

Lecture 2: Markov Decision Processes

Bolei Zhou

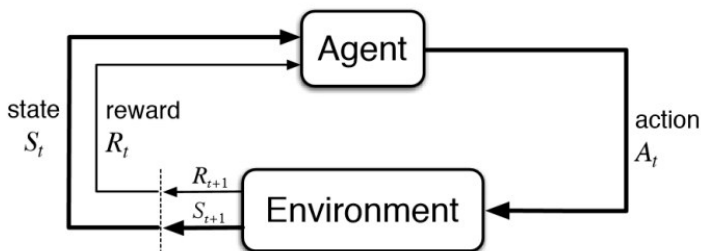
<https://github.com/zhoubolei/introRL>

July 4, 2020

Plan

- ① Last Time
 - ① Key elements of an RL agent: model, value, policy
- ② This Time: Decision Making in MDP
 - ① Markov Chain → Markov Reward Process (MRP) → Markov Decision Processes (MDP) 马尔可夫链 -> 马尔可夫奖励过程 (MRP) -> 马尔可夫决策过程 (MDP)
 - ② Policy evaluation in MDP
 - ③ Control in MDP: policy iteration and value iteration

Markov Decision Process (MDP)



- 1 Markov Decision Process can model a lot of real-world problem. It formally describes the framework of reinforcement learning
- 2 Under MDP, the environment is **fully observable**.
 - 1 Optimal control primarily deals with continuous MDPs
 - 2 Partially observable problems can be converted into MDPs

Define the Markov Models

- Markov Processes
- Markov Reward Processes(MRPs)
- Markov Decision Processes (MDPs)

Markov Property

- ① The history of states: $h_t = \{s_1, s_2, s_3, \dots, s_t\}$
- ② State s_t is Markovian if and only if:

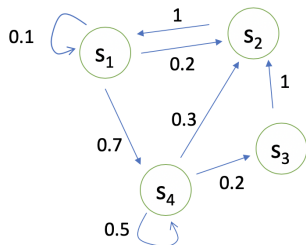
$$p(s_{t+1}|s_t) = p(s_{t+1}|h_t) \quad (1)$$

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|h_t, a_t) \quad (2)$$

- ③ “The future is independent of the past given the present”

下一个状态取决于当前状态，而跟再之前的状态没有关系

Markov Process/Markov Chain

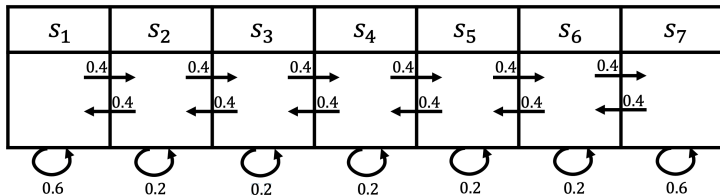


状态转移矩阵

- ① State transition matrix P specifies $p(s_{t+1} = s' | s_t = s)$

$$P = \begin{bmatrix} P(s_1|s_1) & P(s_2|s_1) & \dots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \dots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \dots & P(s_N|s_N) \end{bmatrix}$$

Example of MP



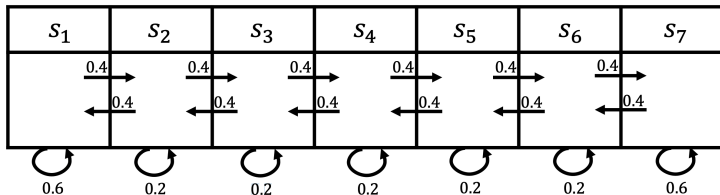
① Sample episodes starting from s_3

- ① s_3, s_4, s_5, s_6, s_6
- ② s_3, s_2, s_3, s_2, s_1
- ③ s_3, s_4, s_4, s_5, s_5

Markov Reward Process (MRP)

- ① Markov Reward Process is a Markov Chain + reward
- ② Definition of Markov Reward Process (MRP)
 - ① S is a (finite) set of states ($s \in S$)
 - ② P is dynamics/transition model that specifies $P(S_{t+1} = s' | s_t = s)$
 - ③ R is a reward function $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$
 - ④ Discount factor $\gamma \in [0, 1]$
- ③ If finite number of states, R can be a vector

Example of MRP



Reward: +5 in s_1 , +10 in s_7 , 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$ If finite number of states, R can be a vector

Return and Value function

- ① Definition of **Horizon** “轨迹”的长度（序列）
 - ① Number of maximum time steps in each episode
 - ② Can be infinite, otherwise called finite Markov (reward) Process

- ② Definition of **Return** 折扣之后的收益只和

- ① Discounted sum of rewards from time step t to horizon

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots + \gamma^{T-t-1} R_T$$

越往后得到的奖励折扣越多（更希望得到立即的奖励）

- ③ Definition of **state value function** $V_t(s)$ for a MRP

- ① Expected return from t in state s

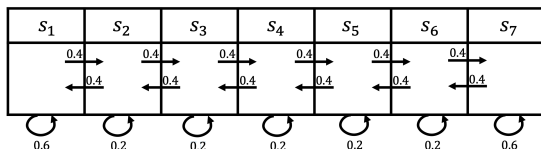
$$\begin{aligned} V_t(s) &= \mathbb{E}[G_t | s_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T | s_t = s] \end{aligned}$$

- ② Present value of future rewards 进入某一个状态之后，将来有多大价值

Why Discount Factor γ

- ① Avoids infinite returns in cyclic Markov processes
 - ② Uncertainty about the future may not be fully represented
 - ③ If the reward is financial, immediate rewards may earn more interest than delayed rewards
 - ④ Animal/human behaviour shows preference for immediate reward
 - ⑤ It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.
 - ① $\gamma = 0$: Only care about the immediate reward
 - ② $\gamma = 1$: Future reward is equal to the immediate reward.
- 类似于深度学习中的超参数

Example of MRP



- ➊ Reward: +5 in s_1 , +10 in s_7 , 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$
- ➋ Sample returns G for a 4-step episodes with $\gamma = 1/2$
 - ➊ return for s_4, s_5, s_6, s_7 : $0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 10 = 1.25$
 - ➋ return for s_4, s_3, s_2, s_1 : $0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 5 = 0.625$
 - ➌ return s_4, s_5, s_6, s_6 : $= 0$
- ➌ How to compute the value function? For example, the value of state s_4 as $V(s_4)$

Computing the Value of a Markov Reward Process

- ① Value function: expected return from starting in state s

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t = s]$$

- ② MRP value function satisfies the following **Bellman equation**:

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future reward}}$$

- ③ Practice: To derive the Bellman equation for $V(s)$

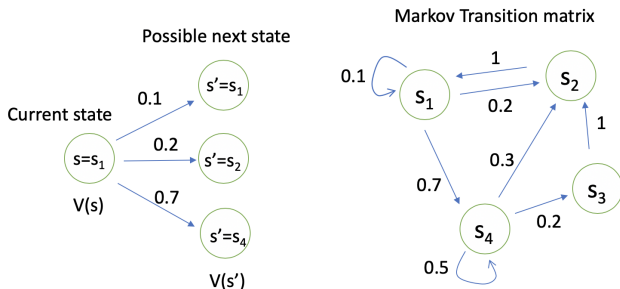
① Hint: $V(s) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots] | s_t = s]$

[马尔科夫决策过程之Bellman Equation（贝尔曼方程） - 知乎](<https://zhuanlan.zhihu.com/p/35261164>)

Understanding Bellman Equation

- ① **Bellman equation** describes the iterative relations of states

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$$



Matrix Form of Bellman Equation for MRP

Therefore, we can express $V(s)$ using the matrix form:

$$\begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_N) \end{bmatrix} = \begin{bmatrix} R(s_1) \\ R(s_2) \\ \vdots \\ R(s_N) \end{bmatrix} + \gamma \begin{bmatrix} P(s_1|s_1) & P(s_2|s_1) & \dots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \dots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \dots & P(s_N|s_N) \end{bmatrix} \begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_N) \end{bmatrix}$$

$$V = R + \gamma PV$$

① Analytic solution for value of MRP: $V = (I - \gamma P)^{-1}R$

① Matrix inverse takes the complexity $O(N^3)$ for N states 求逆的复杂度太高

② Only possible for a small MRPs 适合比较小的MRP

Iterative Algorithm for Computing Value of a MRP

- ① Iterative methods for large MRPs:
 - ① Dynamic Programming
 - ② Monte-Carlo evaluation
 - ③ Temporal-Difference learning

Monte Carlo Algorithm for Computing Value of a MRP

Algorithm 1 Monte Carlo simulation to calculate MRP value function

- 1: $i \leftarrow 0, G_t \leftarrow 0$
 - 2: **while** $i \neq N$ **do**
 - 3: generate an episode, starting from state s and time t
 - 4: Using the generated episode, calculate return $g = \sum_{i=t}^{H-1} \gamma^{i-t} r_i$
 - 5: $G_t \leftarrow G_t + g, i \leftarrow i + 1$
 - 6: **end while**
 - 7: $V_t(s) \leftarrow G_t / N$
-

① For example: to calculate $V(s_4)$ we can generate a lot of trajectories then take the average of the returns: 随机取样, 重复N次, 取平均值

- ① return for $s_4, s_5, s_6, s_7 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 10 = 1.25$
- ② return for $s_4, s_3, s_2, s_1 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 5 = 0.625$
- ③ return $s_4, s_5, s_6 : = 0$
- ④ more trajectories

Iterative Algorithm for Computing Value of a MRP

Algorithm 2 Iterative algorithm to calculate MRP value function

- 1: for all states $s \in S$, $V'(s) \leftarrow 0$, $V(s) \leftarrow \infty$
 - 2: **while** $\|V - V'\| > \epsilon$ **do** 不停地迭代，直到变化不大的时候停止
 - 3: $V \leftarrow V'$
 - 4: For all states $s \in S$, $V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$
 - 5: **end while**
 - 6: return $V'(s)$ for all $s \in S$
-

[Bootstrapping算法_百度百科](<https://baike.baidu.com/item/Bootstrapping%E7%AE%97%E6%B3%95/5017514>)

Markov Decision Process (MDP)

① Markov Decision Process is Markov Reward Process with decisions.

② Definition of MDP

① S is a finite set of states

② A is a finite set of actions MDP比MRP多了Action这个组成

③ P^a is dynamics/transition model for each action

$$P(s_{t+1} = s' | s_t = s, a_t = a)$$

④ R is a reward function $R(s_t = s, a_t = a) = \mathbb{E}[r_t | s_t = s, a_t = a]$

⑤ Discount factor $\gamma \in [0, 1]$ 可以获得的奖励基于当前的状态和当前采取的行为

③ MDP is a tuple: (S, A, P, R, γ)

Policy in MDP

策略函数用于决定当前状态采取什么样的行动

- ① Policy specifies what action to take in each state
- ② Give a state, specify a distribution over actions
- ③ Policy: $\pi(a|s) = P(a_t = a | s_t = s)$
- ④ Policies are stationary (time-independent), $A_t \sim \pi(a|s)$ for any $t > 0$

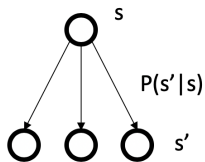
Policy in MDP

当有一个MDP和一个决策函数时，可以将MDP转换为MRP

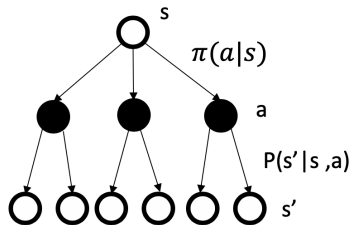
- ① Given an MDP (S, A, P, R, γ) and a policy π
- ② The state sequence S_1, S_2, \dots is a Markov process (S, P^π)
- ③ The state and reward sequence $S_1, R_1, S_2, R_2, \dots$ is a Markov reward process $(S, P^\pi, R^\pi, \gamma)$ where,

$$P^\pi(s'|s) = \sum_{a \in A} \pi(a|s) P(s'|s, a)$$
$$R^\pi(s) = \sum_{a \in A} \pi(a|s) R(s, a)$$

Comparison of MP/MRP and MDP



MRP



MDP

Value function for MDP

- 1 The state-value function $v^\pi(s)$ of an MDP is the expected return starting from state s , and following policy π

$$v^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] \quad (3)$$

- 2 The action-value function $q^\pi(s, a)$ is the expected return starting from state s , taking action a , and then following policy π

$$q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, A_t = a] \quad (4)$$

- 3 We have the relation between $v^\pi(s)$ and $q^\pi(s, a)$

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s, a) \quad (5)$$

Bellman Expectation Equation

- 1 The state-value function can be decomposed into immediate reward plus discounted value of the successor state,

$$v^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma v^{\pi}(s_{t+1}) | s_t = s] \quad (6)$$

- 2 The action-value function can similarly be decomposed

$$q^{\pi}(s, a) = E_{\pi}[R_{t+1} + \gamma q^{\pi}(s_{t+1}, A_{t+1}) | s_t = s, A_t = a] \quad (7)$$

Bellman Expectation Equation for V^π and Q^π

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s, a) \quad (8)$$

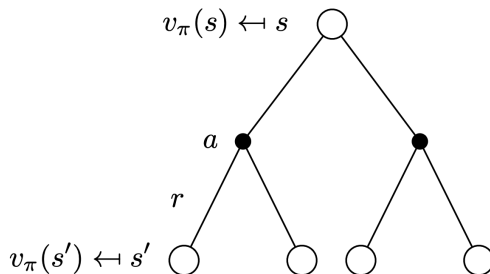
$$q^\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s, a) v^\pi(s') \quad (9)$$

Thus

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^\pi(s')) \quad (10)$$

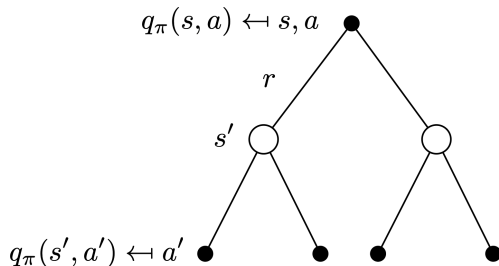
$$q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{a' \in A} \pi(a'|s') q^\pi(s', a') \quad (11)$$

Backup Diagram for V^π



$$v^\pi(s) = \sum_{a \in A} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^\pi(s')) \quad (12)$$

Backup Diagram for Q^π



$$q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{a' \in A} \pi(a'|s') q^\pi(s', a') \quad (13)$$

Policy Evaluation

- 1 Evaluate the value of state given a policy π : compute $v^\pi(s)$
- 2 Also called as (value) prediction

Example: Navigate the boat




Figure: Markov Chain/MRP: Go with river stream




Figure: MDP: Navigate the boat

Example: Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

- ① Two actions: *Left* and *Right*
- ② For all actions, reward: $+5$ in s_1 , $+10$ in s_7 , 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$
- ③ Let's have a deterministic policy $\pi(s) = \text{Left}$ and $\gamma = 0$ for any state s , then what is the value of the policy?
 - ① $V^\pi = [5, 0, 0, 0, 0, 0, 10]$ since $\gamma = 0$

Example: Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

- 1 $R = [5, 0, 0, 0, 0, 0, 10]$
- 2 Practice 1: Deterministic policy $\pi(s) = \text{Left}$ with $\gamma = 0.5$ for any state s , then what are the state values under the policy?
- 3 Practice 2: Stochastic policy $P(\pi(s) = \text{Left}) = 0.5$ and $P(\pi(s) = \text{Right}) = 0.5$ and $\gamma = 0.5$ for any state s , then what are the state values under the policy?

- 4 Iteration t :

$$v_t^\pi(s) = \sum_a P(\pi(s) = a)(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)v_{t-1}^\pi(s'))$$

① Session 1 of Lecture 2 ends here

Decision Making in Markov Decision Process (MDP)

- ① **Prediction** (evaluate a given policy):
 - ① Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π or MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
 - ② Output: value function v^π
- ② **Control** (search the optimal policy):
 - ① Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
 - ② Output: optimal value function v^* and optimal policy π^*
- ③ Prediction and control in MDP can be solved by dynamic programming.

Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- ① Optimal substructure
 - ① Principle of optimality applies
 - ② Optimal solution can be decomposed into subproblems
- ② Overlapping subproblems
 - ① Subproblems recur many times
 - ② Solutions can be cached and reused

Markov decision processes satisfy both properties

- ① Bellman equation gives recursive decomposition
- ② Value function stores and reuses solutions

Policy evaluation on MDP

- ① Objective: Evaluate a given policy π for a MDP
- ② Output: the value function under policy v^π
- ③ Solution: iteration on Bellman expectation backup
- ④ Algorithm: Synchronous backup
 - ① At each iteration $t+1$
update $v_{t+1}(s)$ from $v_t(s')$ for all states $s \in \mathcal{S}$ where s' is a successor state of s

$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_t(s')) \quad (14)$$

- ⑤ Convergence: $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v^\pi$

Policy evaluation: Iteration on Bellman expectation backup

Bellman expectation backup for a particular policy

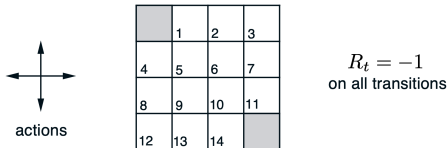
$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_t(s')) \quad (15)$$

Or if in the form of MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}, \gamma \rangle$

$$v_{t+1}(s) = R^\pi(s) + \gamma P^\pi(s'|s) v_t(s') \quad (16)$$

Evaluating a Random Policy in the Small Gridworld

Example 4.1 in the Sutton RL textbook.



- 1 Undiscounted episodic MDP ($\gamma = 1$)
- 2 Nonterminal states $1, \dots, 14$
- 3 Two terminal states (two shaded squares)
- 4 Action leading out of grid leaves state unchanged, $P(7|7, \text{right}) = 1$
- 5 Reward is -1 until the terminal state is reached
- 6 Transition is deterministic given the action, e.g., $P(6|5, \text{right}) = 1$
- 7 Uniform random policy $\pi(l|\cdot) = \pi(r|\cdot) = \pi(u|\cdot) = \pi(d|\cdot) = 0.25$

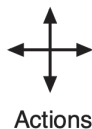
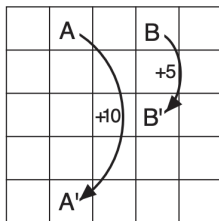
A live demo on policy evaluation

$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s)(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)v^{\pi}(s')) \quad (17)$$

- 1 https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Practice: Gridworld

Textbook Example 3.5: GridWorld



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Optimal Value Function

- 1 The optimal state-value function $v^*(s)$ is the maximum value function over all policies

$$v^*(s) = \max_{\pi} v^{\pi}(s)$$

- 2 The optimal policy

$$\pi^*(s) = \arg \max_{\pi} v^{\pi}(s)$$

- 3 An MDP is “solved” when we know the optimal value
- 4 There exists a unique optimal value function, but could be multiple optimal policies (two actions that have the same optimal value function)

Finding Optimal Policy

- ① An optimal policy can be found by maximizing over $q^*(s, a)$,

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg \max_{a \in A} q^*(s, a) \\ 0, & \text{otherwise} \end{cases}$$

- ② There is always a deterministic optimal policy for any MDP
- ③ If we know $q^*(s, a)$, we immediately have the optimal policy

Policy Search

- ① One option is to enumerate search the best policy
- ② Number of deterministic policies is $|\mathcal{A}|^{|S|}$
- ③ Other approaches such as **policy iteration** and **value iteration** are more efficient

- 1 Compute the optimal policy

$$\pi^*(s) = \arg \max_{\pi} v^{\pi}(s) \quad (18)$$

- 2 Optimal policy for a MDP in an infinite horizon problem (agent acts forever) is
 - 1 Deterministic
 - 2 Stationary (does not depend on time step)
 - 3 Unique? Not necessarily, may have state-actions with identical optimal values

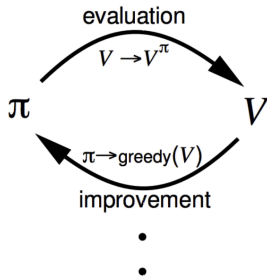
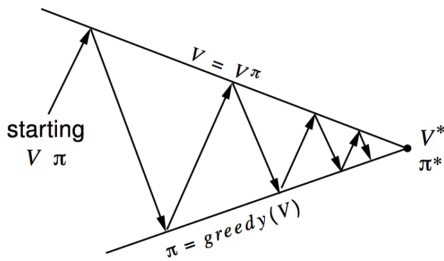
Improve a Policy through Policy Iteration

先通过现有的Policy求出价值，再根据求出的价值使用贪心算法修改策略

① Iterate through the two steps:

- ① **Evaluate** the policy π (computing v given current π)
- ② **Improve** the policy by acting greedily with respect to v^π

$$\pi' = \text{greedy}(v^\pi) \quad (19)$$



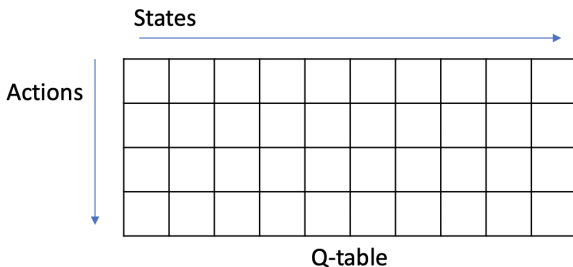
Policy Improvement

- 1 Compute the state-action value of a policy π :

$$q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v^{\pi_i}(s') \quad (20)$$

- 2 Compute new policy π_{i+1} for all $s \in \mathcal{S}$ following

$$\pi_{i+1}(s) = \arg \max_a q^{\pi_i}(s, a) \quad (21)$$



Monotonic Improvement in Policy

Policy的改进过程是单调递增的

- 1 Consider a deterministic policy $a = \pi(s)$
- 2 We improve the policy through

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

- 3 This improves the value from any state s over one step,

$$q^\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q^\pi(s, a) \geq q^\pi(s, \pi(s)) = v^\pi(s)$$

- 4 It therefore improves the value function, $v_{\pi'}(s) \geq v^\pi(s)$

$$\begin{aligned} v^\pi(s) &\leq q^\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v^\pi(S_{t+1} | S_t = s)] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q^\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q^\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s] = v_{\pi'}(s) \end{aligned}$$

Monotonic Improvement in Policy

- 1 If improvements stop,

$$q^\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q^\pi(s, a) = q^\pi(s, \pi(s)) = v^\pi(s)$$

- 2 Thus the Bellman optimality equation has been satisfied

$$v^\pi(s) = \max_{a \in \mathcal{A}} q^\pi(s, a)$$

- 3 Therefore $v^\pi(s) = v^*(s)$ for all $s \in \mathcal{S}$, so π is an optimal policy

Bellman Optimality Equation

- 1 The optimal value functions are reached by the Bellman optimality equations:

$$v^*(s) = \max_a q^*(s, a)$$

$$q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$$

thus

$$v^*(s) = \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$$

$$q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} q^*(s', a')$$

Value Iteration by turning the Bellman Optimality Equation as update rule

- 1 If we know the solution to subproblem $v^*(s')$, which is optimal.
- 2 Then the solution for the optimal $v^*(s)$ can be found by iteration over the following Bellman Optimality backup rule,

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v(s') \right)$$

- 3 The idea of value iteration is to apply these updates iteratively

Algorithm of Value Iteration

- ① Objective: find the optimal policy π
- ② Solution: iteration on the Bellman optimality backup
- ③ Value Iteration algorithm:
 - ① initialize $k = 1$ and $v_0(s) = 0$ for all states s
 - ② For $k = 1 : H$
 - ① for each state s

$$q_{k+1}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v_k(s') \quad (22)$$

$$v_{k+1}(s) = \max_a q_{k+1}(s, a) \quad (23)$$

- ② $k \leftarrow k + 1$
- ③ To retrieve the optimal policy after the value iteration:

$$\pi(s) = \arg \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v_{k+1}(s') \quad (24)$$

Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

训练结束后离g越远value越小

After the optimal values are reached, we run policy extraction to retrieve the optimal policy.

Difference between Policy Iteration and Value Iteration

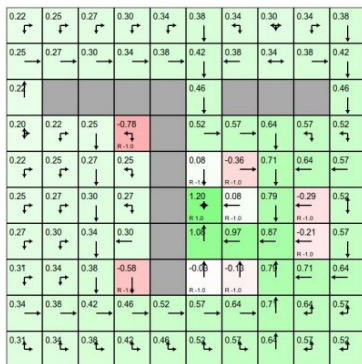
- 1 Policy iteration includes: **policy evaluation + policy improvement**, and the two are repeated iteratively until policy converges.
- 2 Value iteration includes: **finding optimal value function + one policy extraction**. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).
- 3 Finding optimal value function can also be seen as a combination of policy improvement (due to max) and truncated policy evaluation (the reassignment of $v(s)$ after just one sweep of all states regardless of convergence).

Summary for Prediction and Control in MDP

Table: Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Demo of policy iteration and value iteration



- 1 Policy iteration: Iteration of policy evaluation and policy improvement(update)
- 2 Value iteration
- 3 https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Policy iteration and value iteration on FrozenLake

① <https://github.com/cuhkrlcourse/RLEXample/tree/master/MDP>

End

- ① Summary: MDP, policy evaluation, policy iteration, and value iteration
- ② Optional Homework 1 is available at <https://github.com/cuhkrlcourse/ierg6130-assignment>
- ③ Next Week: Model-free methods
- ④ Reading: Textbook Chapter 5 and 6

- ① Additional slides on improving the dynamic programming in MDP

Asynchronous Dynamic Programming

- ① A major drawback to the DP methods is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set.
- ② If the state set is very large, for example, the game of backgammon has over 10^{20} states. Thousands of years to be taken to finish one sweep.
- ③ Asynchronous DP algorithms are in-place iterative DP that are not organized in terms of systematic sweeps of the state set
- ④ The values of some states may be updated several times before the values of others are updated once.

Improving Dynamic Programming

Synchronous dynamic programming is usually slow. Three simple ideas to extend DP for asynchronous dynamic programming:

- 1 In-place dynamic programming
- 2 Prioritized sweeping
- 3 Real-time dynamic programming

In-Places Dynamic Programming

- 1 Synchronous value iteration stores two copies of value function:

for all s in \mathcal{S}

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_{old}(s') \right)$$

$$v_{old} \leftarrow v_{new}$$

- 2 In-place value iteration only stores one copy of value function:

for all s in \mathcal{S}

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v(s') \right)$$

Prioritized Sweeping

- 1 Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v(s') \right) - v(s) \right|$$

- 2 Backup the state with the largest remaining Bellman error
- 3 Update Bellman error of affected states after each backup
- 4 Can be implemented efficiently by maintaining a priority queue

Real-Time Dynamic Programming

- ① To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP
- ② The agent's experience can be used to determine the states to which the DP algorithm applies its updates
- ③ We can apply updates to states as the agent visits them. So focus on the parts of the state set that are most relevant to the agent
- ④ After each time-step S_t, A_t , backup the state S_t ,

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left(R(S_t, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|S_t, a) v(s') \right)$$

Sample Backups

- ① The key design for RL algorithms such as Q-learning and SARSA in next lectures
- ② Using sample rewards and sample transition pairs $\langle S, A, R, S' \rangle$, rather than the reward function \mathcal{R} and transition dynamics \mathcal{P}
- ③ Benefits:
 - ① Model-free: no advance knowledge of MDP required
 - ② Break the curse of dimensionality through sampling
 - ③ Cost of backup is constant, independent of $n = |\mathcal{S}|$

Approximate Dynamic Programming

- ① Using a function approximator $\hat{v}(s, \mathbf{w})$
- ② Fitted value iteration repeats at each iteration k ,
 - ① Sample state s from the state cache $\tilde{\mathcal{S}}$

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \hat{v}(s', \mathbf{w}_k) \right)$$

- ② Train next value function $\hat{v}(s', \mathbf{w}_{k+1})$ using targets $\langle s, \tilde{v}_k(s) \rangle$.
- ③ Key idea behind the Deep Q-Learning