强化学习

Part 1

强化学习背景

定义

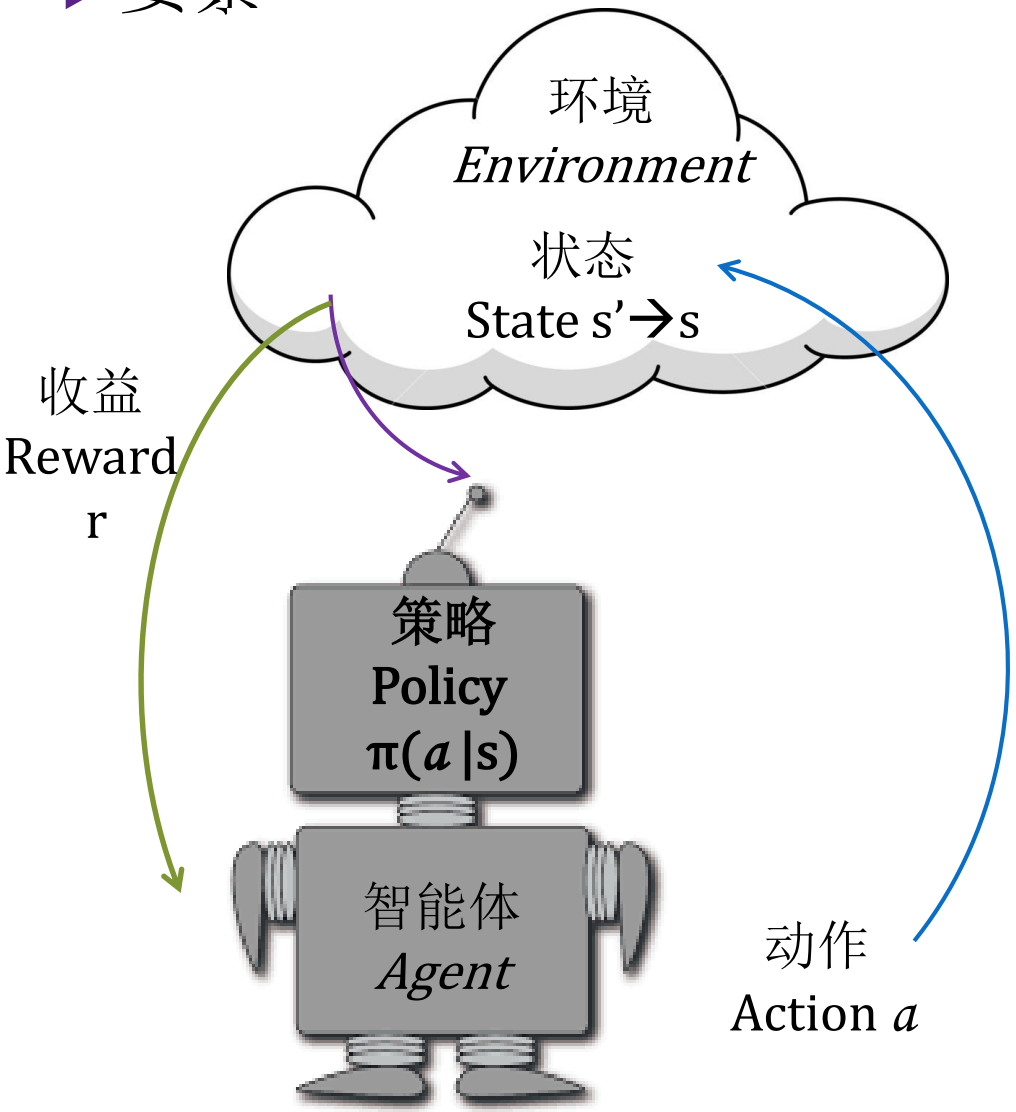
强化学习适用于序贯决策任务。这类任务的特点是：需要连续不断地做出决策，才能实现最终目标。强化学习的目标是尝试发现怎样的策略会产生最丰富的策略，即学习“如何把当前的情景映射诚动作才能使得数值化的累计收益最大化”。

特点

强化学习的设定中，动作往往影响的不仅仅是即时的收益，也会影响下一个情境，从而影响到随后的收益。
强化学习也通常允许学习模型通过尝试去发现哪些动作会产生丰富的收益。
所以，强化学习有着两个显著的特征；**延迟收益**与**允许试错**。

强化学习定义

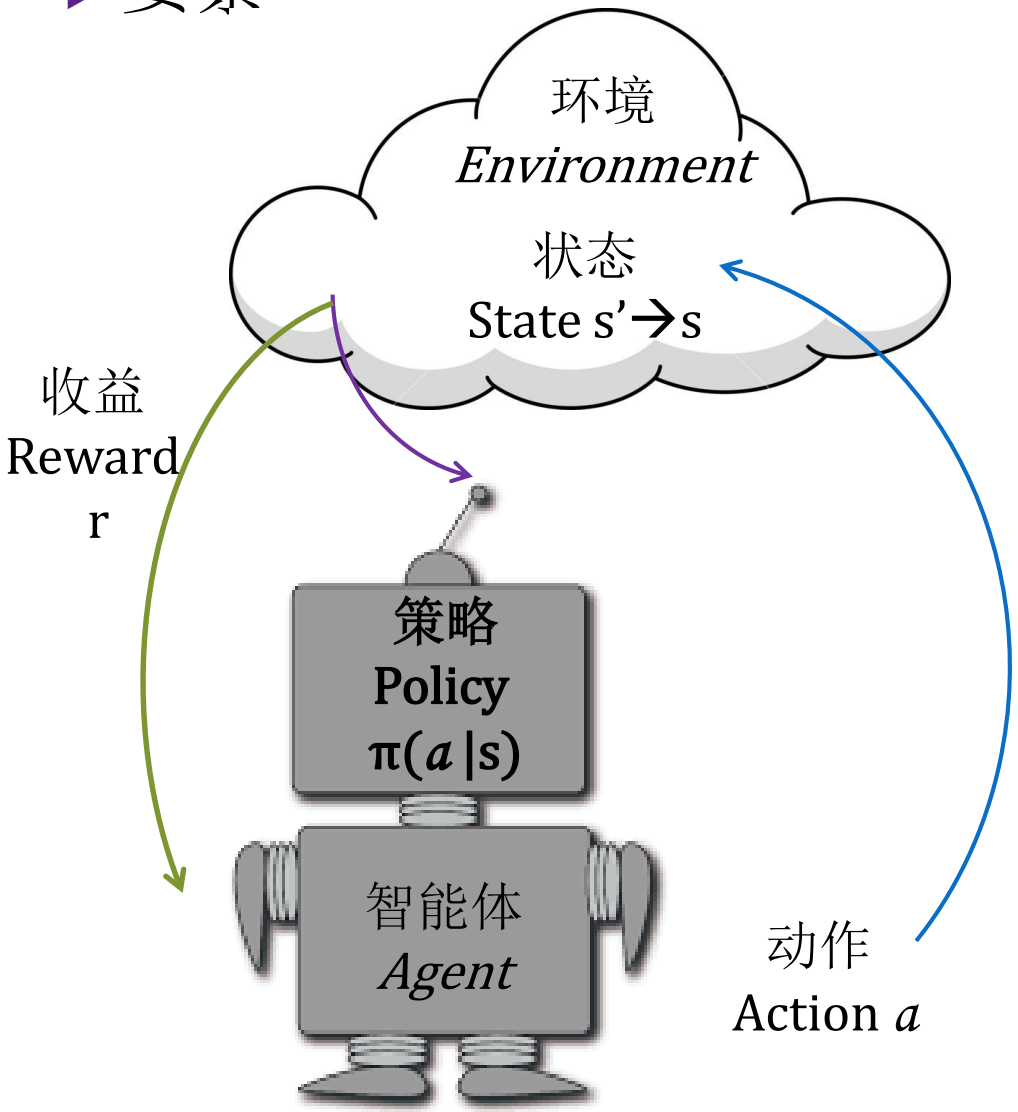
▶ 要素



★ 强化学习的设定中，智能体和（不确定的）环境是能够**交互**的。
这种交互体现在：智能体可以采取动作从而使得环境的状态发生改变。

强化学习定义

▶ 要素



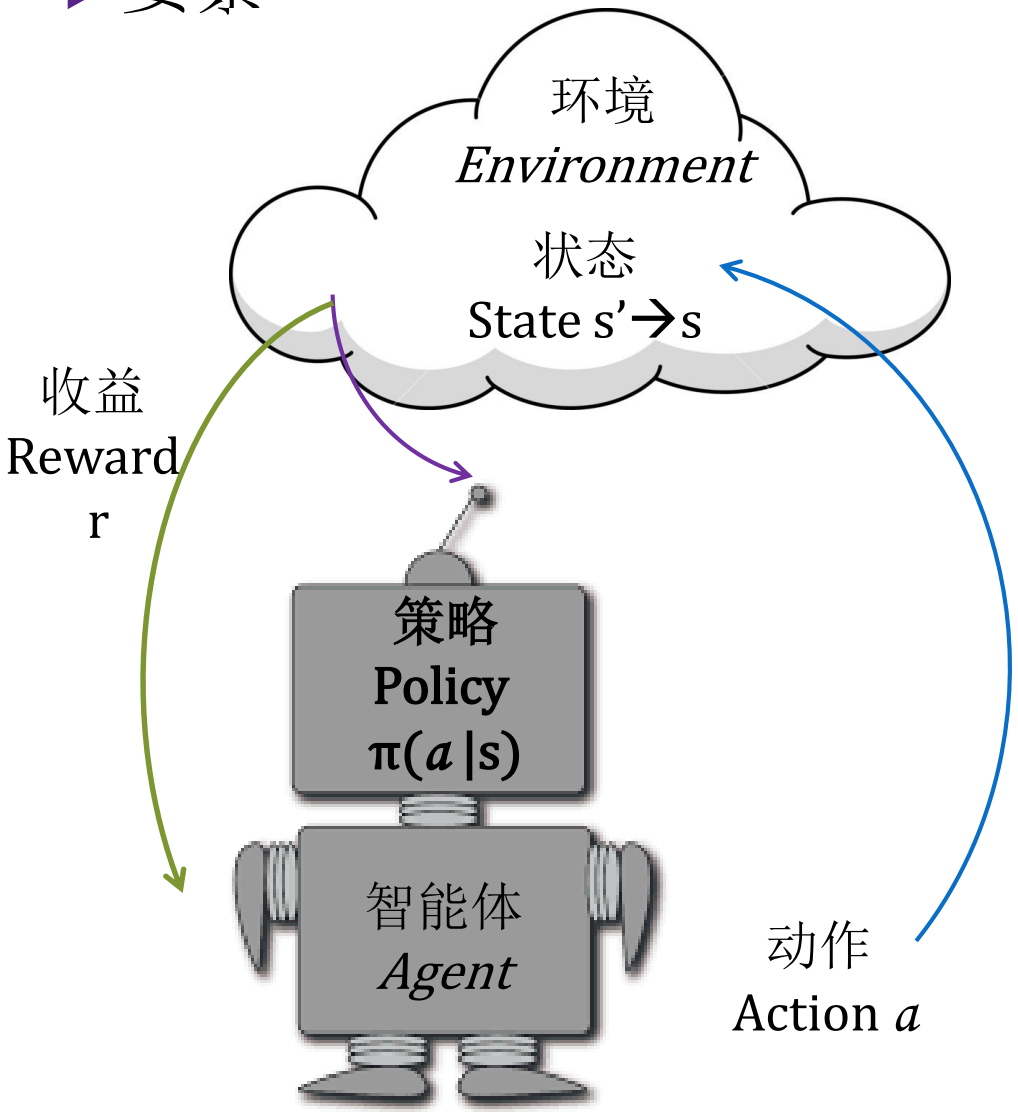
★ 为了使得智能体是目标导向的。环境状态的改变将对应一个收益信号。**收益 Reward** 信号反映了在短时间内什么状态是好的。

★ 然而，收益信号仅仅体现了在短时间内什么动作/状态是好的。由于强化学习的设定中，有延迟收益的特点，我们引入**回报 Gain** 来对应将来累积的总收益，以体现了从长远的角度来看什么状态是好的。

价值 Value对应了智能体从某个状态开始，将来累积的总收益（即回报）的期望。

强化学习定义

▶ 要素

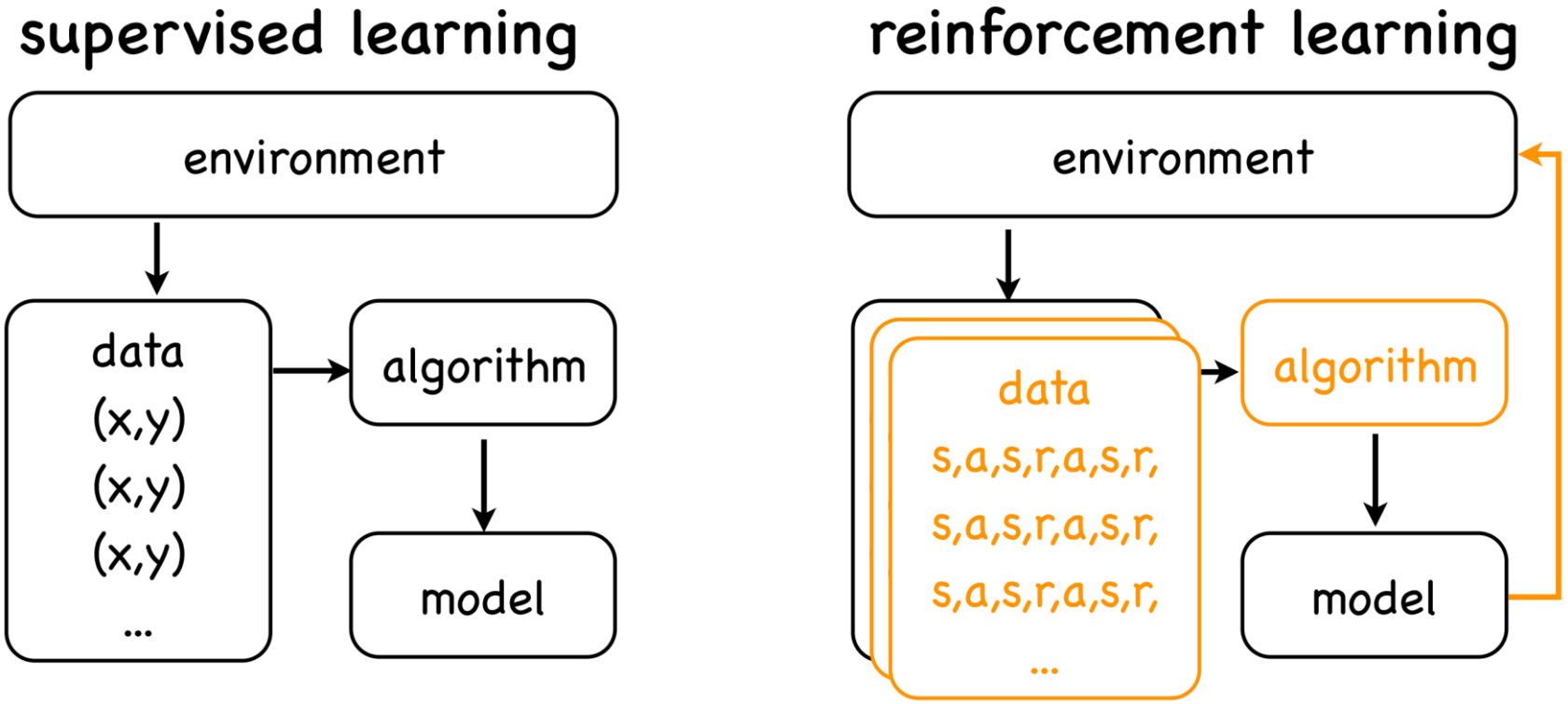


★ 智能体的学习目标对应了**策略 Policy**，策略定义了智能体的行为方式。简单地说，它是一个环境状态到动作的映射函数。且这种映射函数可以是随机函数，即在某种状态下以某种概率分布来选择动作。

$$\pi(a|s)$$

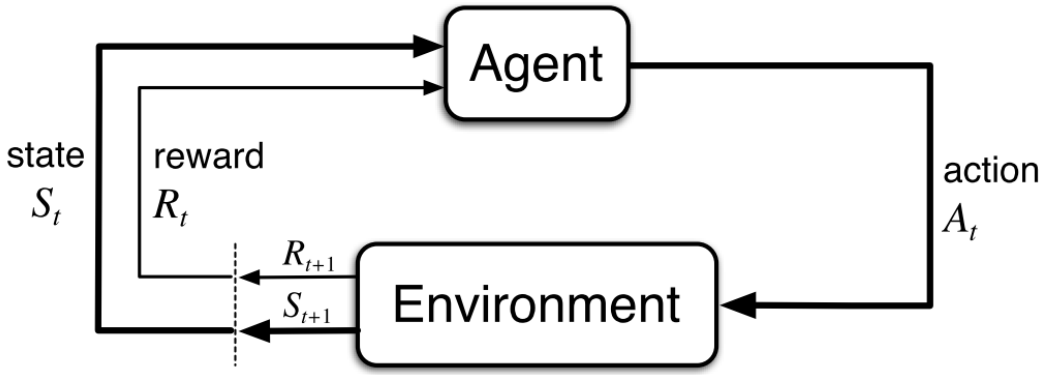
强化学习框架

- 强化学习 vs 监督学习



基于马尔科夫决策过程

- 有限马尔科夫决策过程 *finite MDP*



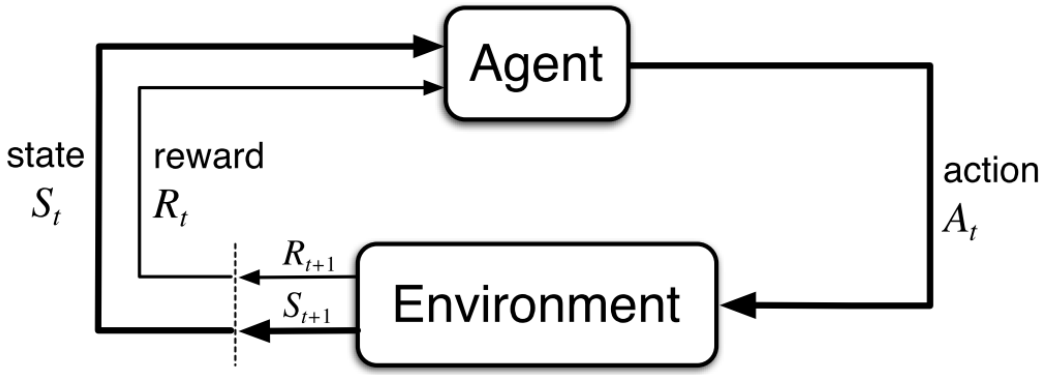
交互轨迹：$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

★ 状态、动作、收益集合 $(\mathcal{S}, \mathcal{A}, \mathcal{R})$都只有有限个元素。

★ S_t 和 R_t 的每个值出现的概率只取决于前一个状态和动作：S_{t-1}和 A_{t-1}, 而与更早之前的状态和动作完全无关。

基于马尔科夫决策过程

- 环境的刻画

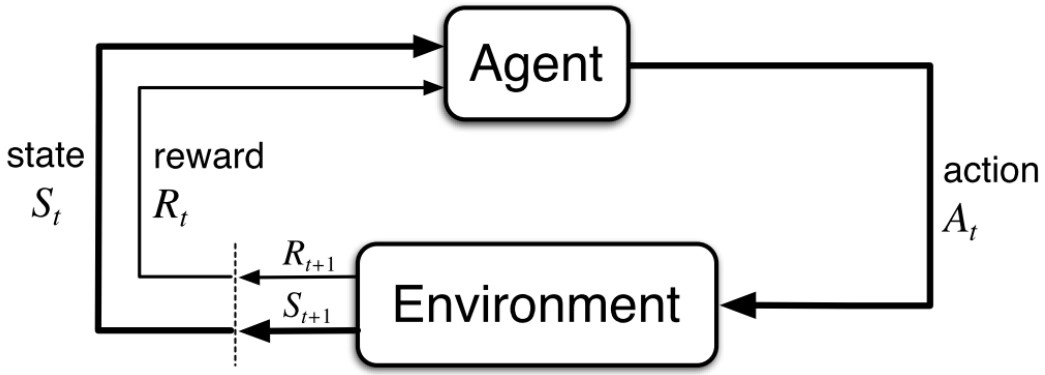


动态函数 $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ 定义了MDP的动态特性：

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \quad \text{for all } s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$$

基于马尔科夫决策过程

- 收益的刻画



“状态-动作”二元组的期望收益 $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s,a) \doteq \mathbb{E}[R_t \mid S_{t-1}=s, A_{t-1}=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r \mid s, a)$$

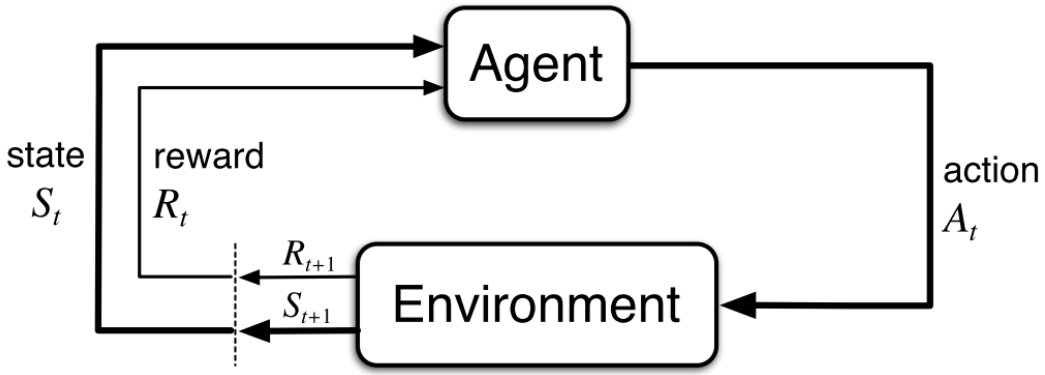
“状态-动作-后续状态”三元组的期望收益 $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r(s,a,s') \doteq \mathbb{E}[R_t \mid S_{t-1}=s, A_{t-1}=a, S_t=s'] = \sum_{r\in\mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

$$p(s' \mid s, a) \doteq \Pr\{S_t=s' \mid S_{t-1}=s, A_{t-1}=a\} = \sum_{r\in\mathcal{R}} p(s', r \mid s, a)$$

基于马尔科夫决策过程

- 收益的刻画



“状态-动作”二元组的期望收益 $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s,a) \doteq \mathbb{E}[R_t \mid S_{t-1}=s, A_{t-1}=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r \mid s, a)$$

“状态-动作-后续状态”三元组的期望收益 $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r(s,a,s') \doteq \mathbb{E}[R_t \mid S_{t-1}=s, A_{t-1}=a, S_t=s'] = \sum_{r\in\mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

$$p(s' \mid s, a) \doteq \Pr\{S_t=s' \mid S_{t-1}=s, A_{t-1}=a\} = \sum_{r\in\mathcal{R}} p(s', r \mid s, a)$$

基于马尔科夫决策过程

▶ 回报的刻画

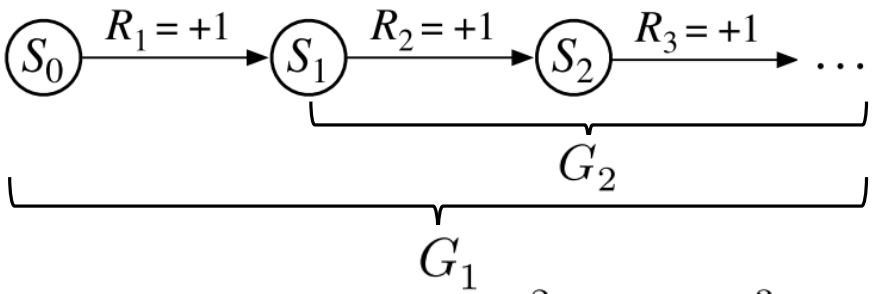
把时刻t后接受到的收益序列表示为$R_{t+1}, R_{t+2}, R_{t+3}, \ldots$，
若任务是有限步的/分幕式的，则定义

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

若任务是持续式的，则定义

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$, $0 \le \gamma \le 1$称为折扣率 *discount rate*

统一为：$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $T = \infty$（持续式） $\gamma = 1$（分幕式）



$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

邻接时刻的回报可以用一个递归的方式联系起来

基于马尔科夫决策过程

▶ 价值的刻画（价值函数）

价值函数评估了当前智能体在给定状态（或给定状态与动作）下采取某种特定策略能够获得多少未来逾期的收益。价值函数即为“回报的期望值”。

■ 状态价值函数 *State Value Functions*

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}$$

表示了若我们从状态s开始，用π作为策略能够获得的回报的期望。

■ 状态-动作价值函数 *State-action Value Functions*

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

表示了若我们从状态s开始，执行了动作a后，再用π作为策略能够获得的回报的期望。

■ 关系

$$v_\pi(s) = \sum_a \pi(a|s)\, q_\pi(s,a) \qquad q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

基于马尔科夫决策过程

- 最优策略的刻画

当且仅当对于所有的$s \in \mathcal{S}$，有 $v_\pi(s) \geq v_{\pi'}(s)$，则有 $\pi \geq \pi'$

最优策略 π_* 为所有策略中最优/大的策略，对应的最优价值函数为

$$v_*(s) = \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S}$$

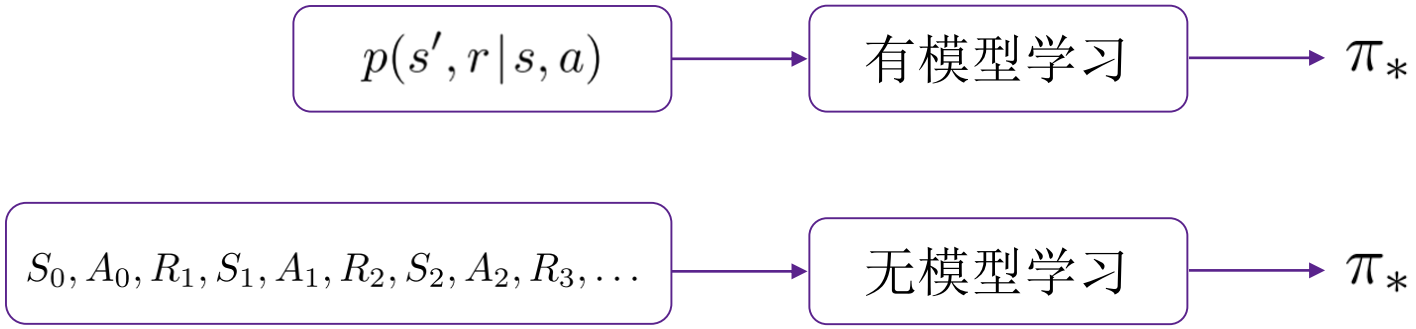
基于马尔科夫决策过程

- 最优策略的刻画

当且仅当对于所有的$s \in \mathcal{S}$，有 $v_\pi(s) \geq v_{\pi'}(s)$，则有 $\pi \geq \pi'$

最优策略 π_* 为所有策略中最优/大的策略，对应的最优价值函数为

$$v_*(s) = \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S}$$



有模型学习

- 已知

$\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$

- 已知

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, $a \in \mathcal{A}(s)$

- 求解

π_*

策略迭代

从一个初始策略(通常是随机策略)出发，不断改进策略,直到策略收敛、不再改变为止。这样的做法称为“策略迭代” *policy iteration*

$$\pi_0 \longrightarrow \pi_1 \longrightarrow \pi_2 \longrightarrow \cdots \longrightarrow \pi_*$$

$$\pi \to \pi'$$

$$v_{\pi'}(s) \geq v_\pi(s) \text{ for all } s \in \mathcal{S}$$

策略迭代

每一步策略迭代分成两步，先进行**策略评估** *policy evaluation*，然后进行**策略改进** *policy improvement*，这是由于策略π如何改进为π'是需要通过v_π函数来获得的

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

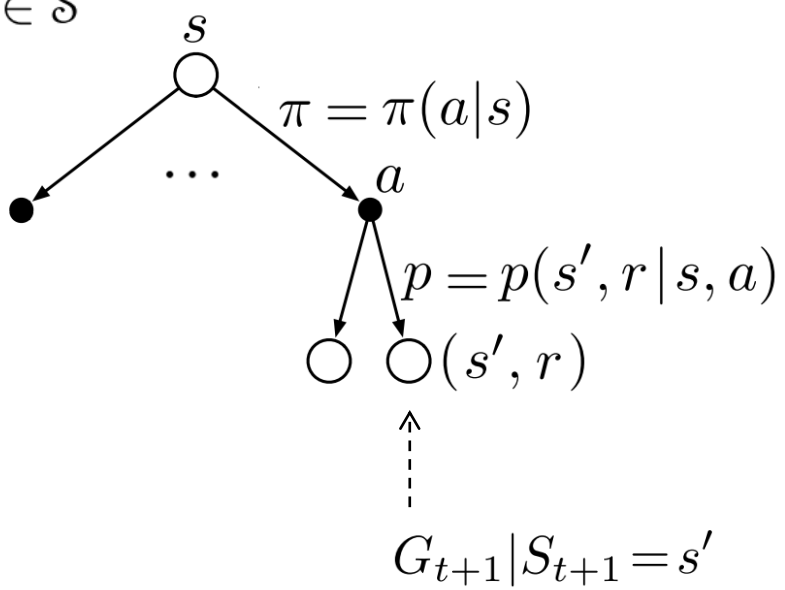
$$\pi \underset{?}{\rightarrow} v_\pi \underset{?}{\rightarrow} \pi' \quad \Longrightarrow \quad v_{\pi'}(s) \geq v_\pi(s) \ \text{ for all } s \in \mathcal{S}$$

有模型学习

- 策略评估
  - 贝尔曼等式 Bellman equation

$$v_\pi(s') \xrightarrow[\pi(a|s)]{p(s',r|s,a)} v_\pi(s)$$

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \quad \text{（此处用到了前面介绍的 } G_t \text{ 的递归表达）} \\
&= \sum_a \sum_{s'} \sum_r \Pr\{A=a,\ S_{t+1}=s', R_{t+1}=r\} \left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_\pi(s')\right], \quad \text{for all } s \in \mathcal{S}
\end{aligned}$$



策略迭代

▶ 策略评估

■ 动态规划 *Dynamic Programming*：

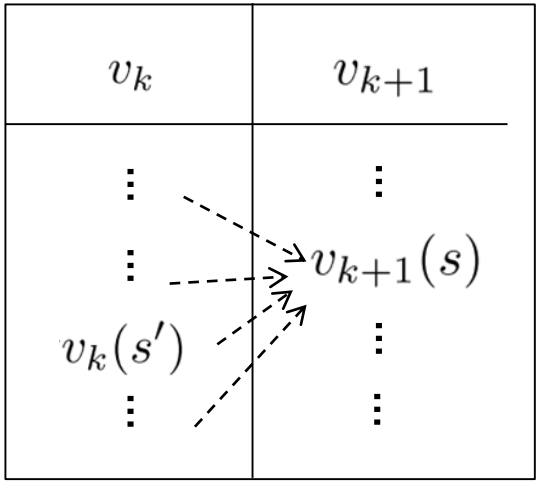
回顾价值函数递归的定义：

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big] \text{ for all } s \in \mathcal{S}:$$

这提示了我们应当用动态规划方法来计算它：

从某个初始函数 v_0 出发，迭代式地得到到序列 $v_0, v_1, v_2, \cdots$：

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big]$$



策略迭代

▶ 策略评估

■ 动态规划 *Dynamic Programming*：

当 $k \to \infty$ 时，序列 $\{v_k\}$ 将收敛于 v_π（迭代评估方法只能在极限意义下收敛）

实际应用的时候，通常设定一个收敛条件，例如设定一个$\theta > 0$来控制评估的精准度：

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:

  $\Delta \leftarrow 0$

  Loop for each $s \in \mathcal{S}$:

    $v \leftarrow V(s)$

    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

收敛性的证明请参考：

RL Course by David Silver - Lecture 3: Planning by Dynamic Programming (from page 35)

策略迭代

- 策略改进*Policy Improvement*

Q: $\pi \to v_\pi \to \pi' \Longrightarrow v_{\pi'}(s) \geq v_\pi(s)$

?

A: $\pi' = \text{greedy}(v_\pi)$

$$\text{for all } s \in \mathcal{S}: \quad \pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, q_\pi(s, a)$$

策略迭代

- 策略改进*Policy Improvement*

Q: why $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_\pi(s,a) \longrightarrow v_{\pi'}(s) \geq v_\pi(s)$?

A:
$$
\begin{aligned}
v_\pi(s) &= \sum_a \pi(a|s)\ q_\pi(s,a) \\
&\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s)
\end{aligned}
$$

策略迭代

- 策略评估 + 策略改进

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

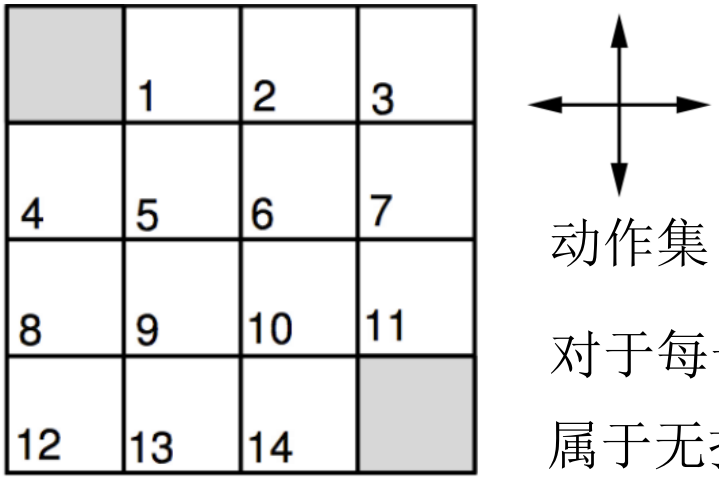
2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

策略迭代

▶ 示例

环境设定：



动作集

对于每一次的转移，有 $R_t = -1$

属于无折扣的分幕式任务，当走到了阴影处任务完成

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

$$p(6, -1|5, \texttt{right}) = 1,\ p(7, -1|7, \texttt{right}) = 1$$

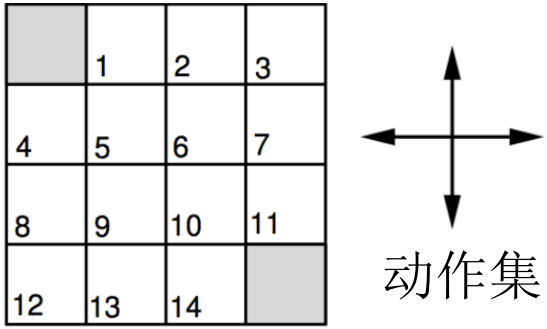
$$p(10, r|5, \texttt{right}) = 0 \text{ for all } r \in \mathcal{R}$$

策略迭代

- 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略

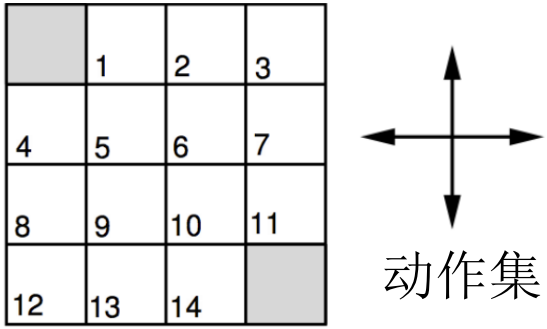


策略迭代

- 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略



$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

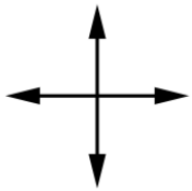
策略迭代

▶ 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略

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



动作集

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$$

从k=1到k=2时，状态1处的值函数：

V2(1) = 0.25*(-1-1.0) + 0.25*(-1-1.0) + 0.25*(-1-1.0) + 0.25*(-1-0.0)=-1.75

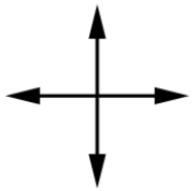
策略迭代

▶ 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略

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



动作集

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

v_{π_0}

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

π_1

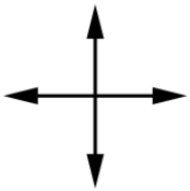
策略迭代

▶ 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略

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



动作集

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

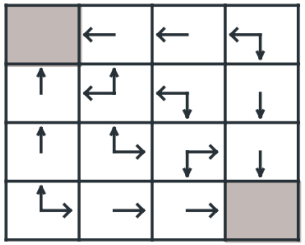
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

v_{π_0}

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

π_1

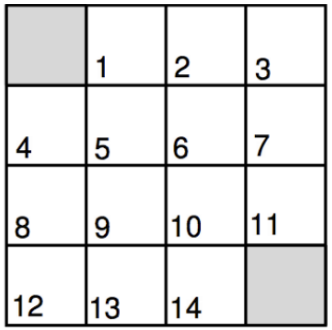


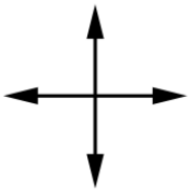
思考

▶ 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略

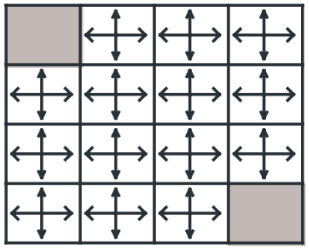




动作集

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

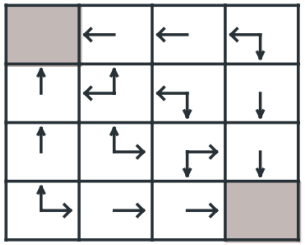
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

v_{π_0}

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

π_1

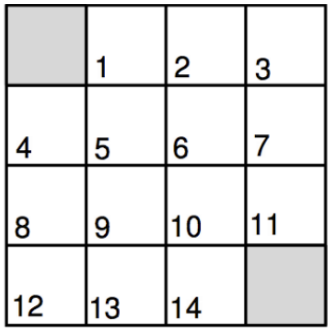


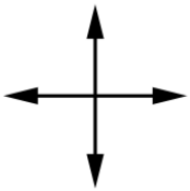
思考

▶ 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略

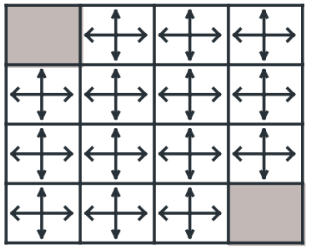




动作集

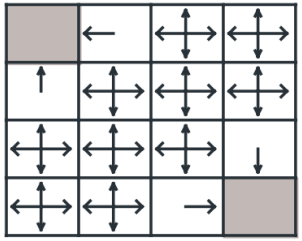
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

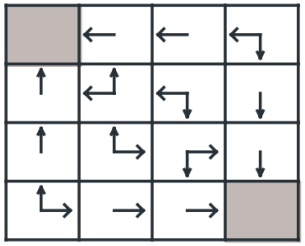
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

v_{π_0}

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

π_1

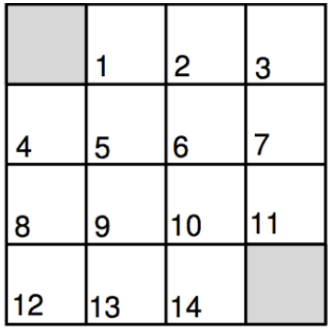


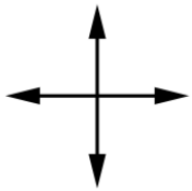
思考

▶ 示例

$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1$

π_0: 随机策略　　π_1: 贪婪策略

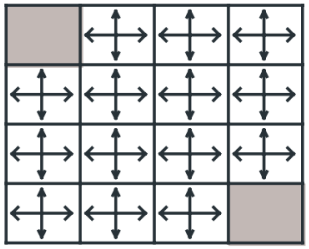




动作集

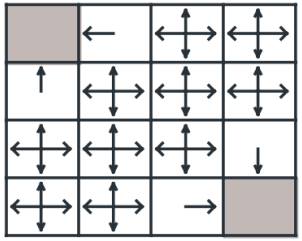
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



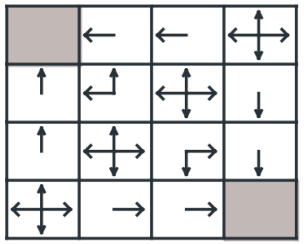
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



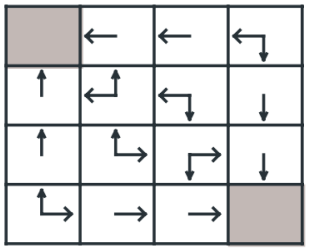
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



$k = 3$

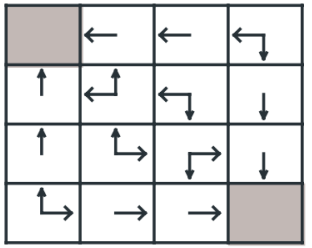
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

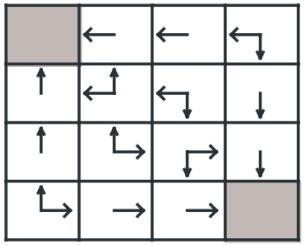
v_{π_0}



π_1

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

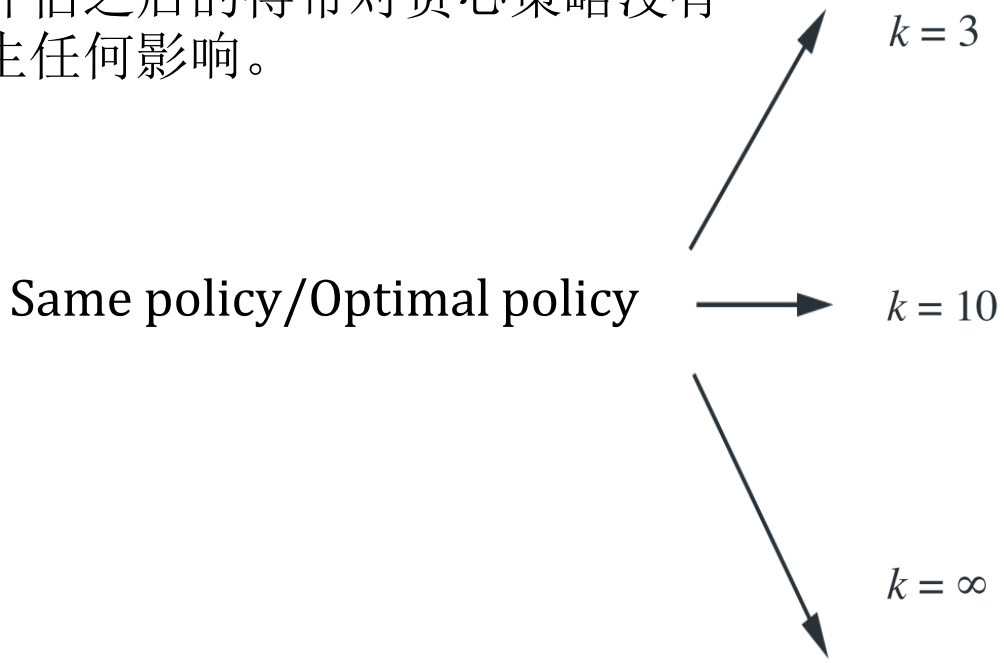


价值迭代

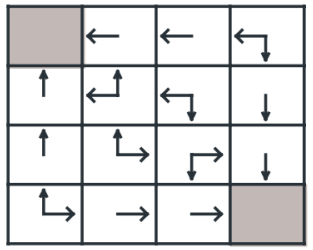
▶ 启发

策略迭代算法每一次迭代都设计了策略评估，而策略评估是一个需要遍历状态集合的迭代过程。并且若策略评估是迭代的方式进行的，那么收敛到　理论上只有在迭代极限次才会成立。需要每次完全等到策略评估过程完全收敛吗？是否可以提早实施策略改进？

★ 在网格行走的例子中，前三轮策略评估之后的得带对贪心策略没有产生任何影响。

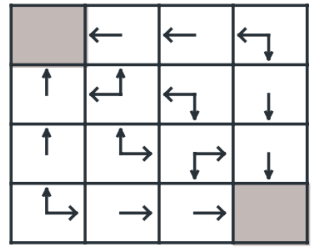


0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



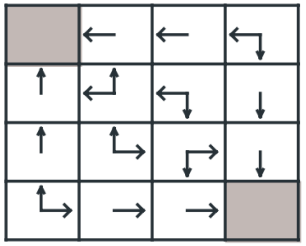
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

v_{π_0}



π_1

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



价值迭代

算法

采取一种特殊的方法阶段策略迭代中的策略评估过程：在一次遍历（即对每个状态进行了一次更新）后即可停止策略评估，并得到改进策略。此时截断的策略评估和策略改进结合起来可以写成一个更简单的更新公式：

$$
\begin{aligned}
v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

价值迭代

- 算法

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

无模型

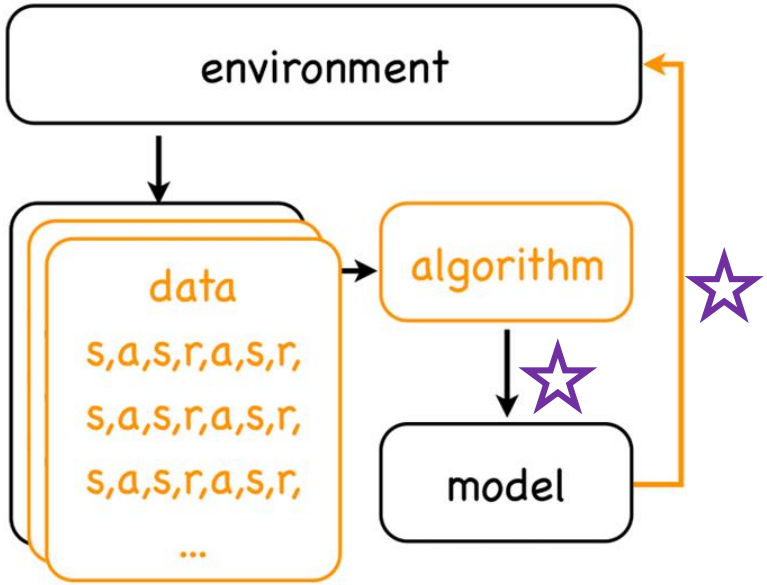
- 已知（可选）

$\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$

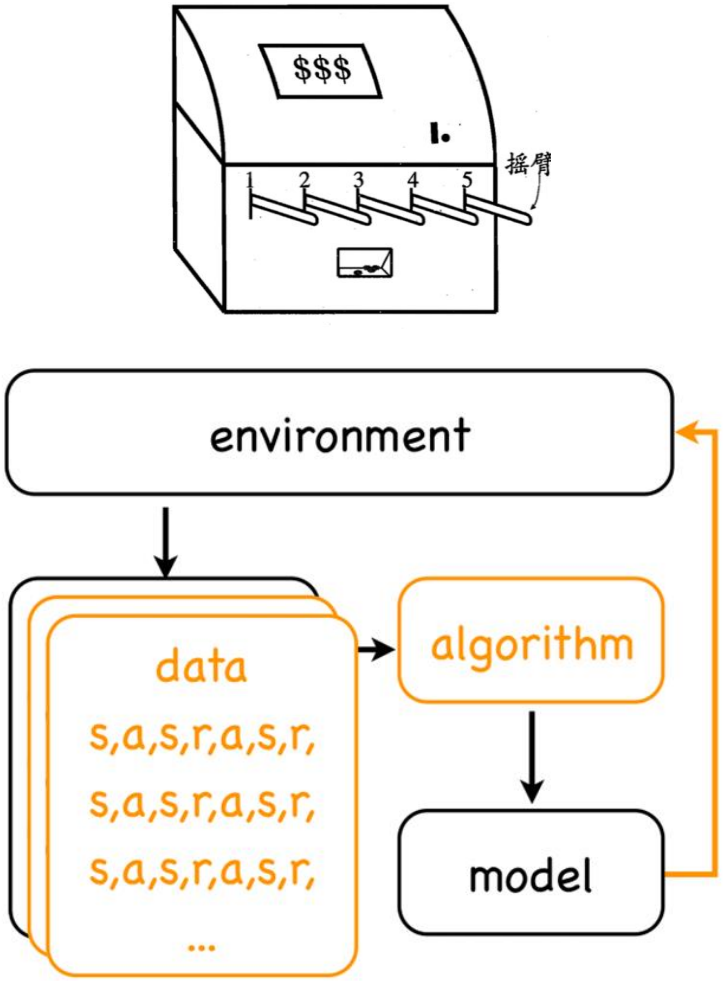
- 未知

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, $a \in \mathcal{A}(s)$



多臂赌博机 *Multi-armed Bandits*



赌徒投入一个硬币后，选择一个摇杆，每个摇杆有一定的概率吐出硬币，这个概率赌徒**并不知道**。赌徒的目标就是通过找到一个策略来使自己在等量成本下，收益最大。

多臂赌博机 *Multi-armed Bandits*

▶ 探索-利用两难 *exploration and exploitation tradeoff*

- 探索 *exploration*：为了获知每个摇臂的出币期望，那么策略应该把机会均匀分配给各个摇臂。通过出币样本，更新期望的近似估计。
- 利用 *exploitation*：如果有了期望知识，那么策略应该把机会留给出币期望最大的摇臂。
- 探索-利用两难：探索过多意味着不能获得较高的收益,而利用过多意味着可能错过更高回报的机会。

▶ ϵ-贪心方法

以概率ϵ进行探索，即以均匀概率随机选取一个摇臂；
以概率$1-\epsilon$进行利用，即选择当前平均奖赏最高的摇臂（若有多个，则随机选取一个）

蒙特卡洛方法

- 增量式实现

$$
\begin{aligned}
Q_n &= \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1} \\
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]
\end{aligned}
$$

为了计算 Q_{n+1} 只需要存储 Q_n 和 n

新的估计值 $\leftarrow$ 旧的估计值 + 步长 $\Big[$目标 $-$旧的估计值$\Big]$

- 增量式实现

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]
\end{aligned}$$

为了计算 Q_{n+1} 只需要存储 Q_n 和 n

新的估计值 $\leftarrow$ 旧的估计值 + 步长 $\Big[$目标 $-$旧的估计值$\Big]$

多臂赌博机 *Multi-armed Bandits*

- 应对非平稳环境

新的估计值 $\leftarrow$ 旧的估计值 + 步长 $\Big[$目标 $-$旧的估计值$\Big]$

$$Q_{n+1} = Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]$$

$$Q_{n+1} = Q_n + \alpha\Big[R_n - Q_n\Big]$$

多臂赌博机 *Multi-armed Bandits*

- 算法

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

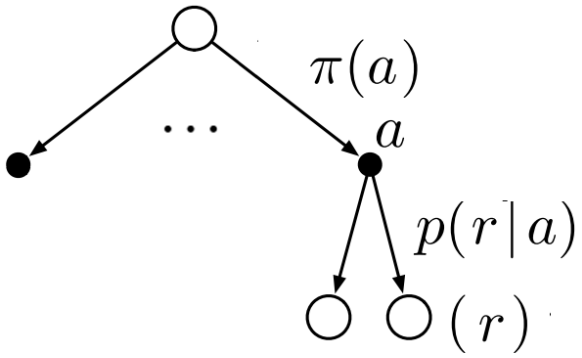
$R \leftarrow bandit(A)$

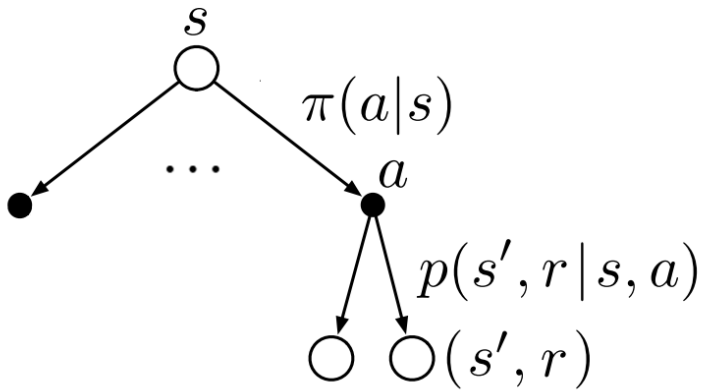
$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

基于蒙特卡洛的方法

- 对比





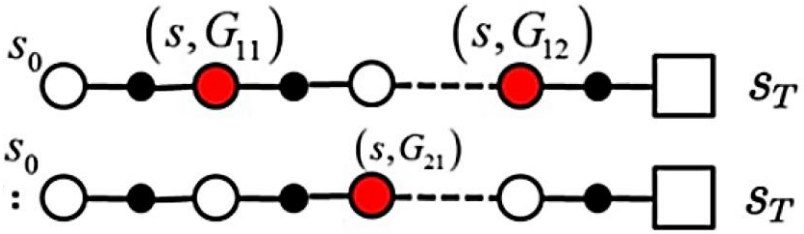
赌博机算法采样并平均每个动作的收益

$Q(a)$

蒙特卡洛算法采样并平均每个“动作-状态”二元组的收益

$Q(s, a)$

基于蒙特卡洛的方法



- 首次访问 *First-visit*

$$v(s) = \frac{G_{11}(s) + G_{21}(s) + \cdots}{N(s)}$$

- 每次访问 *Every-visit*

$$v(s) = \frac{G_{11}(s) + G_{12}(s) + \cdots + G_{21}(s) + \cdots}{N(s)}$$

基于蒙特卡洛的方法

- 算法

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

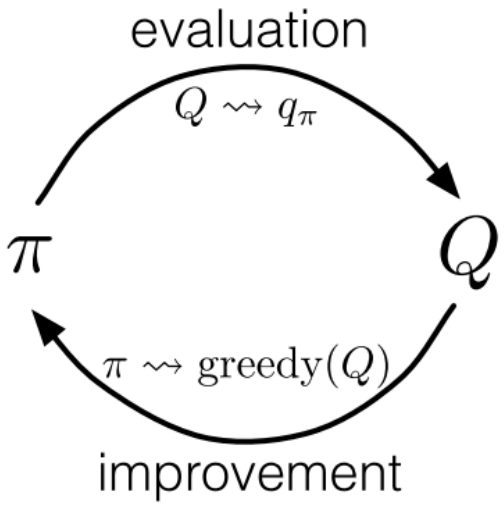
Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

基于蒙特卡洛的方法

- 策略迭代

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$



基于蒙特卡洛的方法

- 算法

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

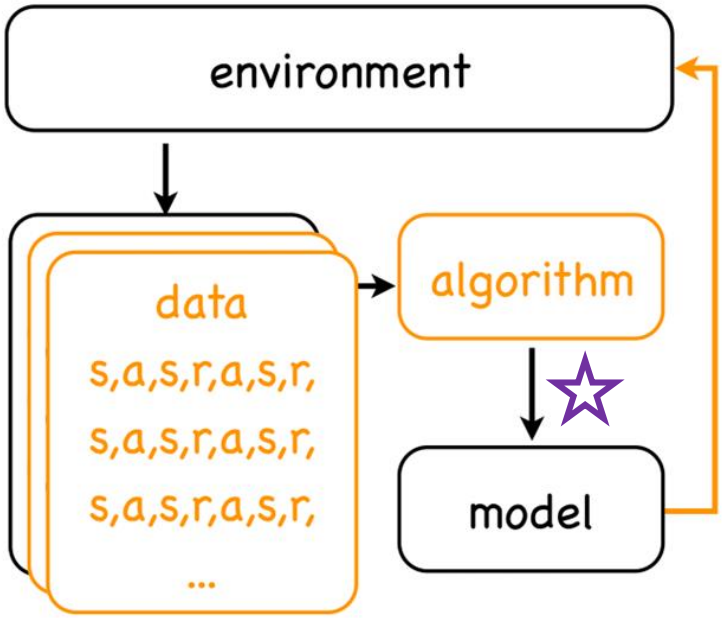
$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

基于时序差分的方法

- 动机



$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$（误差更新估计值）

对$V(S_t)$的更新一定要等到一次完整的访问完成后，取得回报后才开始实施。

基于时序差分的方法

$$S_0, A_0, R_1, \ldots, S_t, A_t, R_{t+1}, S_{t+1}, \ldots, S_{T-1}, A_{T-1}, R_T$$

▶ 思想

对$V(S_t)$的更新并不一定要等到一次完整的访问完成后，取得回报后才开始实施。
当在访问的过程中，可以等到下一个时刻的收益R_{t+1}获得后即用$R_{t+1} + \gamma V(S_{t+1})$
取代蒙特卡洛方法中的G_t来进行更新：

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big]$$

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big]$$

★ TD算法也使用了自己来更新自己，即利用$V(S_{t+1})$来计算误差更新$V(S_t)$，所以它可以看做是蒙特卡洛方法和DP“自举”法的结合。

★ 这种方式对应于向前走一步后开始更新，称为单步时序差分 *one-step TD*，记为$TD(0)$

★ n步时序差分 *one-step TD*：

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

基于时序差分的方法

▶ 思想

对$V(S_t)$的更新并不一定要等到一次完整的访问完成后，取得回报后才开始实施。
当在访问的过程中，可以等到下一个时刻的收益R_{t+1}获得后即用$R_{t+1}+\gamma V(S_{t+1})$
取代蒙特卡洛方法中的G_t来进行更新：

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

★ TD算法也使用了自己来更新自己，即利用$V(S_{t+1})$来计算误差更新$V(S_t)$，所以它可以看做是蒙特卡洛方法和DP“自举”法的结合。
★ 这种方式对应于向前走一步后开始更新，称为单步时序差分 *one-step TD*，记为 *TD(0)*
★ n步时序差分 *one-step TD*：

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

基于异策略的方法

- 同策略 *on-policy* 与异策略 *off-policy*

前面的算法版本是同策略的，即产生样本的行为策略 *behavior policy* 与评估和改善的目标策略 *target policy* 是同一个(ϵ-soft)策略。

然而，我们执行ε-贪心策略的原因是因为尽可能地让所有的状态动作对被访问到。但真实使用的时候我们并不会使用ε-贪心策略。所以我们评估和改善的对象可以改为非ε-soft策略，这叫做异策略。

基于异策略的方法

异策略的实现

记评估和改善的(目标)策略为 π

生成采样的行为策略为 $b \neq \pi$

如何令行为策略产生的数据能够帮助我们估计 v_π or q_π ?

基于函数逼近法

- 基于线性函数

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S) \mathbf{w}_j$$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S) \longrightarrow \Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \mathbf{x}(S)$$

$$\text{Update} = \textit{step-size} \times \textit{prediction error} \times \textit{feature value}$$

函数逼近法

- 如何获得 $v_\pi(s)$
  - 基于蒙特卡洛方法

$\langle S_1, G_1\rangle, \langle S_2, G_2\rangle, ..., \langle S_T, G_T\rangle$

  - 基于时序差分方法（以单步为例）

$\langle S_1, R_2 + \gamma\hat{v}(S_2, \mathbf{w})\rangle, \langle S_2, R_3 + \gamma\hat{v}(S_3, \mathbf{w})\rangle, ..., \langle S_{T-1}, R_T\rangle$

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

THANKS

Some images and slides are from the internet. If related to copyright, please contact me.