机器学习
Machine Learning

第13讲：强化学习-1

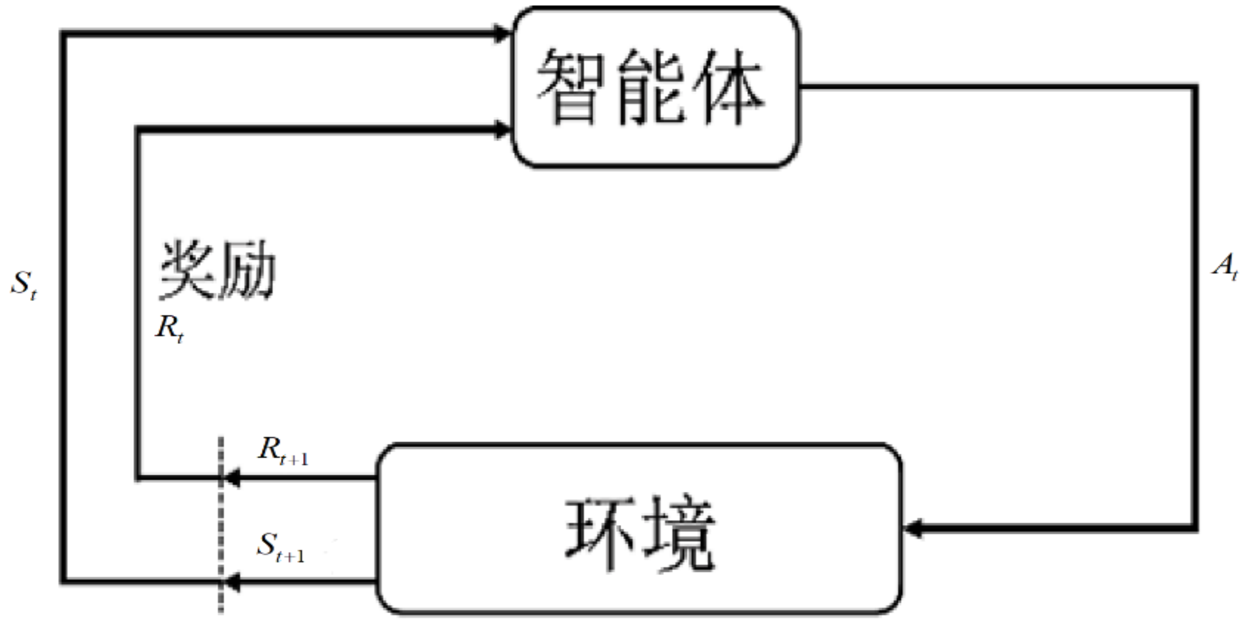
Reinforcement Learning-1

（Tabular Solution）

强化学习的基本问题

- 强化学习（增强学习）（reinforcement learning，RL）研究智能体基于对环境的认知做出行动来最大化长期收益，是解决智能控制问题的重要方法。
- 强化学习的主体为**智能体**（agent）。智能体面对一个**环境**（environment），与环境的交互，感知环境的状态并获得当前环境的奖励（reward），决策当前要采取的动作（action），以最大化决策策略所能获得的长期收益。

1. 强化学习的基本结构模型

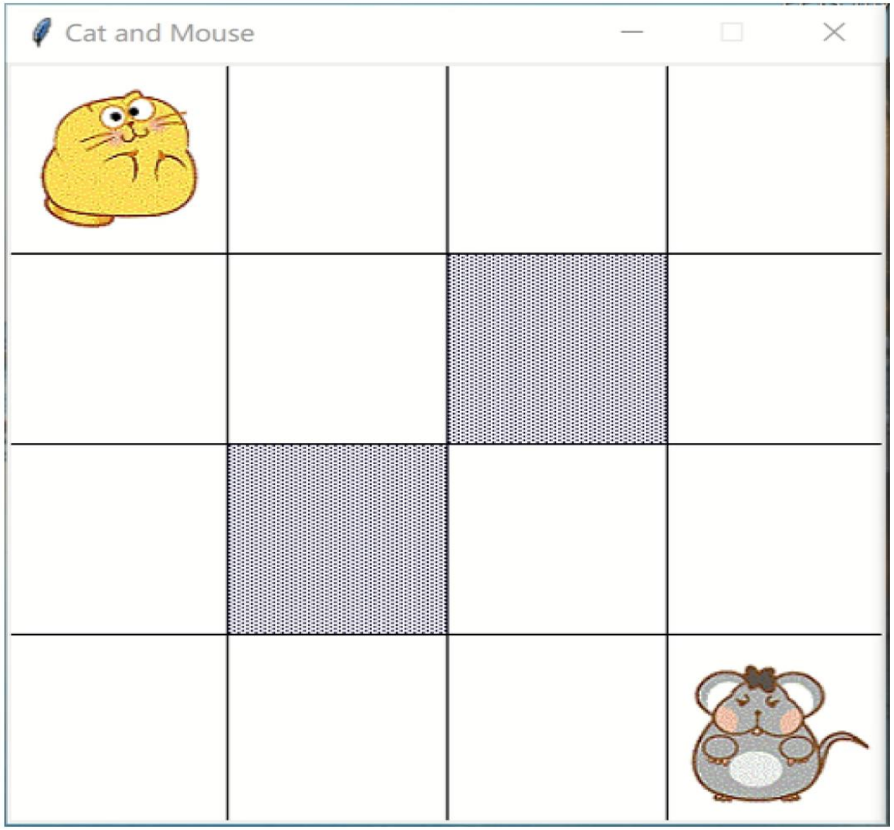


交互中产生：“状态、动作、奖励”的序列

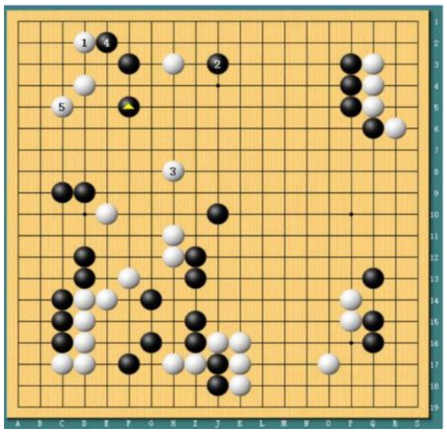
$$\{S_0, A_0, R_1, S_1, A_1, R_2, \cdots, S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \cdots,\}$$

强化学习的简单示例

一个简化的猫抓老鼠游戏



强化学习解决的实际示例



AlphaGo

机器人控制

对弈麻将（MSRA）

2. 马尔可夫决策过程

RL的大部分问题可建模为马尔可夫决策过程（Markov decision process，MDP）

定义：一个MDP由一个五元组 $(\mathcal{S},\mathcal{A},r,P_{ss'}^{a},\gamma)$ 构成

$\mathcal{S}=\{s^{(1)},s^{(2)},\cdots\}$ 表示状态集合；

$\mathcal{A}=\{a^{(1)},a^{(2)},\cdots\}$ 表示动作集合

$P_{ss'}^{a}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\rightarrow[0,1]$ 表示状态转移概率

$r:\mathcal{S}\times\mathcal{A}\rightarrow\mathbb{R}$ 是奖励函数

$\gamma\in[0,1]$ 表示折扣因子。

MDP定义的进一步解释

状态转移满足：马尔可夫性

$$P(S_{t+1}|S_0, S_1, \cdots, S_{t-1}, S_t) = P(S_{t+1}|S_t)$$

决策过程产生一个样本序列

$$\{S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \ \cdots, S_{T-1}, A_{T-1}, R_T, S_T\}$$

状态转移概率的定义

$$P_{ss'}^a = P(S_{t+1} = s'|S_t = s, A_t = a)$$

奖励函数的定义

$$r(s, a) = E(R_{t+1}|S_t = s, A_t = a)$$

例：猫和老鼠的例子

描述规则！

状态集合 $\mathcal{S}=\{1,2,\cdots,16\}$

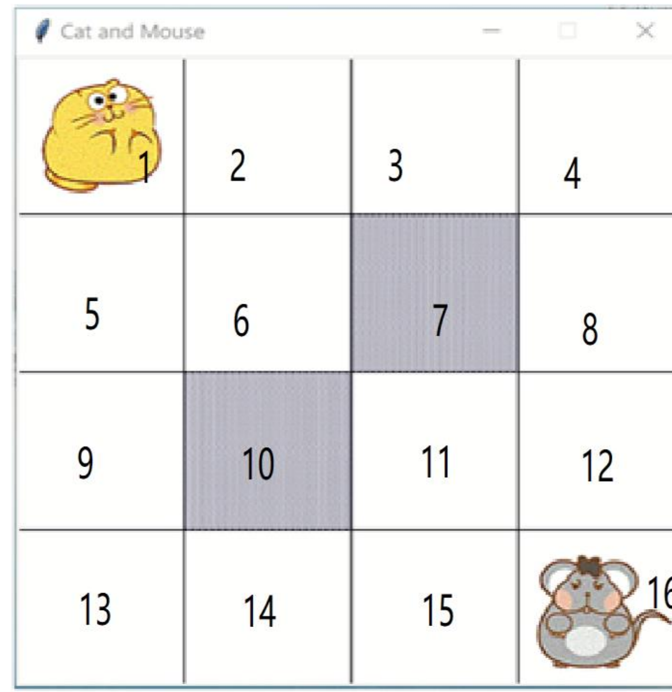
动作集合 $\mathcal{A}=\{up, down, left, right\}$

状态转移概率例子

$$P_{1,1}^{up} = P(S_{t+1} = 1|S_t = 1, A_t = up) = 1$$

$$P_{1,s'\neq 1}^{up} = P(S_{t+1} = s' \neq 1|S_t = 1, A_t = up) = 0$$

$$P_{1,2}^{right} = 1,\ P_{1,s'\neq 2}^{right} = 0$$



奖励例子

$$r(1, right) = E(R_{t+1}|S_t = 1, A_t = right) = -1$$

$$r(15, right) = 10,\ r(9, right) = -10$$

3. 强化学习的基本元素

状态和返回值

从 S_t 出发所获得的累积奖励：返回值（return）

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$
$$= \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

策略函数 在状态 $S_t = s$下，确定智能体的动作$A_t = a$

确定性策略 $a = \pi(s)$

随机策略 $\pi(a|s) = P(A_t = a | S_t = s)$

状态值函数（在给定策略下）

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s]
\end{aligned}
$$

动作-值函数

$$
\begin{aligned}
q_\pi(s,a) &= E_\pi[G_t|S_t = s, A_t = a] \\
&= E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s, A_t = a]
\end{aligned}
$$

4. 贝尔曼（Bellman）方程

决策过程中各状态之间有转移，表示MDP的状态之间值函数关系的一组方程称为贝尔曼（Bellman）方程

第一组形式方程

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

$$q_\pi(s,a) \\ = E_\pi\big[R_{t+1} + \gamma q_\pi\big(S_{t+1}, A_{t+1}\big)|S_t = s, A_t = a\big]$$

贝尔曼方程证明

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] \\
&= E_\pi\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right)|S_t = s\right] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]
\end{aligned}
$$

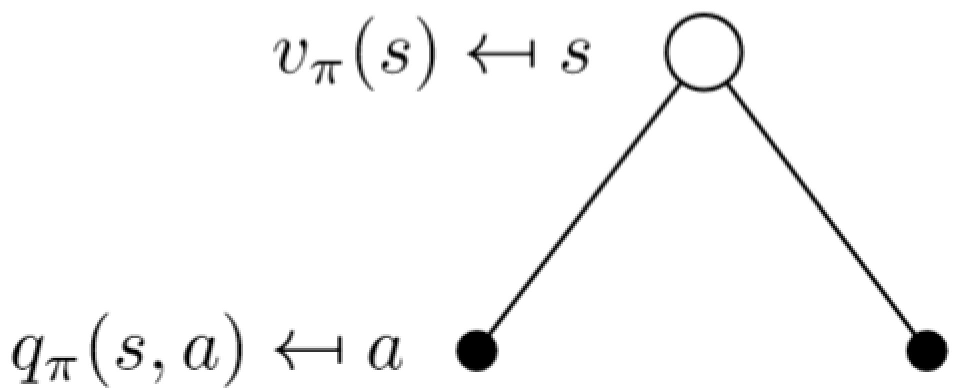
第2组形式

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^{a} v_\pi(s')\right)$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^{a} \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

第2组方程的导出和关系

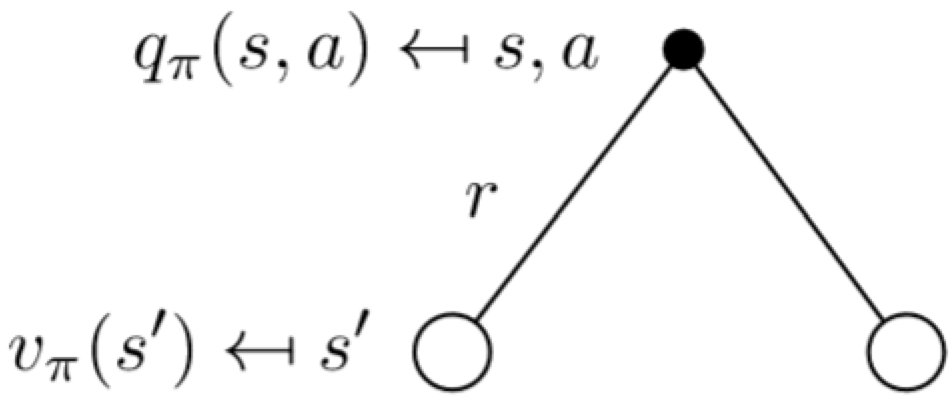
Bellman Expectation Equation for V^{π}



$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_{\pi}(s,a)$$

第2组方程的导出和关系（续）

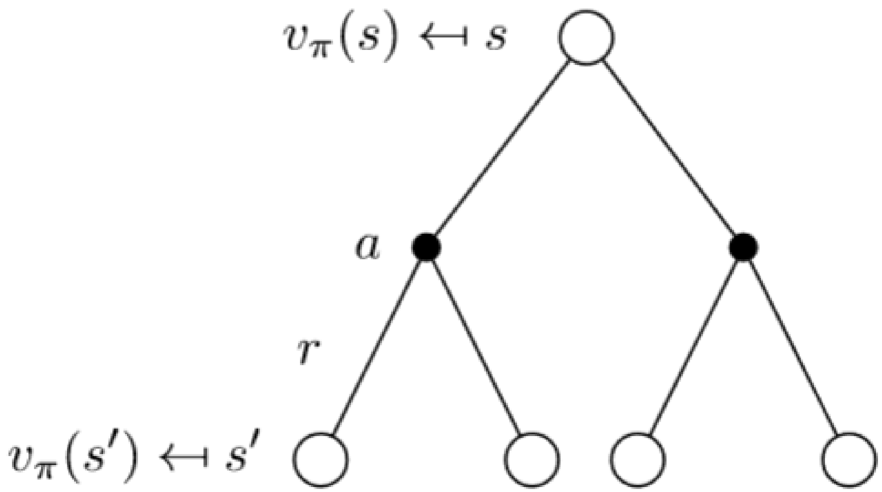
Bellman Expectation Equation for Q^π



$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_\pi(s')$$

第2组方程的导出和关系（续）

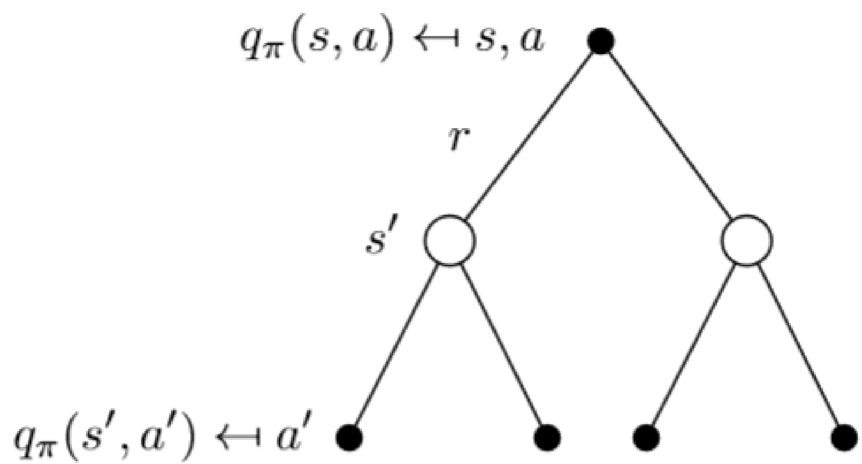
Bellman Expectation Equation for v_π (2)



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_\pi(s') \right)$$

第2组方程的导出和关系（续）

Bellman Expectation Equation for q_π (2)



$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^{a} \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

5. MDP的最优性

最优值函数：Optimal Value Function

Definition

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

$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

最优策略：Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

求最优策略：Find an Optimal Policy

$$\pi^*(a|s) = \begin{cases} 1, & 当 a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\{q_*(s,a)\} \\ 0, & 其他 \end{cases}$$

贪婪策略

6. Bellman最优方程

由

$$v_*(s) = \max_{a \in \mathcal{A}} \{q_*(s,a)\}$$

得

$$v_*(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^{a} v_*(s') \right\}$$

$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^{a} \max_{a' \in \mathcal{A}} \{q_*(s',a')\}$$

例：猫和老鼠的例子

右侧是上下左右等概率策略的值函数

−19.8	−17.8	−15.2	−14.9
−17.8	−14.4	0	−10.5
−15.2	0	−7.3	−3.6
−14.9	−10.5	−3.6	0

以下，左侧为一个更好的策略

右侧为该策略对于的值函数，实际上这是最优策略

↓→	→	→	↓
↓	←↑	•	↓
↓	•	↓→	↓
→	→	→	•

5.0	6.0	7.0	8.0
6.0	5.0	0.0	9.0
7.0	0.0	9.0	10
8.0	9.0	10	0.0

完全知道**MDP**模型！

7. 动态规划
Planning by Dynamic Programming

7.1 策略迭代方法

第一步：对于一个策略（起始时给出一个初始策略），利用贝尔曼期望方程迭代求策略对应的状态值函数，这一步称为策略评估（policy evaluation）；

第二步：利用所求的状态值函数，对策略进行改进，得到更好的策略，然后回到第二步，这一步称为策略改进（policy improvement）。

以上过程反复迭代，当改进后的策略不再变化，已得到最优策略

7.1.1 迭代策略评估
Iterative Policy Evaluation

从初始值 $v_0(s)$开始

迭代表示为

$$v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{ss'}^{a} v_k(s')\right)$$

直到满足

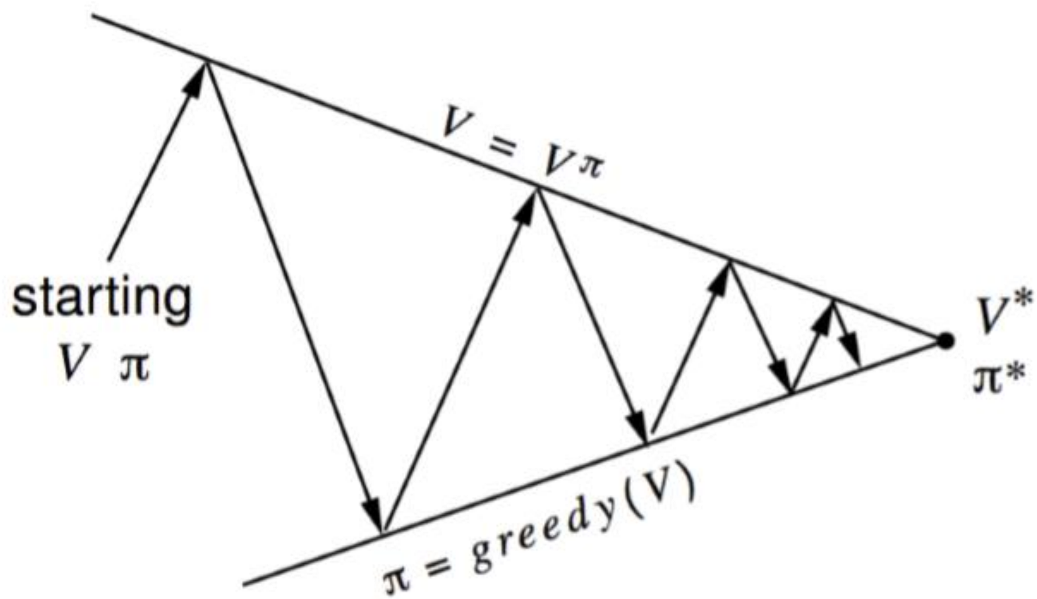
$$\max_{s\in\mathcal{S}} |v_{k+1}(s) - v_k(s)| < \delta$$

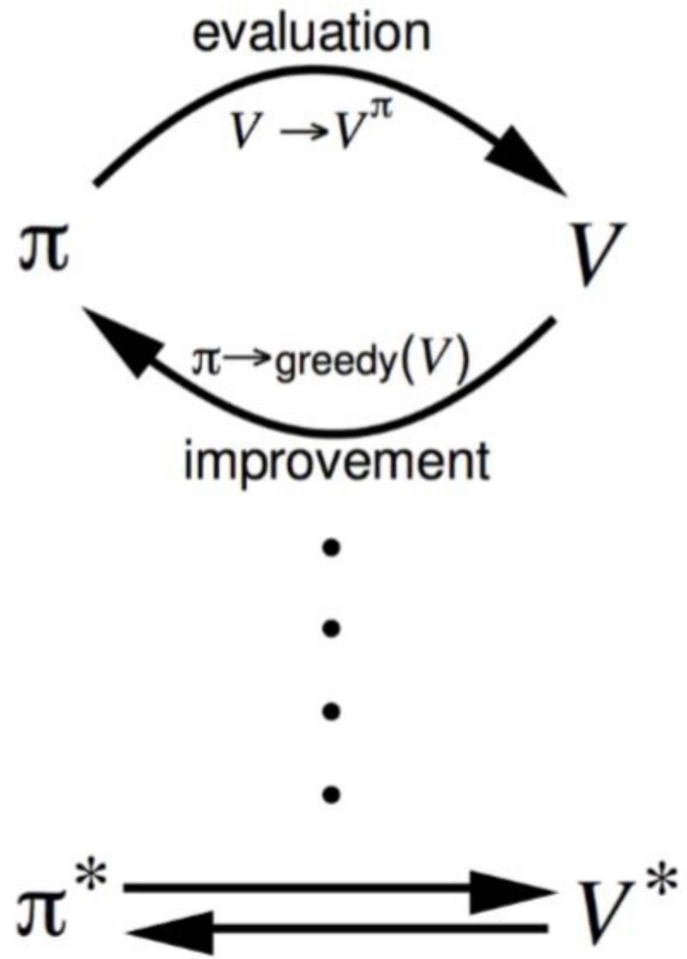
7.1.2 策略优化（ Improve a Policy）

$$\pi'(a|s) = \begin{cases} 1, & \text{当} a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^{a} v_{\pi}(s') \right\} \\ 0, & \text{其他} \end{cases}$$

改进策略的贪婪算法

策略迭代过程示意（ Policy Iteration）





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

例：猫和老鼠

初始策略为四方向等概率。(a)初始值函数，(b)值函数第一步迭代，(c)值函数收敛，(d)策略改进

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

(a)

-1	-1	$-\frac{13}{4}$	-1
-1	$-\frac{22}{4}$	0	$-\frac{13}{4}$
$-\frac{13}{4}$	0	$-\frac{22}{4}$	$\frac{7}{4}$
-1	$-\frac{13}{4}$	$\frac{7}{4}$	0

(b)

–19.8	–17.8	–15.2	–14.9
–17.8	–14.4	0	–10.5
–15.2	0	–7.3	–3.6
–14.9	–10.5	–3.6	0

(c)

↓→	↓	↓	↓
→	↓→	•	↓
→	•	↓→	↓
→	→	→	•

(d)

7.2 广义策略迭代
generalized policy iteration, GPI

策略评估不必到收敛，只做部分策略评估，则进入策略改进，形成一个链式算法

$$\pi_0 \xrightarrow[\text{部分评估}]{} v_{\pi_0} \xrightarrow[\text{改进}]{} \pi_1 \xrightarrow[\text{部分评估}]{} v_{\pi_1} \xrightarrow[\text{改进}]{} \pi_2 \cdots \rightarrow \pi^* \rightarrow v_*$$

例：猫和老鼠

初始策略为四方向等概率。(a)初始值函数，(b)值函数第一步迭代，(c)一步值函数迭代后更新的策略

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

(a)

-1	-1	$-\frac{13}{4}$	-1
-1	$-\frac{22}{4}$	0	$-\frac{13}{4}$
$-\frac{13}{4}$	0	$-\frac{22}{4}$	$\frac{7}{4}$
-1	$-\frac{13}{4}$	$\frac{7}{4}$	0

(b)

↔↕	←↑	↔	↑→
←↑	←↑	•	↓
↕	•	↓→	↓
←↓	→	→	•

7.3 值函数迭代（Value Iteration）

利用贝尔曼最优方程，直接迭代最优值函数

最后由最优值函数，得到最优策略

$$v_{k+1}(s) = \max_{a\in\mathcal{A}}\left\{r(s,a)+\gamma\sum_{s'\in\mathcal{S}}P_{ss'}^{a}v_k(s')\right\},\ \forall s\in\mathcal{S}$$

通过实际交互学习
需要有一个完整**EPISODE**！

8. MC强化学习 Monte-Carlo Reinforcement Learning

智能体通过与环境的交互进行学习，最终得到一种逼近最优的策略

由于需要智能体在环境中进行实际交互，将智能体从开启到结束的过程称为一次试验，一种类型是一次试验的步数有限，将这种类型的试验称为一分幕（episode）

蒙特卡洛方法只用于分幕环境

用MC做策略评估的基本思路

Monte-Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

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

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

MC策略评估算法-1：首次访问计数

First-Visit Monte-Carlo Policy Evaluation

每次完成一个episode，计算G_t，然后按如下更新V

- To evaluate state s
- The first time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

MC策略评估算法-2：每次访问计数

Every-Visit Monte-Carlo Policy Evaluation

每次完成一个episode，计算G_t，然后按如下更新V

- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

均值的增量计算，启发MC的增量算法

Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally,

$$
\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\
&= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)
\end{aligned}
$$

MC的增量算法

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t

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

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

动作-值函数更新

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}\big(G_t - Q(S_t, A_t)\big)$$

表示为学习率形式

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta\big(G_t - Q(S_t, A_t)\big)$$

MC的策略改进

利用一幕的序列计算部分策略评估，进行策略改进

$\varepsilon -$ 贪婪策略

$$\pi(a|s) = \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|\mathcal{A}(s)|}, & \text{当} a = a^* \\ \dfrac{\varepsilon}{|\mathcal{A}(s)|}, & \text{其他动作} \end{cases}$$

简记为：

$$\pi(a|s) = \varepsilon\text{−greedy}\big(Q(s,a)\big)$$

通过实际交互学习！实时！

9. 时间差分学习（TD类算法）Temporal-Difference Learning

MC方法要求一幕结束后，才可以更新值函数

给出一种实时性更高、更灵活的算法。在最基本的情况下，交互过程每进行一步，就可以更新状态值函数

算法称为时序差分算法（temporal difference, TD），基本的TD算法或称为TD(0)算法

参考增量MC算法导出TD算法

重写MC计算值函数的迭代公式

$$V(S_t) \leftarrow V(S_t) + \eta\big(G_t - V(S_t)\big)$$

其中

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$
$$= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \cdots) = R_{t+1} + \gamma G_{t+1}$$

可近似为

$$G_t \approx R_{t+1} + \gamma V(S_{t+1})$$

TD算法：值函数的一步更新

$$V(S_t) \leftarrow V(S_t) + \eta\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$

定义TD误差（TD error）

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

更新公式为

$$V(S_t) \leftarrow V(S_t) + \eta\delta_t$$

更经常使用的是动作-值函数，其更新为

$$Q(S_t, A_t) \\ \leftarrow Q(S_t, A_t) + \eta\big(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big)$$

每次值函数更新后，立刻用更新后的值函数，进行策略更新，用$\varepsilon -$贪婪策略更新策略

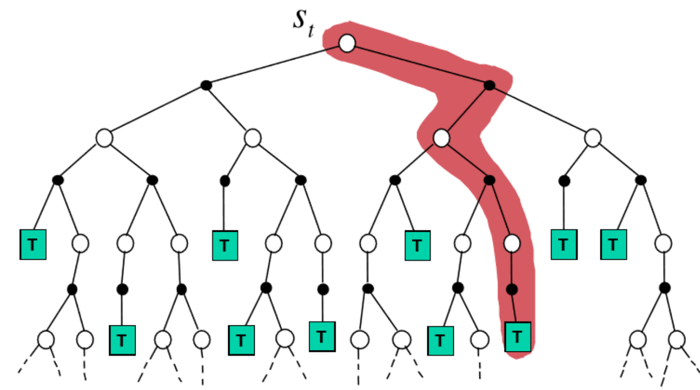
Sarsa算法

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

10. 三种方法的Backup关系图比较

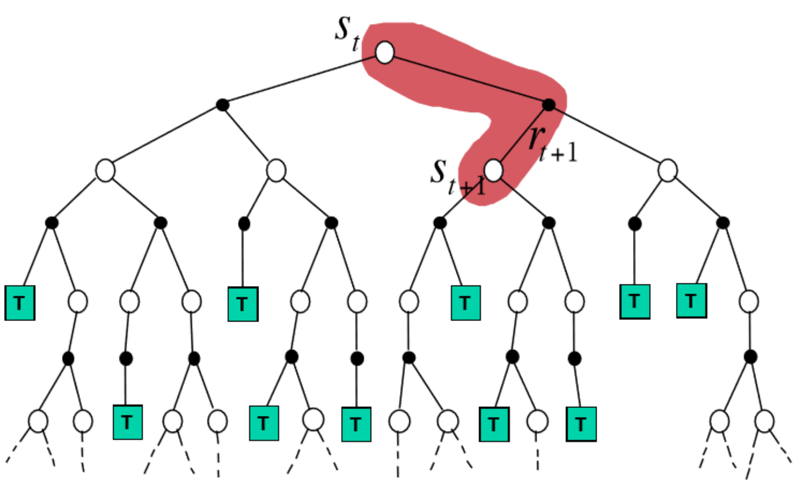
Monte-Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



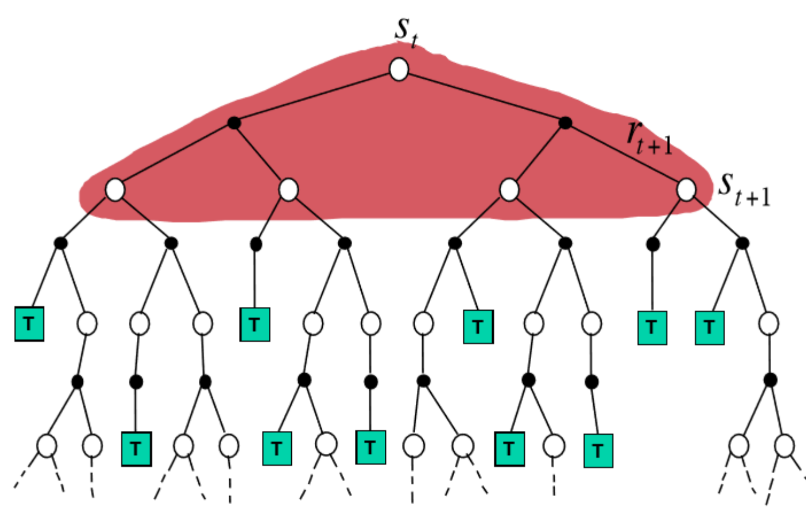
Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right]$$



11. Q-学习

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

Off-Policy Q-学习

Off-Policy Control with Q-Learning

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

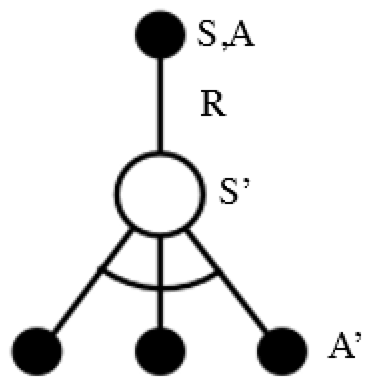
$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ =& R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')) \\ =& R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Q-学习算法

Q-Learning Control Algorithm



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S,A) \right)$$

Theorem

Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow q_*(s,a)$

Off-Policy Q-学习算法描述

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$;
  until S is terminal

12. 学习算法比较（DP和TD）（续）

Relationship Between DP and TD (2)

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	TD Learning $V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}[R + \gamma Q(S',A') \mid s,a]$	Sarsa $Q(S,A) \xleftarrow{\alpha} R + \gamma Q(S',A')$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	Q-Learning $Q(S,A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$