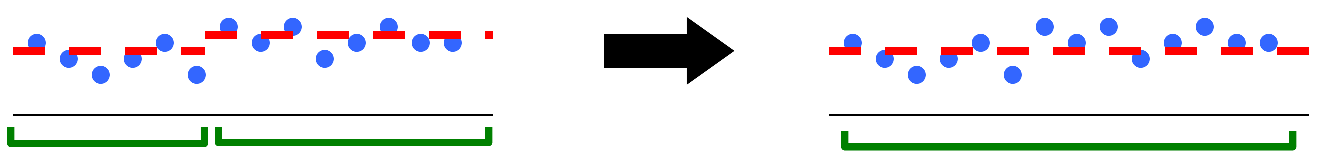


Splitting and Aggregation

- 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

Inverse RL

- **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)

- **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