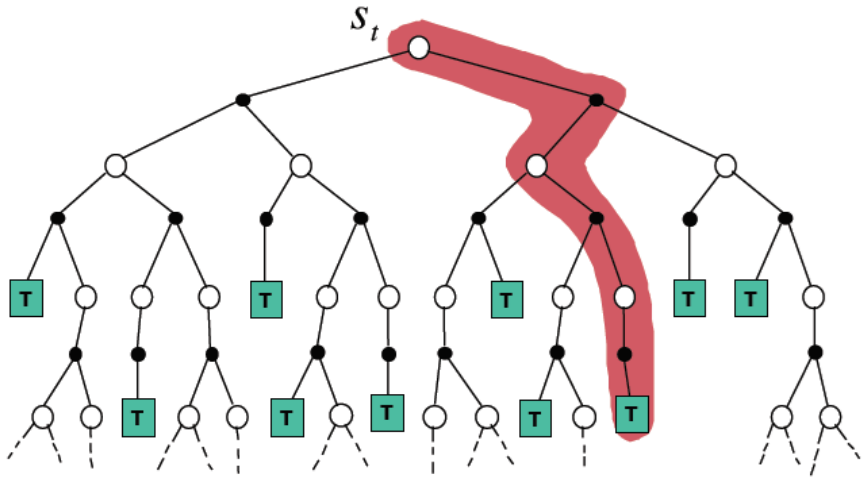
《机器学习及其在化学中的应用》2025年课程

Sec. 13

强化学习3：三种解法

（动态规划、蒙特卡罗、时序差分算法）

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



刘志荣 (LiuZhiRong@pku.edu.cn)

北京大学化学学院

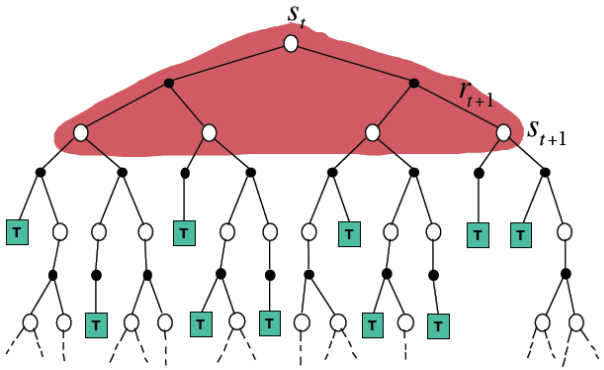
2025.12.8

内容提要

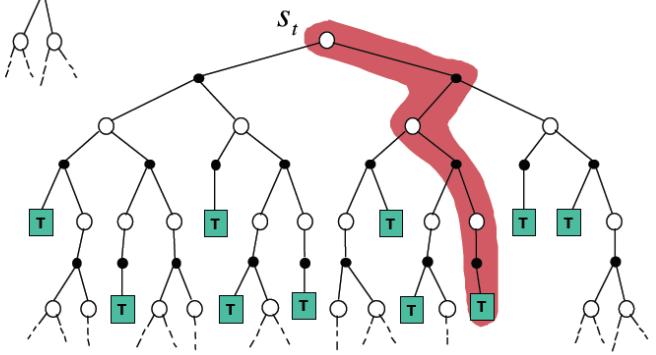
$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

- 动态规划
- 蒙特卡罗方法
- 时序差分算法

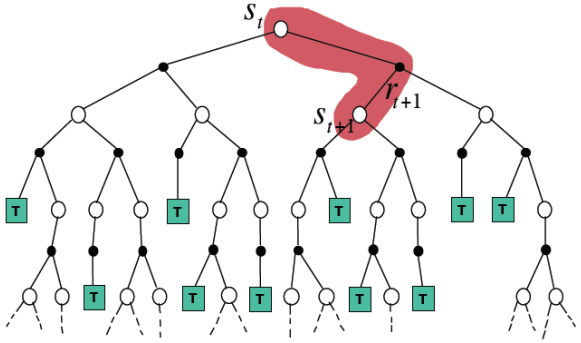
$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$



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

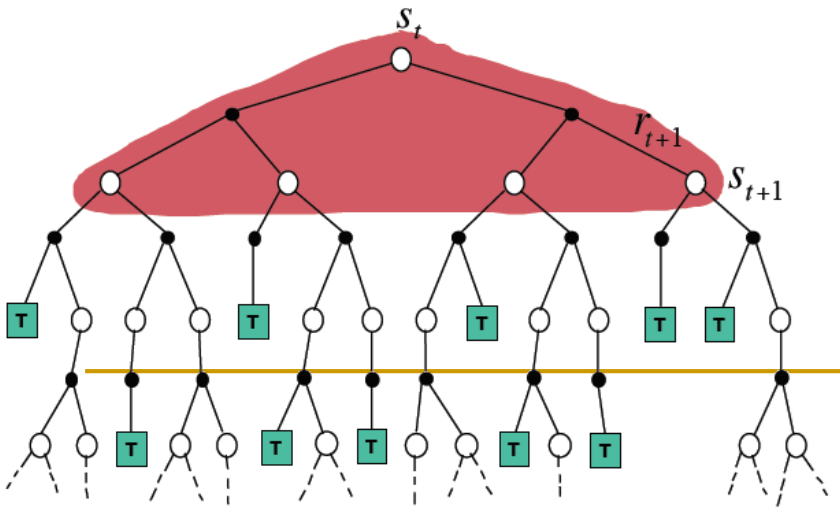


$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$



1. 动态规划
（Dynamic Programming）

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



动态规划（dynamic programming）

- 运筹学的一个分支，是求解决策过程最优化的数学方法。
- 20世纪50年代初R. E. Bellman等人在研究多阶段决策过程的优化问题时，提出了最优化原理（principle of optimality），把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解，从而创立了解决这类过程优化问题的新方法——动态规划。

- 动态规划可用于**模型完全已知**的马尔科夫决策过程的最佳策略的求解。
  - $p(s', r|s, a)$

策略评估（policy evaluation）

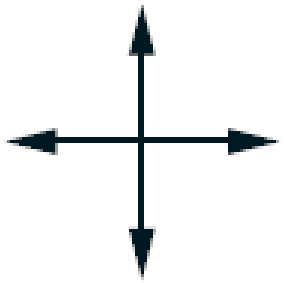
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

- 可作为线性方程组直接求解，
- 或通过迭代法求解：

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

- 在v_π存在的条件下， v_k在$k \to +\infty$时收敛于v_π

例子

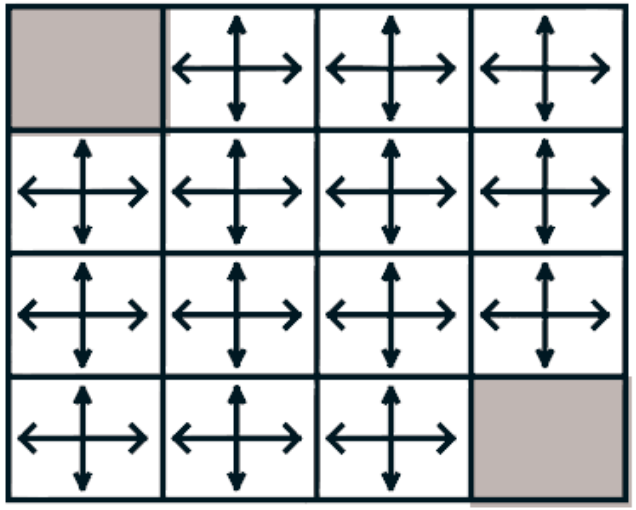


actions

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

$$R_t = -1$$
on all transitions

落到阴影中时停止。



random policy

$k = 0$

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

$k = \infty$

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

策略改进（policy improvement）

- 面对状态s时，如果遵循策略π，则结果（价值）是$v_\pi(s)$。
- 如果不遵循π，能更好吗？
- 如果此时选择行动a，但后面继续遵循π，则结果为

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

  - 如果$q_\pi(s,a) > v_\pi(s)$，则这种选择确实比π更好。
  - 如果下次又碰到s，应用同样的推理，还会选择a

- 策略改进定理：对于确定性的策略π和π'，如果对任意状态s有

$$q_\pi(s,\pi'(s)) \geq v_\pi(s)$$

则π'比π更好，或至少与π一样好：$v_{\pi'}(s) \geq v_\pi(s)$。

贪心策略...

- 基于策略π的价值函数v_π，可做向前单步搜索，构造如下贪心策略

$$\pi'(s) \doteq \arg\max_a q_\pi(s,a)$$

$$= \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

它比π更好或至少一样好。

  - 可据此进行策略改进。

- 如果贪心策略与原策略一样好，则

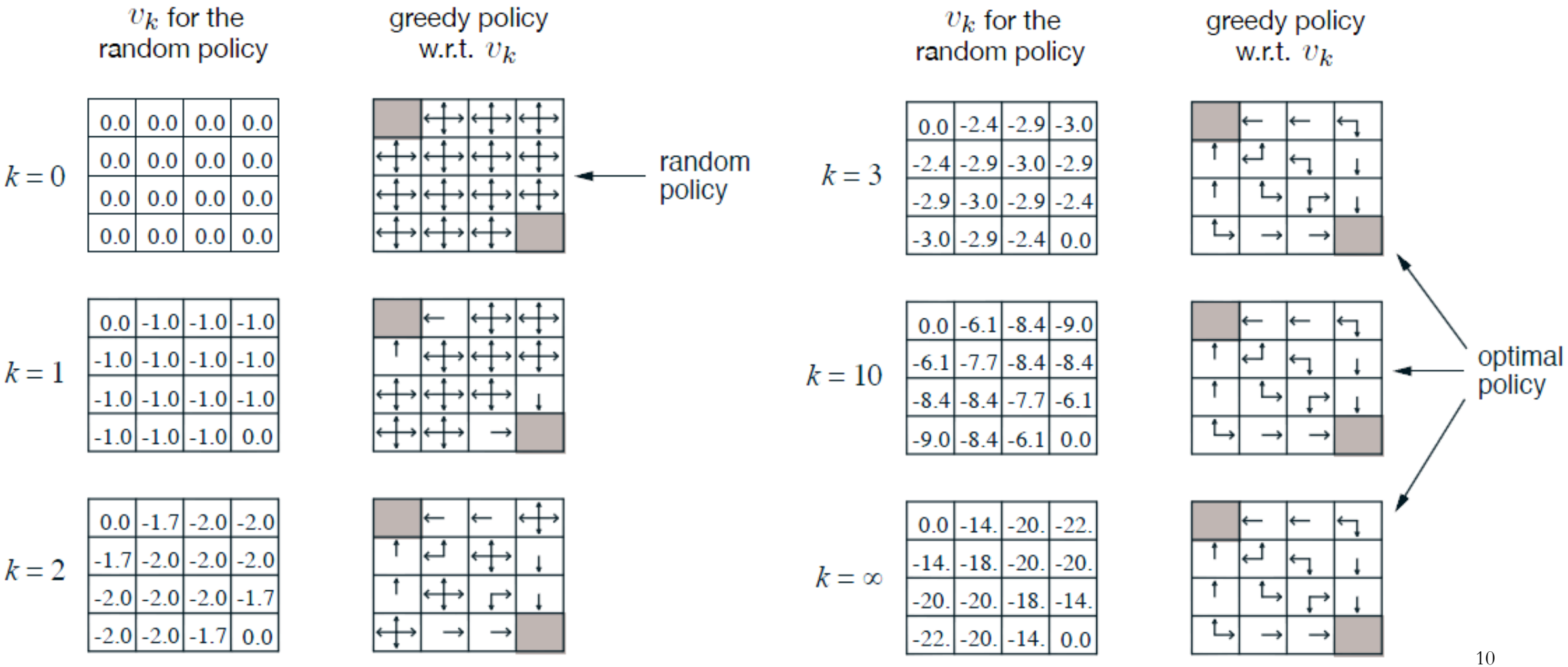
$$v_\pi(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

满足贝尔曼最优方程，因此原策略与贪心策略都是最佳策略。

例子...

在本例子中，基于完全随机的策略，做单步贪心改进，就可得到最优策略！

而且在策略评估阶段不需完全收敛。



策略迭代

- 一旦根据一个策略π的价值函数v_π进行策略改进得到一个新的策略π'，我们就可计算其价值函数$v_{\pi'}$并据此进行策略改进得到进一步提高的π''，…

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

- 策略迭代算法：

（1）策略评估（E）：计算策略π_k的价值函数$v_{\pi_k}(s)$；

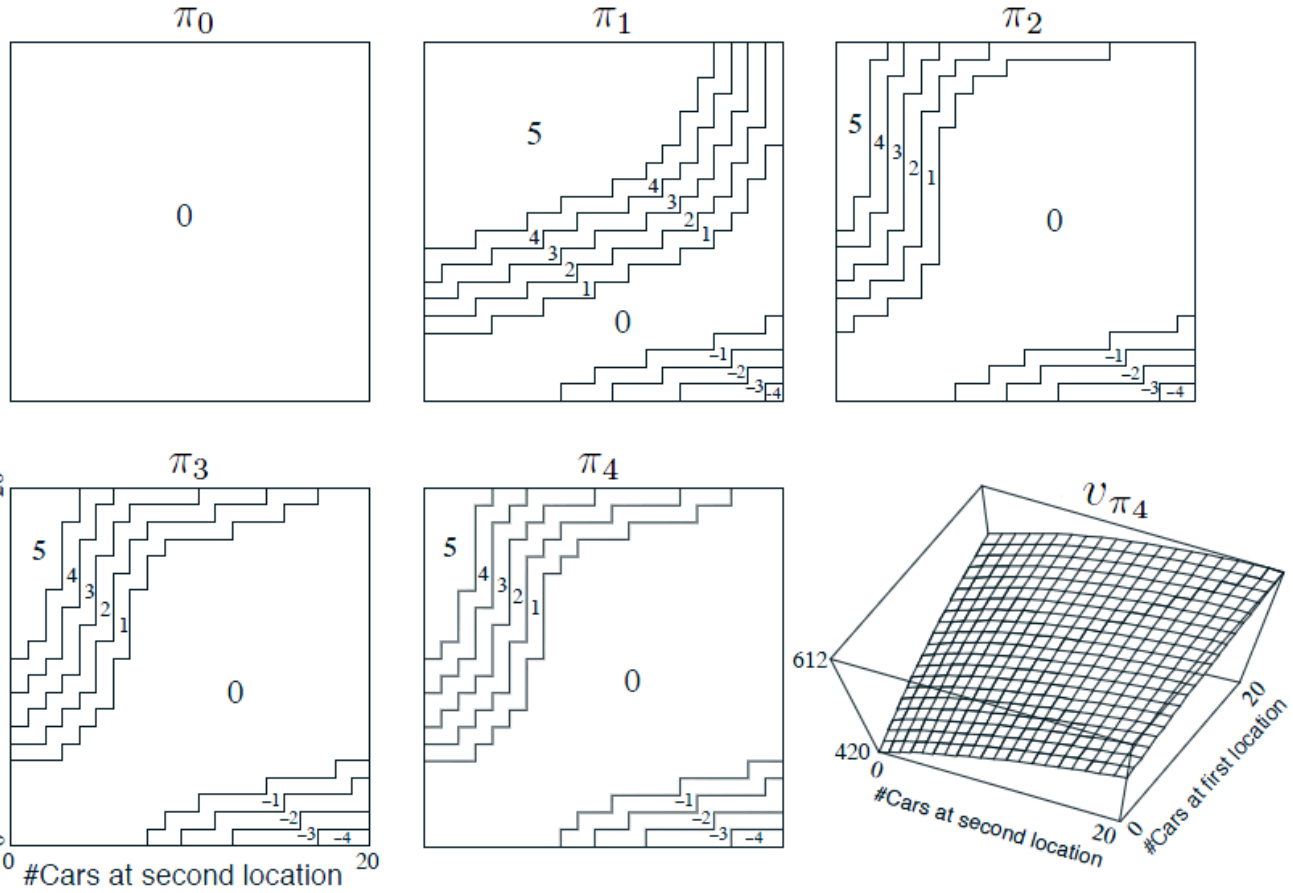
（2）策略改进（I）：根据π_k应用贪心算法得到改进策略π_{k+1}；

（3）$k \leftarrow k+1$，回到（1）。

在$k \to +\infty$时π_k收敛于最优策略π_*

例子：租车的调度

- 有个分公司管理两个停车点（最多停20辆）。每租出一辆车能从总公司获得10元。夜间可在两个地点之间移动车辆（最多5辆），每辆代价2元。假设每天租车与还车的数量满足泊松分布 $\frac{\lambda^n}{n!}e^{-\lambda}$。



租车：$\lambda = 3, 4$

还车：$\lambda = 3, 2$

$\gamma = 0.9$

价值迭代

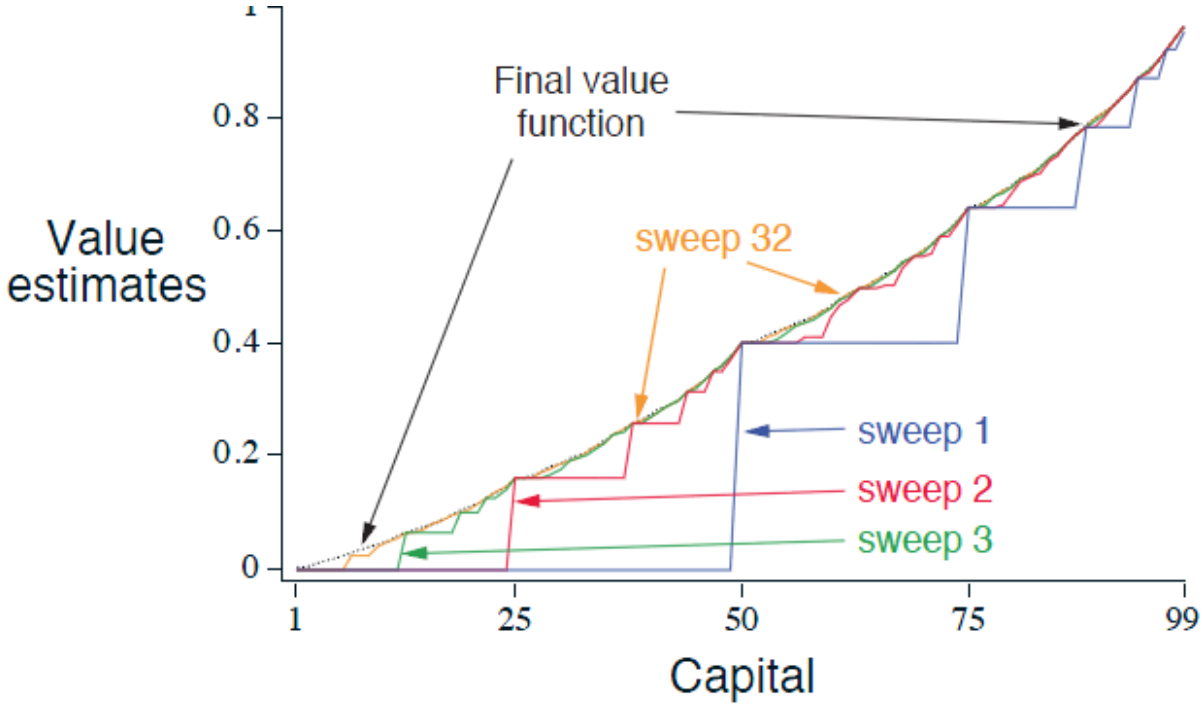
- 在策略迭代算法的策略评估步骤，需要反复迭代才能得到策略π_k的价值函数$v_{\pi_k}(s)$。
- 由于π_k本来就是要不断改进的，精确计算$v_{\pi_k}(s)$并无必要。
- 如果只迭代一次计算$v_{\pi_k}(s)$，则得到价值迭代算法：

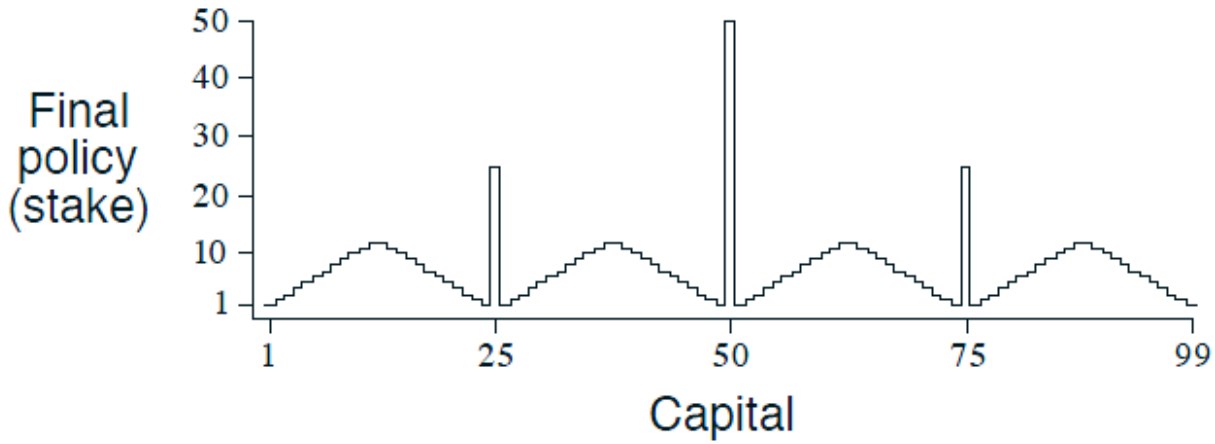
$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

  - 可看成是贝尔曼最优方程的直接迭代算法。

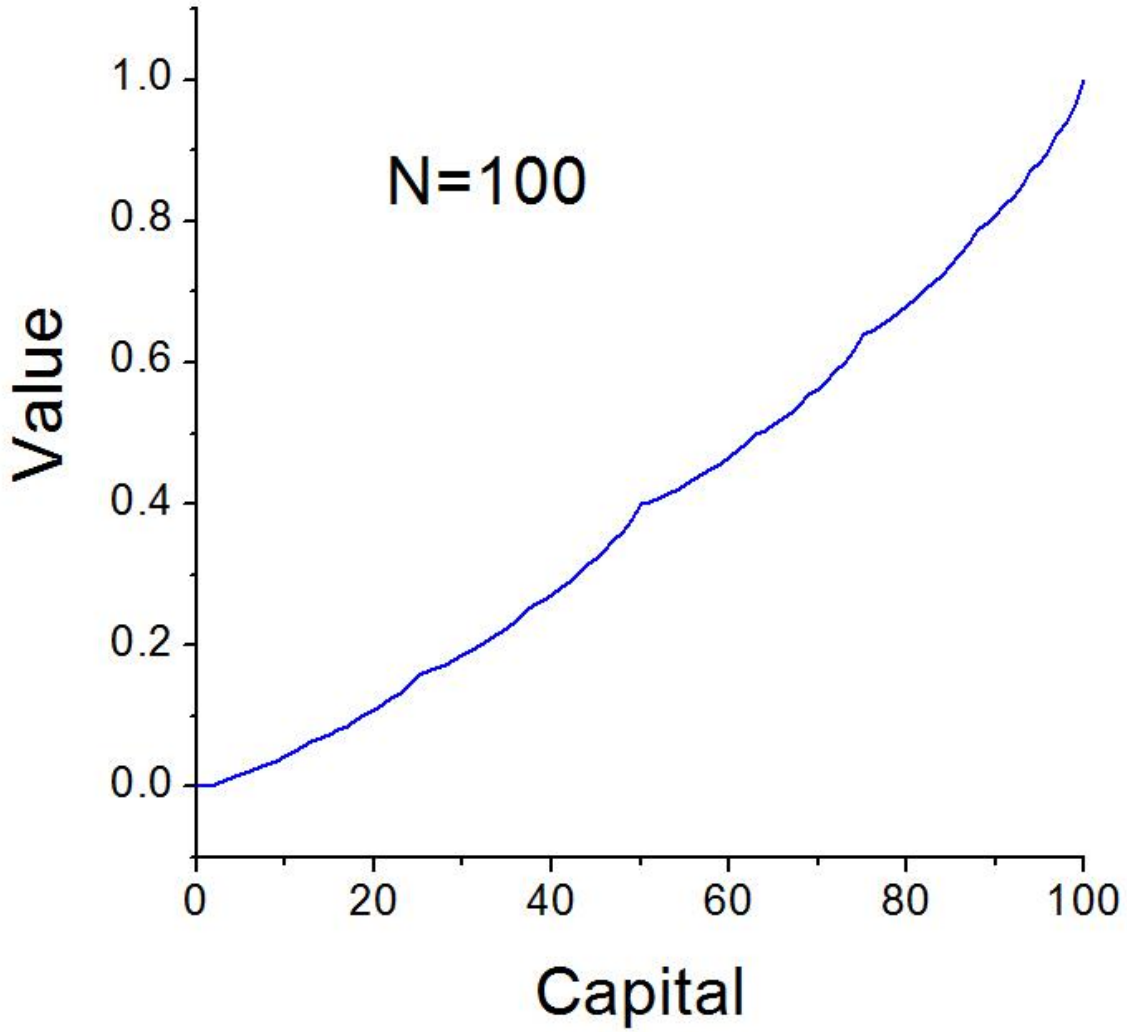
例子：赌徒问题 (Gambler's Problem)

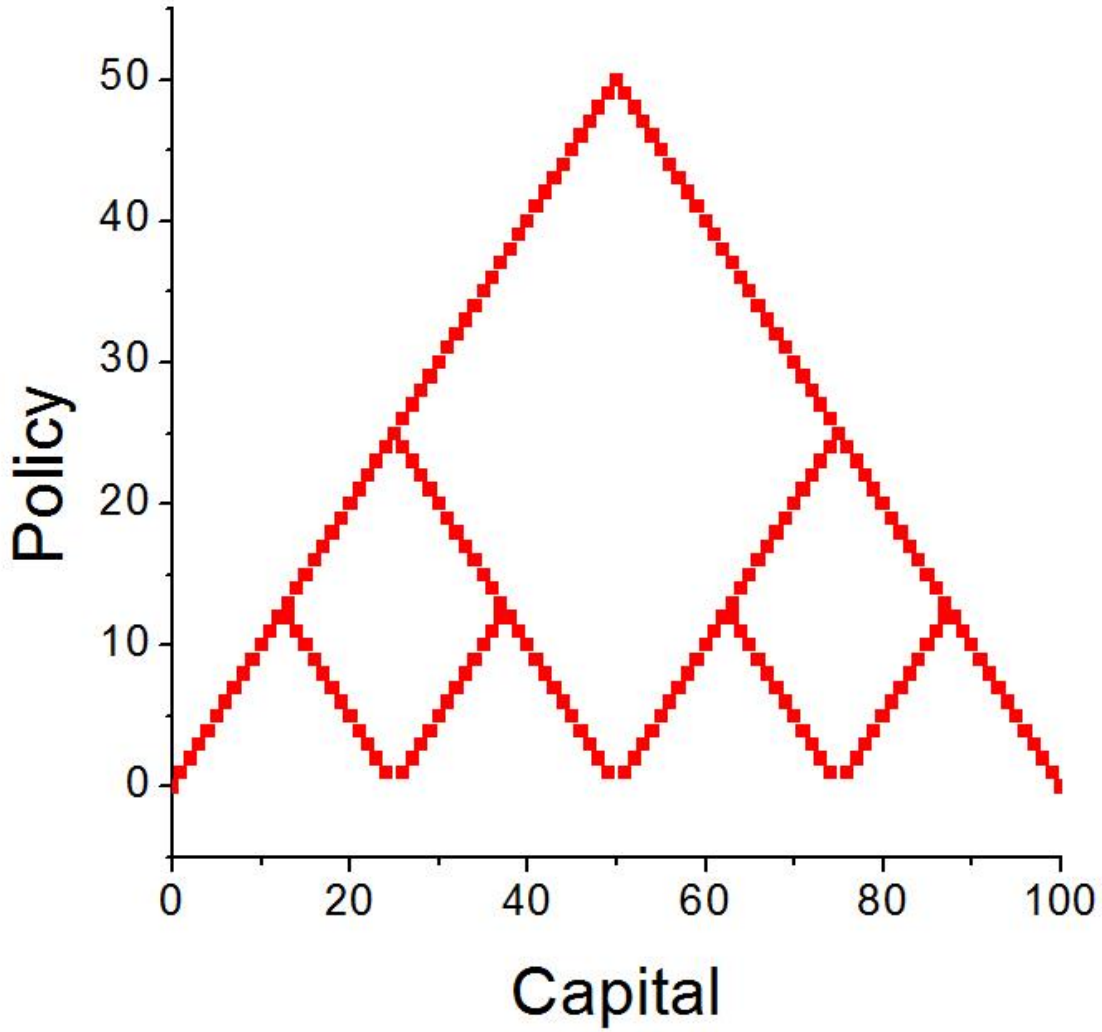
- 抛硬币（向上概率$p_h = 0.4$），钱数达到100元或输光时结束。
- 状态：现有金额。 行动：下注金额（整数）。





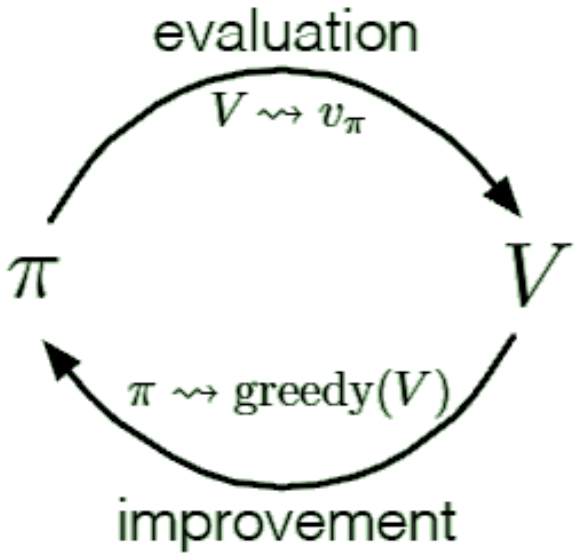
我的编程结果：

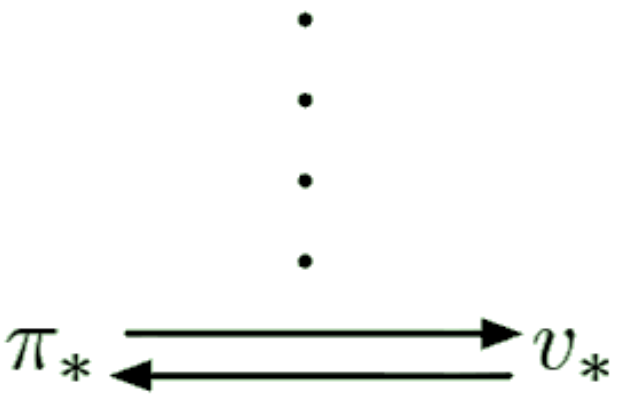

N=100
Value
Capital
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

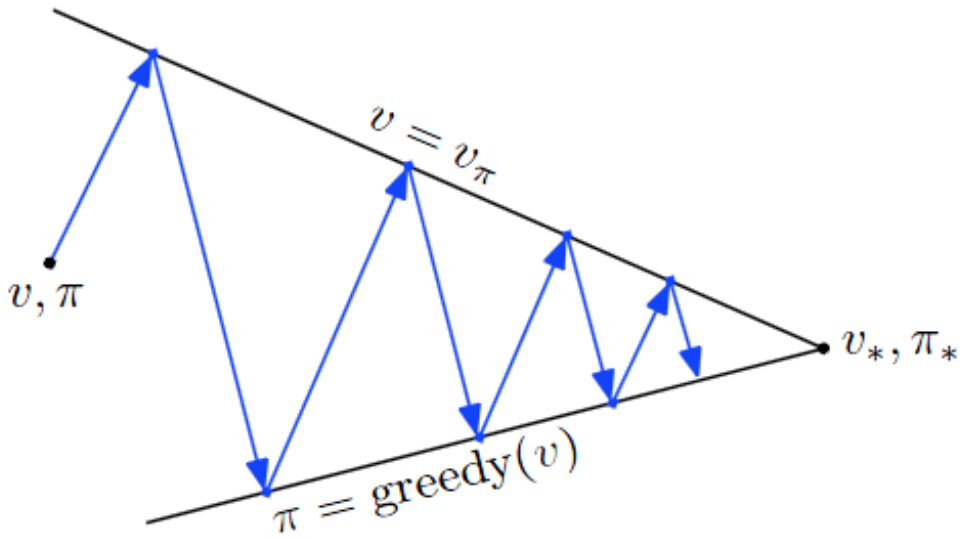

Policy
Capital
50
40
30
20
10
0
0
20
40
60
80
100

广义策略迭代（Generalized Policy Iteration）

- 前述做法可以有很多改动
  - 状态价值的更新可以异步进行；
  - 只有部分状态的价值与策略进行更新；
  - …
- 核心是策略评估（E）与策略改进（I）不断交替。





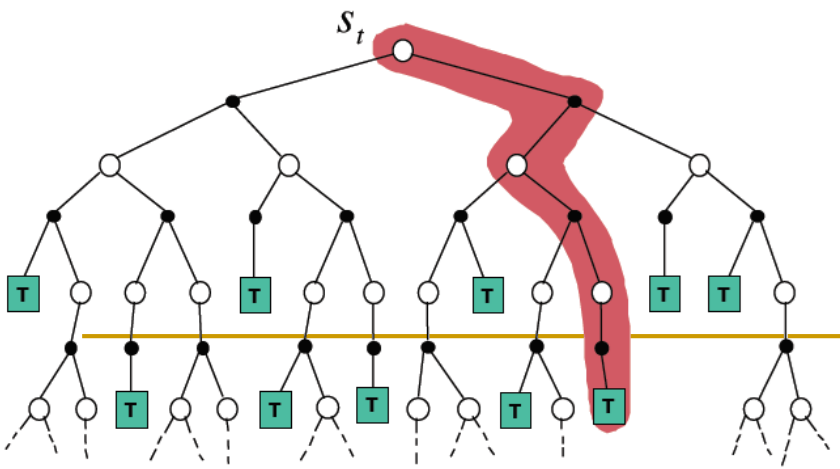


动态规划的效率

- 计算复杂度是状态与行动数目的多项式，指数级优于在策略空间的直接搜索算法。
- 能处理百万量级的状态数目。
- 在维度灾难时直接应用会有困难。例如，双陆棋状态数目~10^{20}。

2. 蒙特卡罗方法
（Monte Carlo methods）

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



蒙特卡罗（Monte Carlo，MC）方法

- 不需要关于环境的完整信息。
- 从经验中学习，即来自真实或模拟的环境交互中采样得到的状态、行动、回报的序列。
- 想法：用采样序列的性质平均值来估计（代替）其理论期望值。
- 例如：通过抛硬币来估计正面朝上的概率（期望值）。
- 例如：状态s在策略π下的价值函数是其最后达到目标（return）的期望值，$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$；而从一条轨迹我们可直接得到

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

进而从多条轨迹得到G_t的平均值来作为$v_\pi(s)$的估计。

首次访问型MC

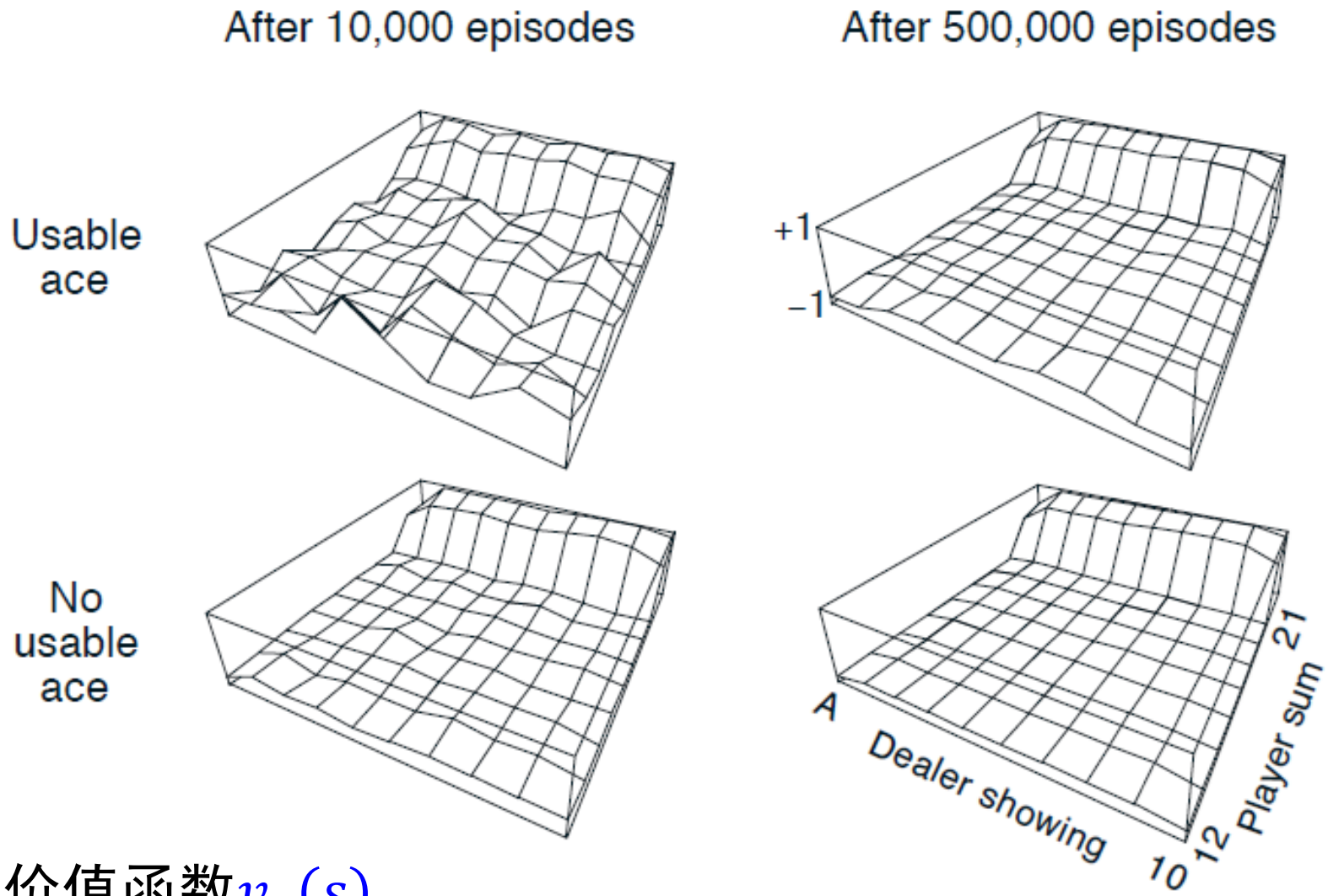
- 输入：待评估的策略π
- 初始化：
  - 对所有的s，用某种缺省值初始化$V_\pi(s)$；
  - 对所有的s，$Returns(s) \leftarrow$ 空列表；
- 无限循环（对每幕）：
  - 根据π生成一幕序列：$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots S_{T-1}, A_{T-1}, R_T$
  - $G \leftarrow 0$
  - 对本幕中的每一步进行循环，$t = T-1, T-2, \ldots, 0$：
    - $G \leftarrow \gamma G + R_{t+1}$；
    - （除非S_t在$S_0, S_1, \ldots, S_{t-1}$中出现过）

      将G加入$Returns(S_t)$，$V_\pi(S_t) \leftarrow \text{average}\big(Returns(S_t)\big)$

例子：21点（blackjack）

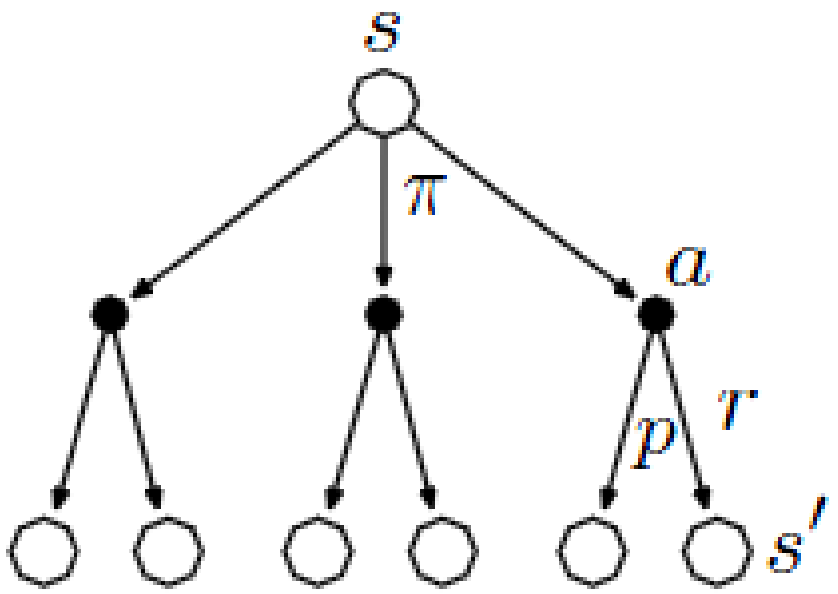
- 庄家有一张明牌，且只能按固定策略：一直要牌，直到点数≥ 17
- 行动：要牌或不要牌。
- 需要做决策的状态（200个）：手里牌的点数（12~21），庄家的明牌（A~10），自己手里是否有可用的A（按11算，未来可重计为1）。
- 虽然关于环境的模型是已知的，但$p(s', r|s, a)$的计算并不容易，较难直接应用动态规划方法。

- 策略：一直要牌直到点数≥ 20

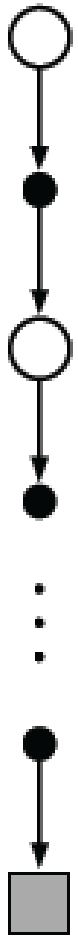


状态价值函数$v_\pi(s)$

- 动态规划需要考虑所有状态的一步转移；MC考虑多步转移，但每步只需要考虑（产生）一个状态。
- MC对每个状态的估计是独立的；动态规划各个状态的$v_\pi(s)$是互相影响的。



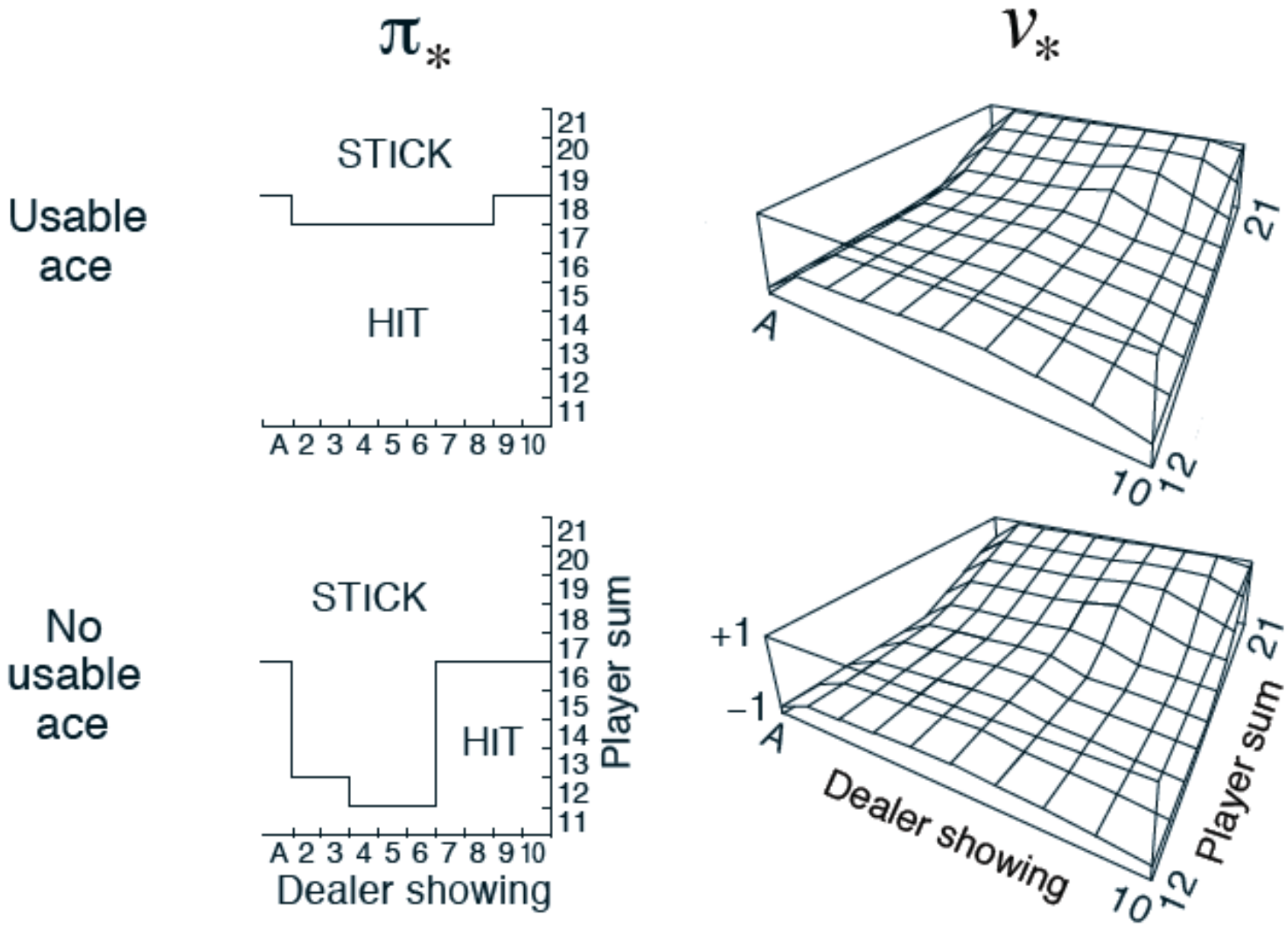
Backup diagram for v_π



策略改进

- 利用MC模拟计算动作的价值函数$q_\pi(s,a)$，就可利用贪心算法对已有策略π进行改进。
- 为了提高$q_\pi(s,a)$的计算效率，可采用试探性开始（exploring starts, ES）：
  - 以一定概率将(s,a)作为初态，其后按策略π进行MC模拟。
- 前面算法的最后一句改成：
  - $Q_\pi(S_t,A_t) \leftarrow \text{average}\big(Returns(S_t,A_t)\big)$
  - $\pi(S_t) \leftarrow \arg\max_a Q_\pi(S_t,a)$
- 也可以采用ϵ-贪心算法（Sutton 5.4）。

例子：21点的最优策略


π_*
v_*
Usable ace
No usable ace
STICK
HIT
STICK
HIT
A 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20 21
Player sum
Dealer showing
+1
−1
A
10
12
21

基于重要性采样的离轨策略

所有的学习控制方法都面临一个困境：希望学到的动作可以使随后的行为是最优的（此时往往只有一个最优行动），但是为了搜索所有的动作（以保证找到最优动作），又需要采取非最优行动。即**探索-利用**的权衡难题。

- 同轨策略（on-policy）：用于生成采样数据序列的策略b和用于实际决策的待评估和改进的策略π是相同的。
- 离轨策略（off-policy）：两者是不同的，即生成的数据“离开”了待优化的策略所决定的决策序列轨迹。
- 原理：不同策略下行动被采纳以及由此产生的轨迹的概率是不同的，但可以通过考虑额外的校正因子从一个策略的结果来推断另一策略的结果。
- 这种思路在很多领域里有重要应用。如：对照试验中的后门调整；分子模拟中的reweighting。

- 从S_t出发，轨迹$S_t, A_t, S_{t+1}, A_{t+1}, \dots S_T$在策略$\pi$下发生的概率为

$$\Pr\{A_t, S_{t+1}, A_{t+1}, \dots S_T | S_t, \pi\}$$
$$= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \dots p(S_T|S_{T-1}, A_{T-1})$$
$$= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)$$

- 因此，同一条轨迹在策略π与策略b下发生的概率之比为

$$\rho_{t:T-1} \doteq \frac{\Pr\{A_t, S_{t+1}, A_{t+1}, \dots S_T | S_t, \pi\}}{\Pr\{A_t, S_{t+1}, A_{t+1}, \dots S_T | S_t, b\}} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)}$$

$$= \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)}$$

- 只与策略π与b有关，与环境的响应机制$p(S_{k+1}|S_k, A_k)$无关。

$$v_b(s) \doteq \mathbb{E}_b[G_t|S_t = s]$$

$$= \sum_{A_t,S_{t+1},A_{t+1},\ldots S_T} \Pr\{A_t, S_{t+1}, A_{t+1}, \ldots S_T|S_t, b\}G_t$$

$$v_\pi(s) = \sum_{A_t,S_{t+1},A_{t+1},\ldots S_T} \Pr\{A_t, S_{t+1}, A_{t+1}, \ldots S_T|S_t, \pi\}G_t$$

$$= \sum_{A_t,S_{t+1},A_{t+1},\ldots S_T} \Pr\{A_t, S_{t+1}, A_{t+1}, \ldots S_T|S_t, b\}\rho_{t:T-1}G_t$$

$$= \mathbb{E}_b[\rho_{t:T-1}G_t|S_t = s]$$

- 因此，根据策略b下产生的蒙特卡罗模拟轨迹，可调整计算得到

$$v_\pi = \frac{\sum_{\text{traj\_}b} \rho_{t:T-1}G_t}{\sum_{\text{traj\_}b} 1}$$

- 由于 $\sum_{A_t,S_{t+1},A_{t+1},\dots S_T} \Pr\{A_t, S_{t+1}, A_{t+1}, \dots S_T | S_t, \pi\} = 1$

我们也可把$v_\pi(s)$写成

$$v_\pi(s) = \frac{\sum_{A_t,S_{t+1},A_{t+1},\dots S_T} \Pr\{A_t, S_{t+1}, A_{t+1}, \dots S_T | S_t, \pi\} G_t}{\sum_{A_t,S_{t+1},A_{t+1},\dots S_T} \Pr\{A_t, S_{t+1}, A_{t+1}, \dots S_T | S_t, \pi\}}$$

$$= \frac{\mathbb{E}_b[\rho_{t:T-1} G_t | S_t = s]}{\mathbb{E}_b[\rho_{t:T-1} | S_t = s]}$$

$$= \frac{\sum_{\text{traj\_}b} \rho_{t:T-1} G_t}{\sum_{\text{traj\_}b} \rho_{t:T-1}}$$

$\Pr\{A_t, S_{t+1}, \dots S_T | S_t, \pi\} = \Pr\{A_t, S_{t+1}, \dots S_T | S_t, b\} \rho_{t:T-1}$

- 在轨迹数目有限时，前一页的方法是无偏差的，但方差比较大；本页的方法是有偏差的，但方差比较小。

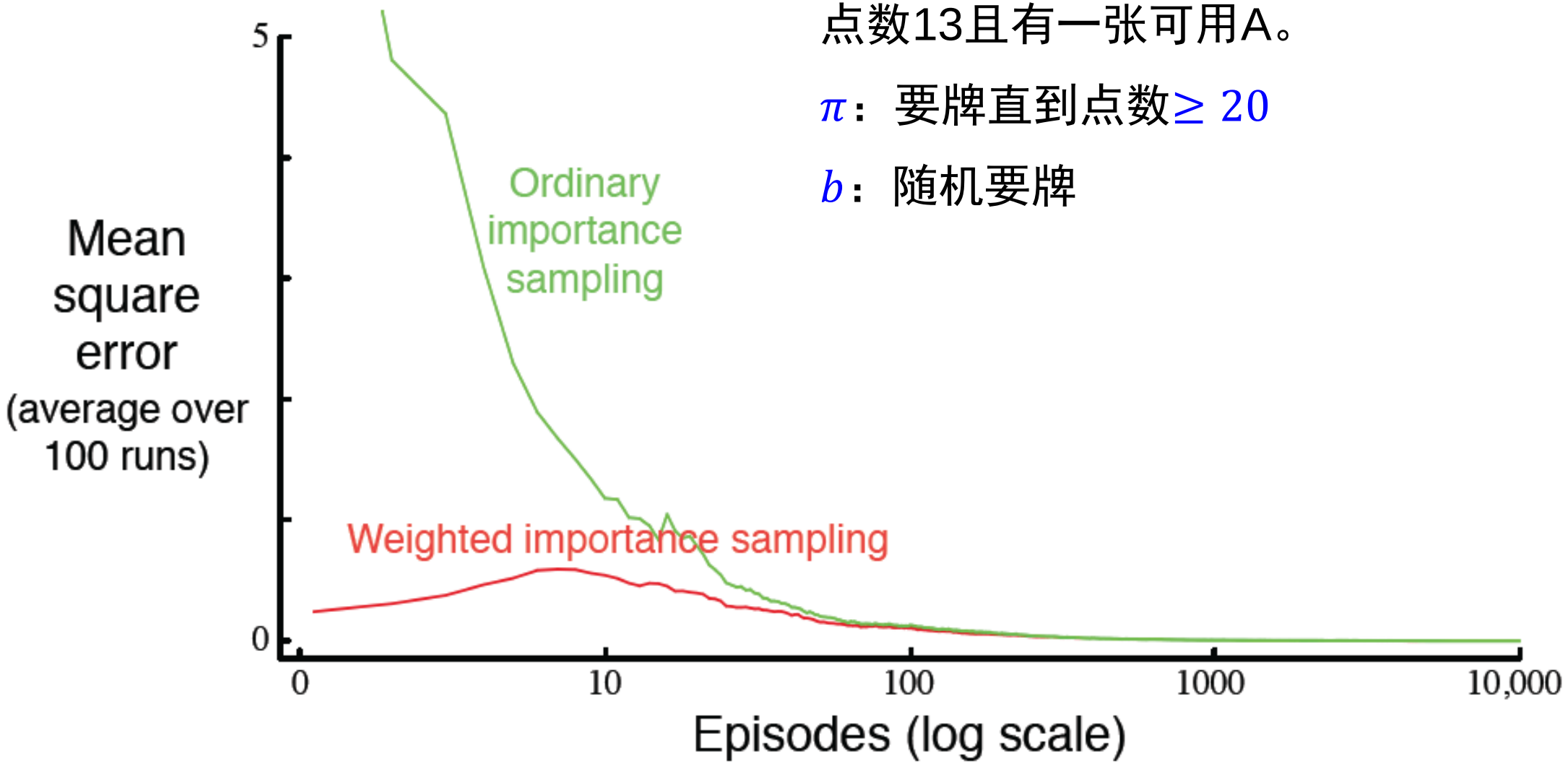
- 实际应用中，后者的效果一般较好。

例子：21点

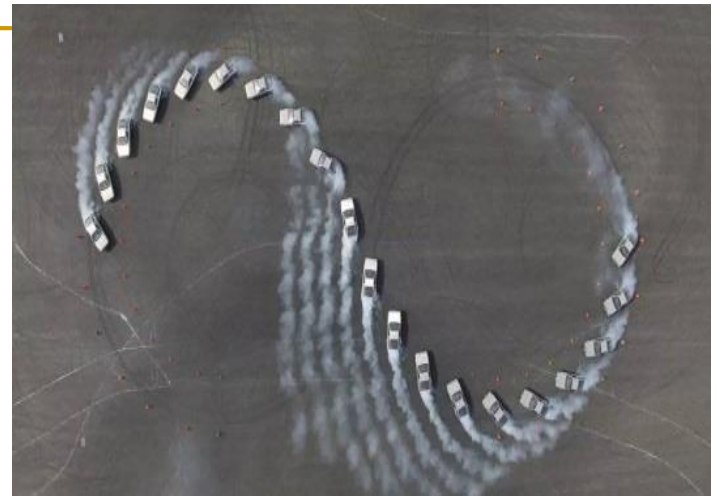
待评估状态：庄家明牌2，玩家点数13且有一张可用A。

π：要牌直到点数≥ 20

b：随机要牌



练习/思考：赛车控制



- 起点位置随机速度为零。
- 动作：两个方向的速度改变$1,\ 0,\ -1$。
- 碰到边界后回到随机起点。

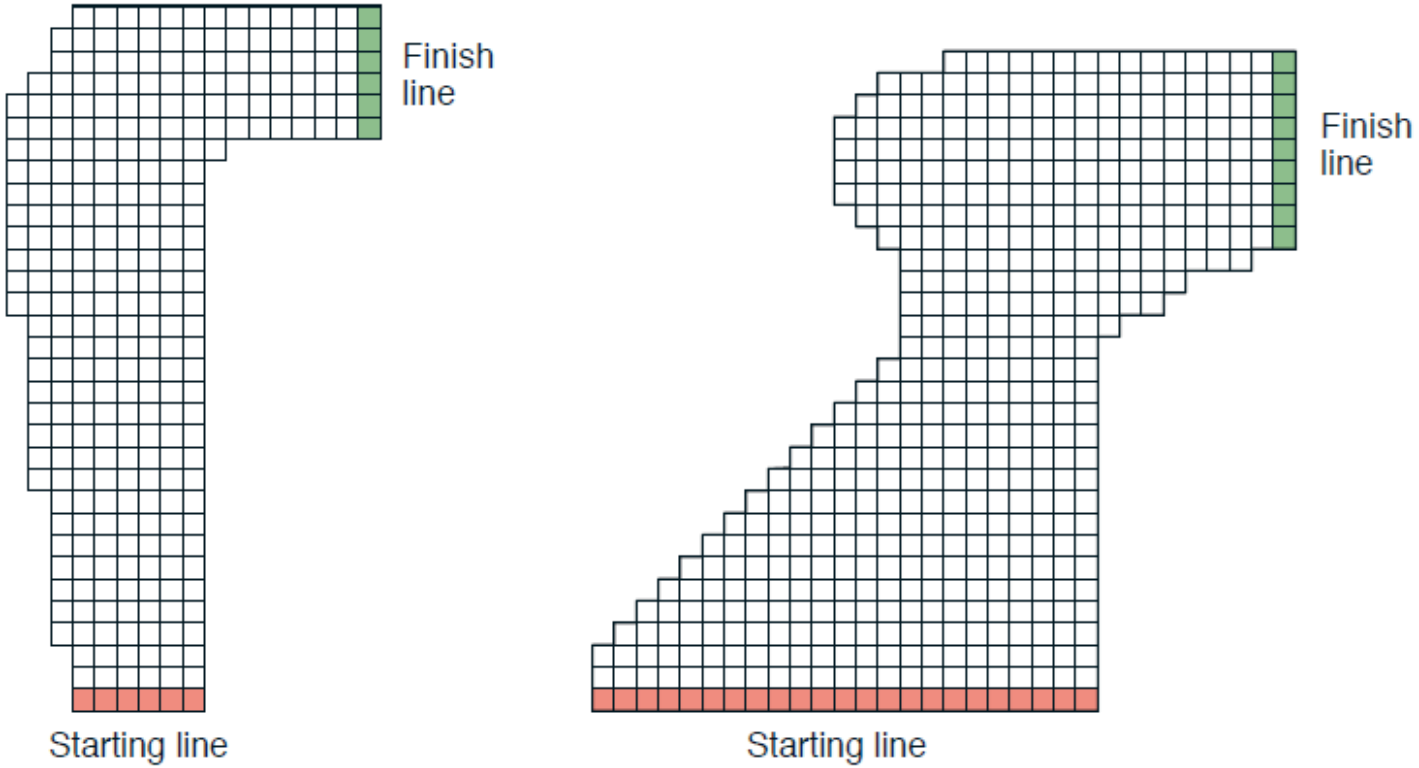
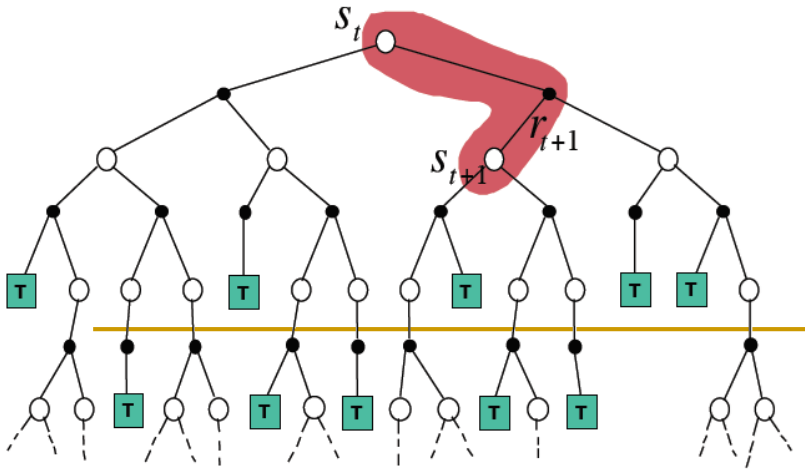


Figure 5.5: A couple of right turns for the racetrack task.

3. 时序差分算法
（Temporal-Difference）

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



时序差分方法

- 结合了动态规划与蒙特卡罗方法的想法。

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

- 蒙特卡罗方法采用轨迹抽样的方法来估计第一个式子里的期望值。
- 动态规划利用第三个式子，能严格处理期望值（利用概率分布信息），但用$V(S_{t+1})$来作为右边$v_\pi(S_{t+1})$的估计。
- 时序差分方法也利用第三个式子，用抽样的方法来估计期望值，同时用$V(S_{t+1})$来作为$v_\pi(S_{t+1})$的估计。

- 在蒙特卡罗模拟的在线学习中，基于$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$，得

$$V(S_t) \leftarrow V(S_t) + \eta[G_t - V(S_t)]$$ （η：学习率，Sutton用α）

- 在时序差分算法中，基于$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$，得到

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

这被称为单步TD或TD(0)。

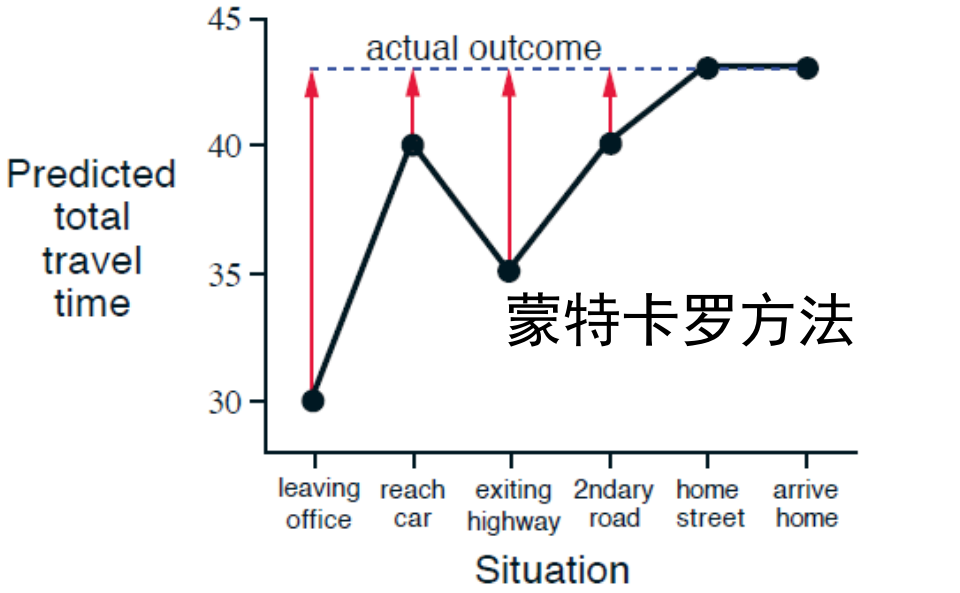
- 其中方括号内的结果

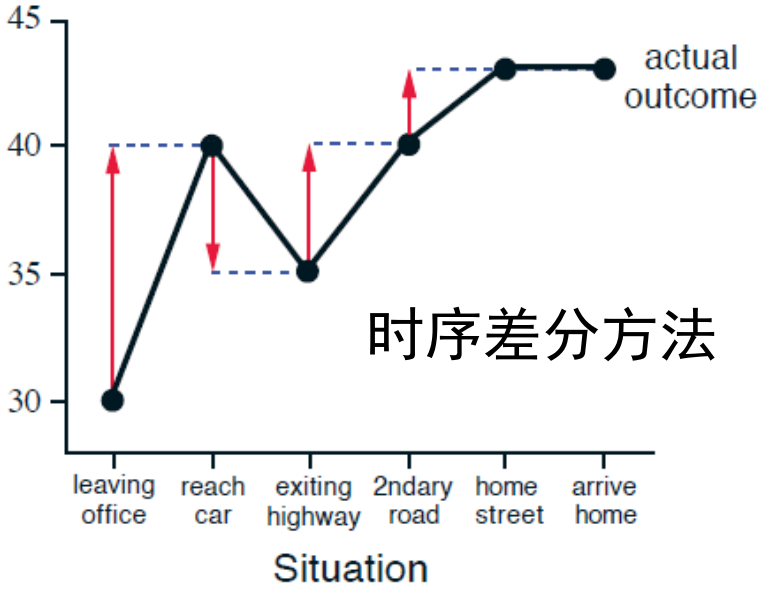
$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

被称为TD误差，描述了状态S_t的价值估计$V(S_t)$与更好的估计$R_{t+1} + \gamma V(S_{t+1})$之间的差异。

例子：开车回家...

状态	时间	估计剩余时间（分钟）	估计总时间（分钟）
离开办公室	6:00	30	30
到车旁，开始下雨	6:05	35	40
下高速	6:20	15	35
在路上堵在卡车后面	6:30	10	40
开到居住的街道	6:40	3	43
到家	6:43	0	43



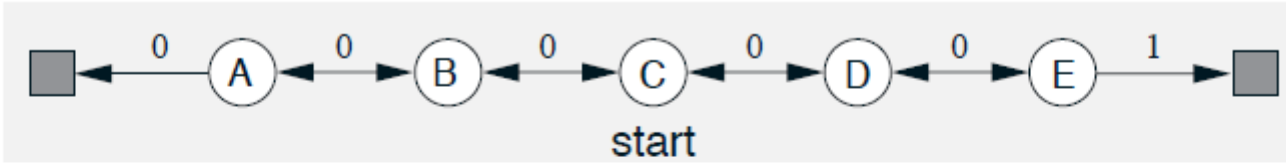


不需要等模拟结束，就可更新结果

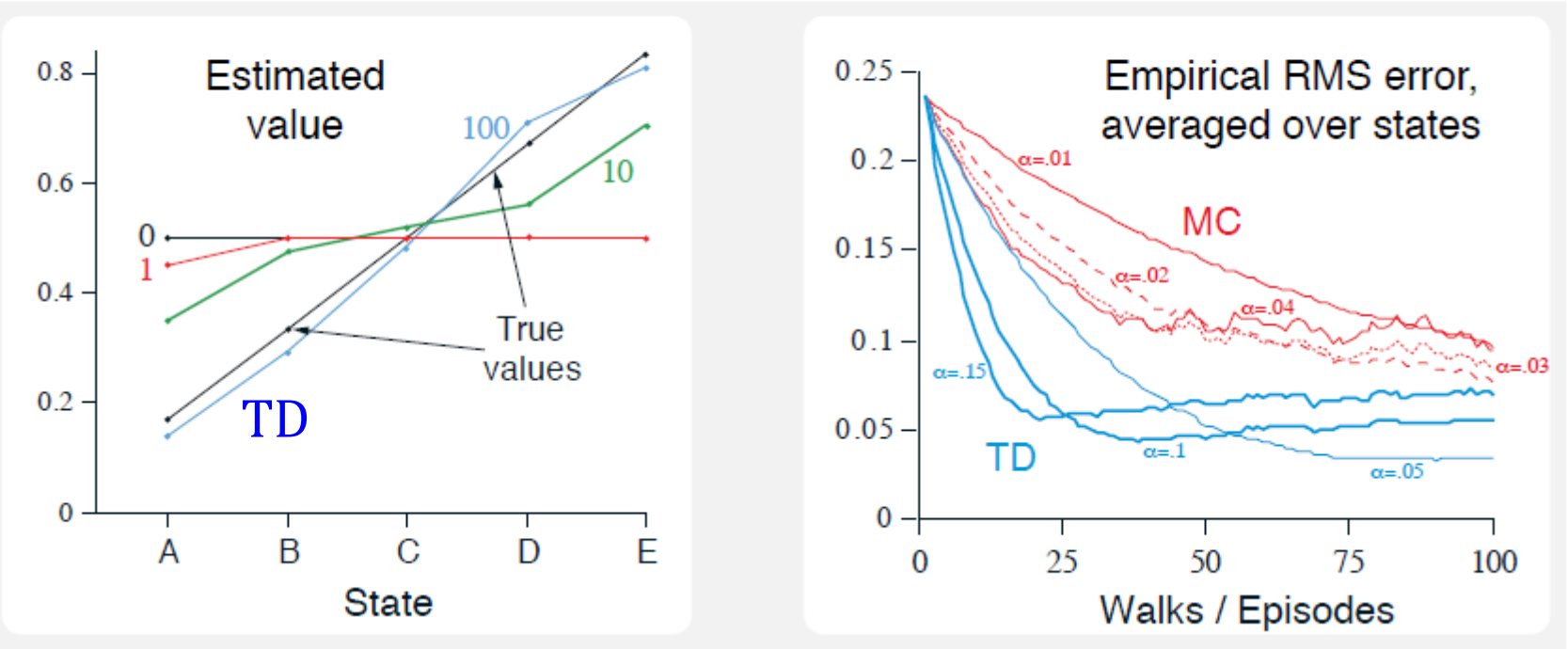
时序差分算法的优势

- 与动态规划相比，不需知道显式的$p(s', r|s, a)$
- 与蒙特卡罗方法相比，不需等一条轨迹结束就能更新$V(S_t)$
- 例子：

马尔科夫回报过程

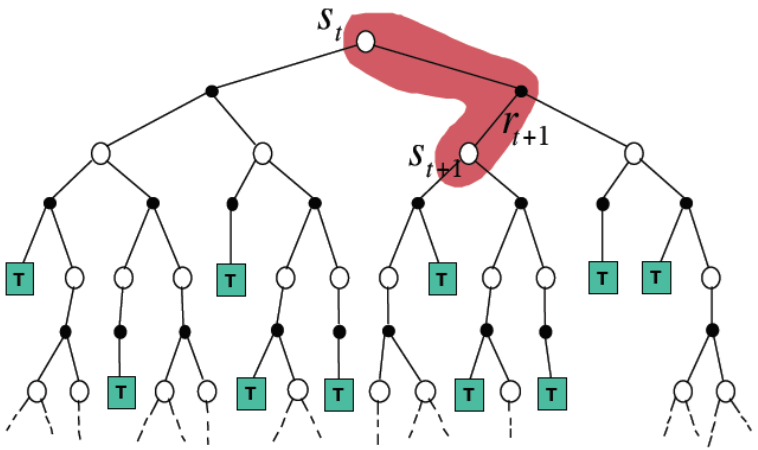


$\eta = 0.1$

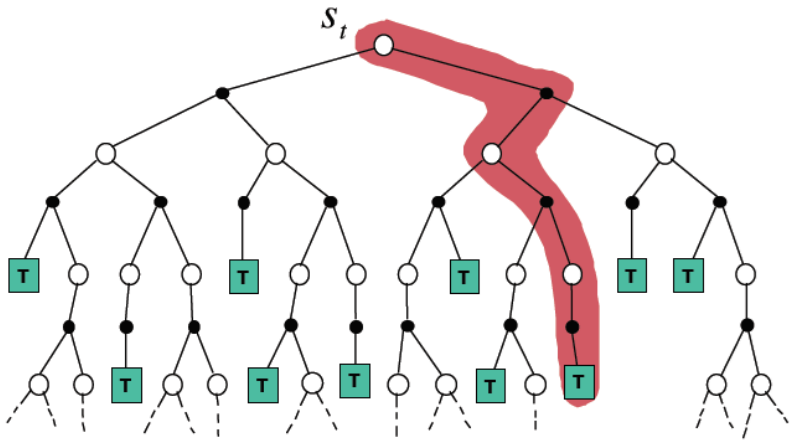


三种算法的不同

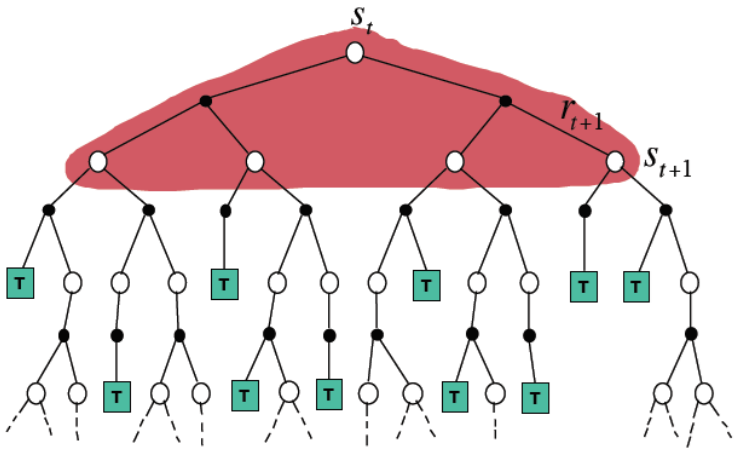
$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



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



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

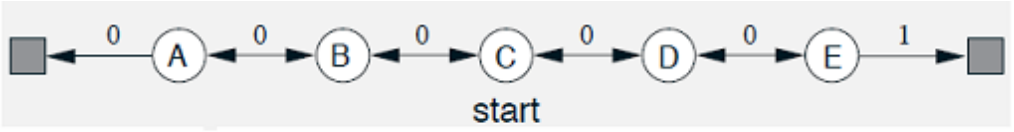


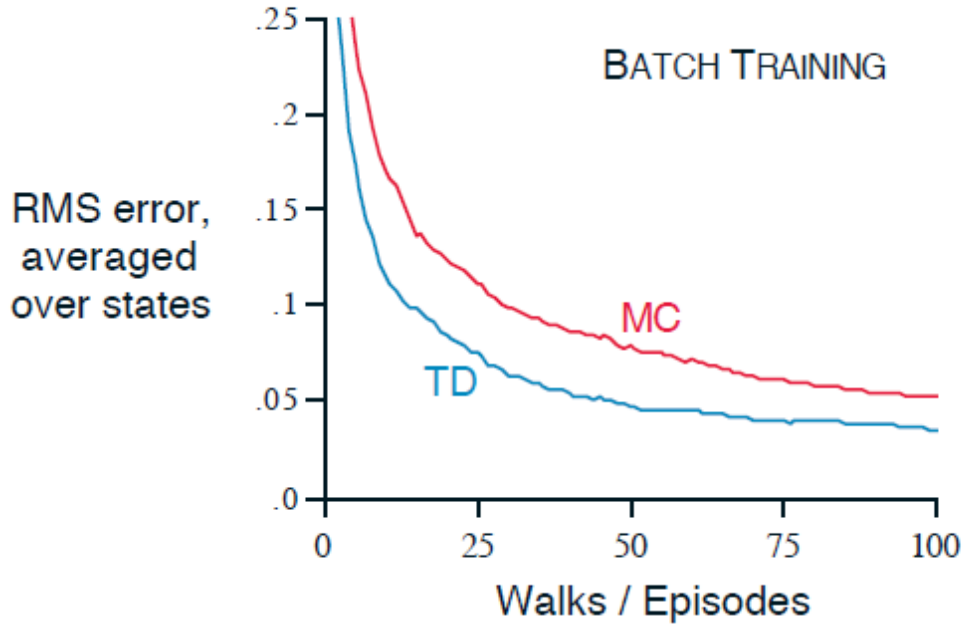
$$\alpha \equiv \eta$$

批量更新（batch updating）与最优性

- 在数据有限时，通常是反复地利用这些数据进行训练，直至迭代收敛。此时 只要η足够小，就能确定地收敛到与η无关的唯一结果。

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



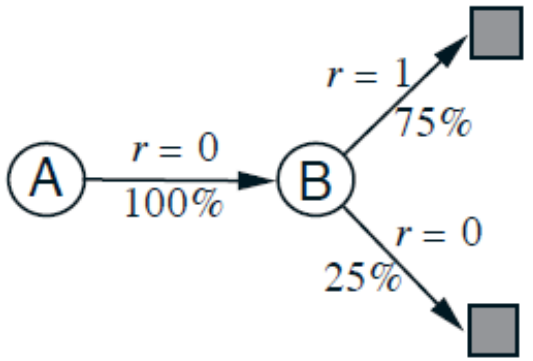


Example 6.4: You are the Predictor

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

$$V(B) = \frac{3}{4}$$

$$V(A) = ?$$



Sarsa：同轨策略下的时序差分控制

- 动作价值函数利用下式来计算：

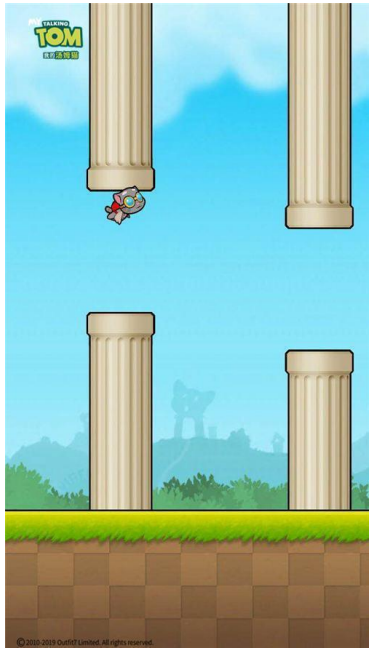
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

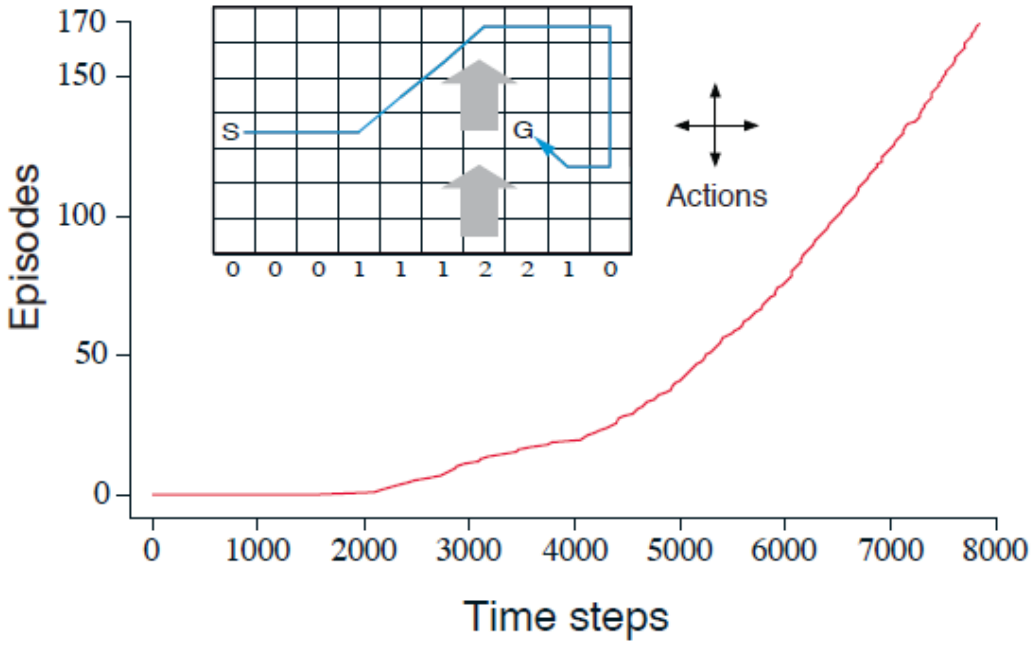
  - 这里利用了五元组$(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$，因此被称为Sarsa

- 策略改进可采用ϵ-贪心算法。

- 例子：

有风的网格世界。

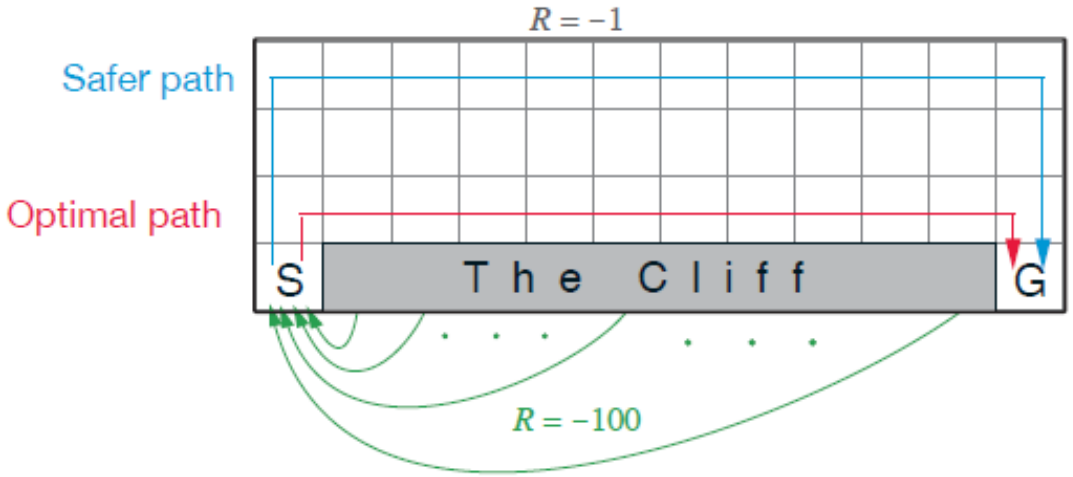




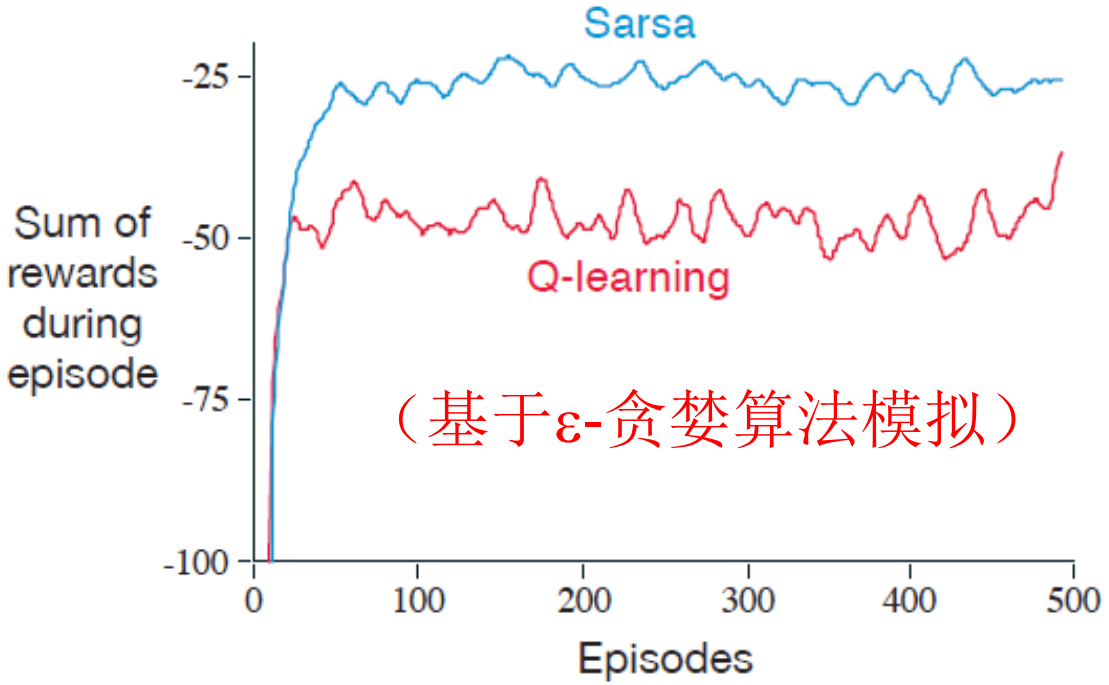
Q-学习：离轨策略下的时序差分控制

- 贝尔曼最优方程：$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$
- Q-学习（Q-learning）从任一策略的结果中估计q_*：

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \eta\left[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\right]$$



例子：有悬崖的网格世界

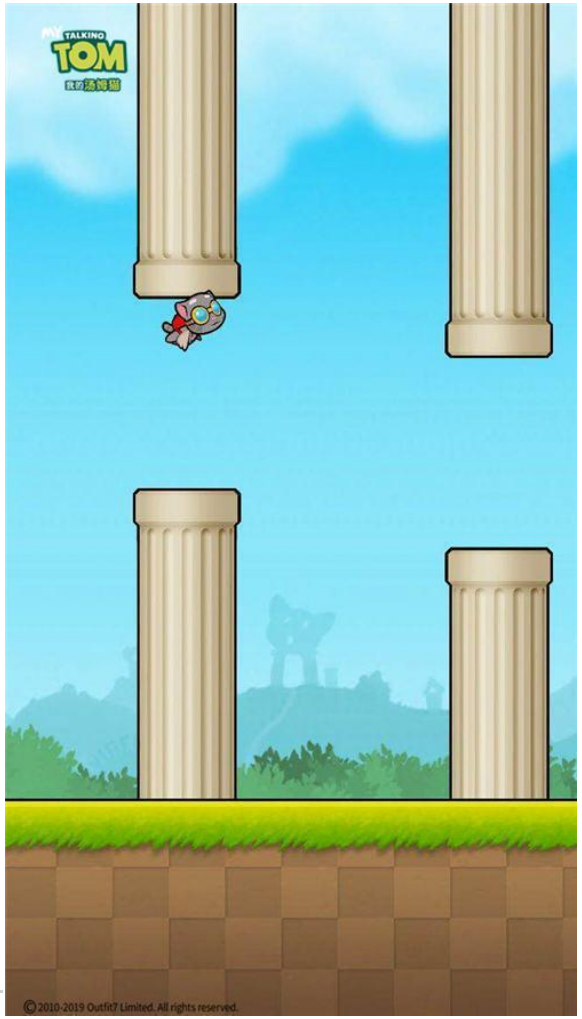


（基于ε-贪婪算法模拟）

例子：飞翔的汤姆猫...

- Q-学习：$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$

- https://enhuiz.github.io/flappybird-ql/



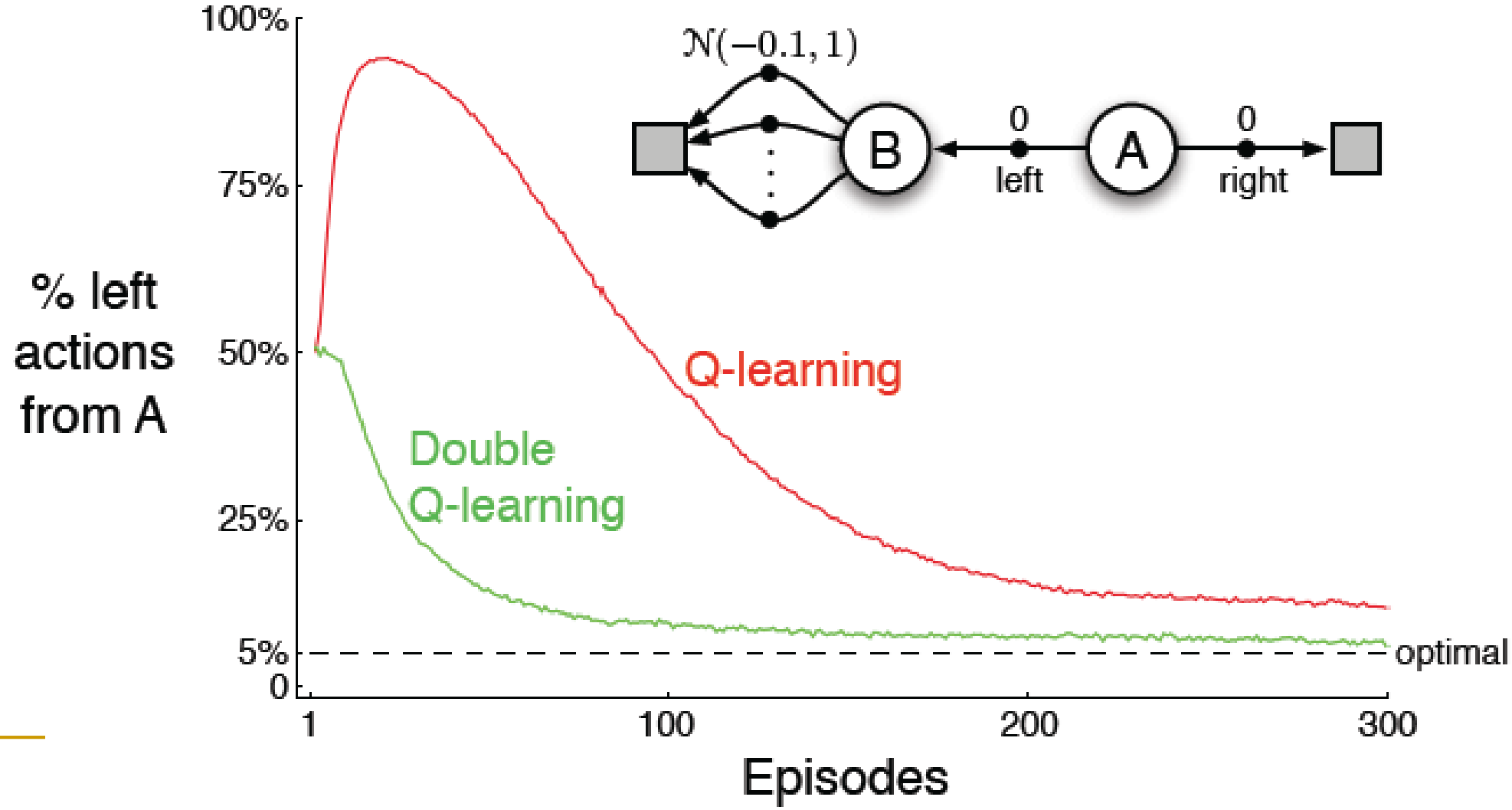
最大化偏差和双Q学习

$\max_a Q(S_{t+1}, a)$

- 为了求得最优策略，通常采用包含最大化操作的贪婪算法。这有可能导致严重的正向偏差，称为最大化偏差（maximization bias）
- 例如：对于状态 S，有很多动作a可以选择；而每个真实的$q(S,a)$值都为0；由于误差的存在，估计的$Q(S,a)$ 要么大于0，要么小于0；在最大化操作后，就得到了一个正值偏差。
- 解决方法：最大化选择与Q估值使用分开的独立数据，如双Q-学习：

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \eta \left[R_{t+1} + \gamma Q_2 \left(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a) \right) - Q_1(S_t, A_t) \right]$$

例子：


100%
75%
50%
25%
5%
0
% left actions from A
$\mathcal{N}(-0.1, 1)$
B
0
left
A
0
right
Q-learning
Double Q-learning
optimal
1
100
200
300
Episodes

小结

- 动态规划可用于模型完全已知的强化学习问题。
- 可通过迭代法求解策略π的价值函数

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

- 策略改进定理：对于确定性的策略π和π'，如果对任意状态s有

$$q_\pi(s,\pi'(s)) \geq v_\pi(s)$$

则π'比π更好，或至少与π一样好：$v_{\pi'}(s) \geq v_\pi(s)$

- 如果贪心策略与原策略一样好，则它们都是最佳策略。
- 策略迭代：策略评估（E）与策略改进（I）不断交替。
- 如果只迭代一次计算$v_{\pi_k}(s)$，则得到价值迭代算法：

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

  - 可看成是贝尔曼最优方程的直接迭代算法。

- 蒙特卡罗方法不需要关于环境的完整信息。用采样序列的性质平均值来估计（代替）其理论期望值
- 同轨策略vs.离轨策略：
  - 同轨策略（on-policy）：用于生成采样数据序列的策略b和用于实际决策的待评估和改进的策略π是相同的。
  - 离轨策略（off-policy）：两者是不同的，即生成的数据“离开”了待优化的策略所决定的决策序列轨迹。
- 时序差分方法结合了动态规划与蒙特卡罗方法的想法

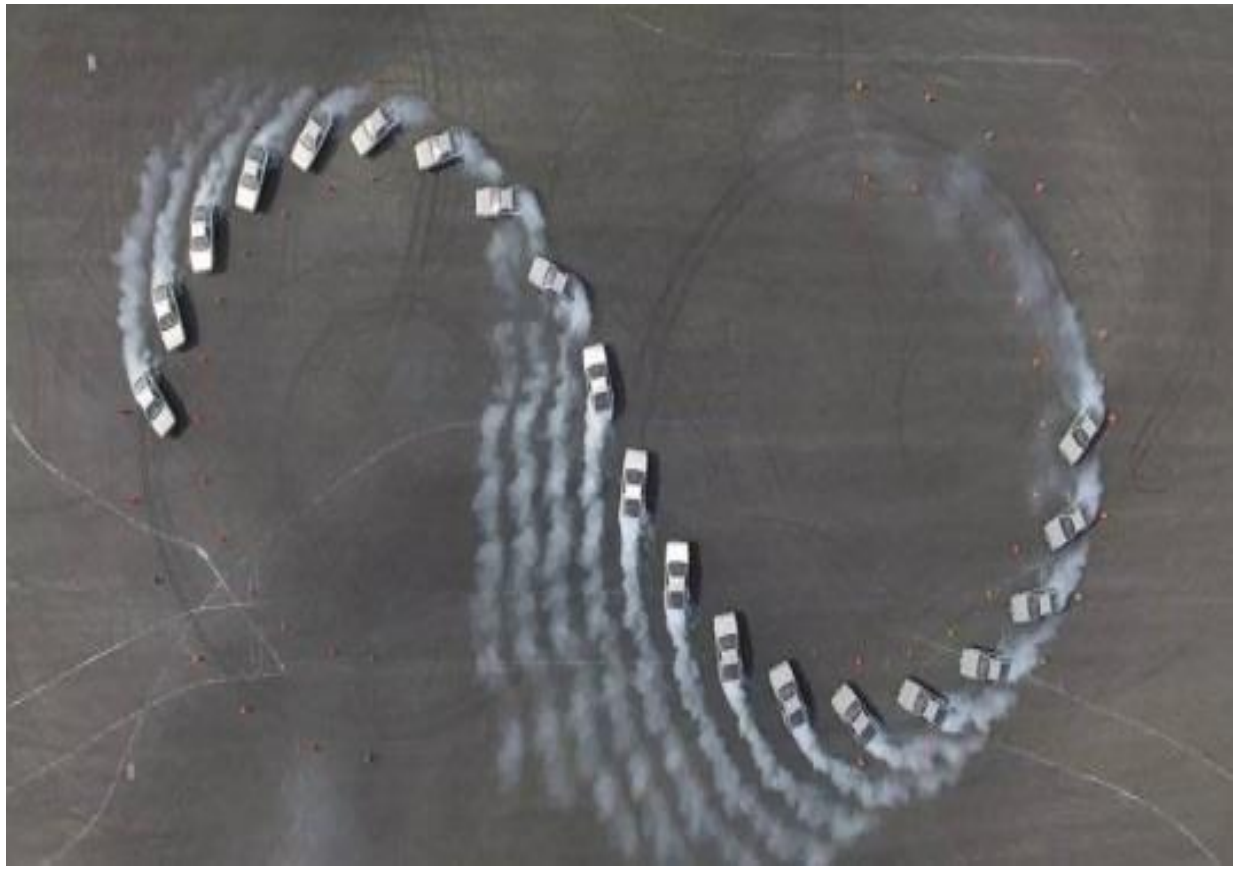
$$V(S_t) \leftarrow V(S_t) + \eta[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

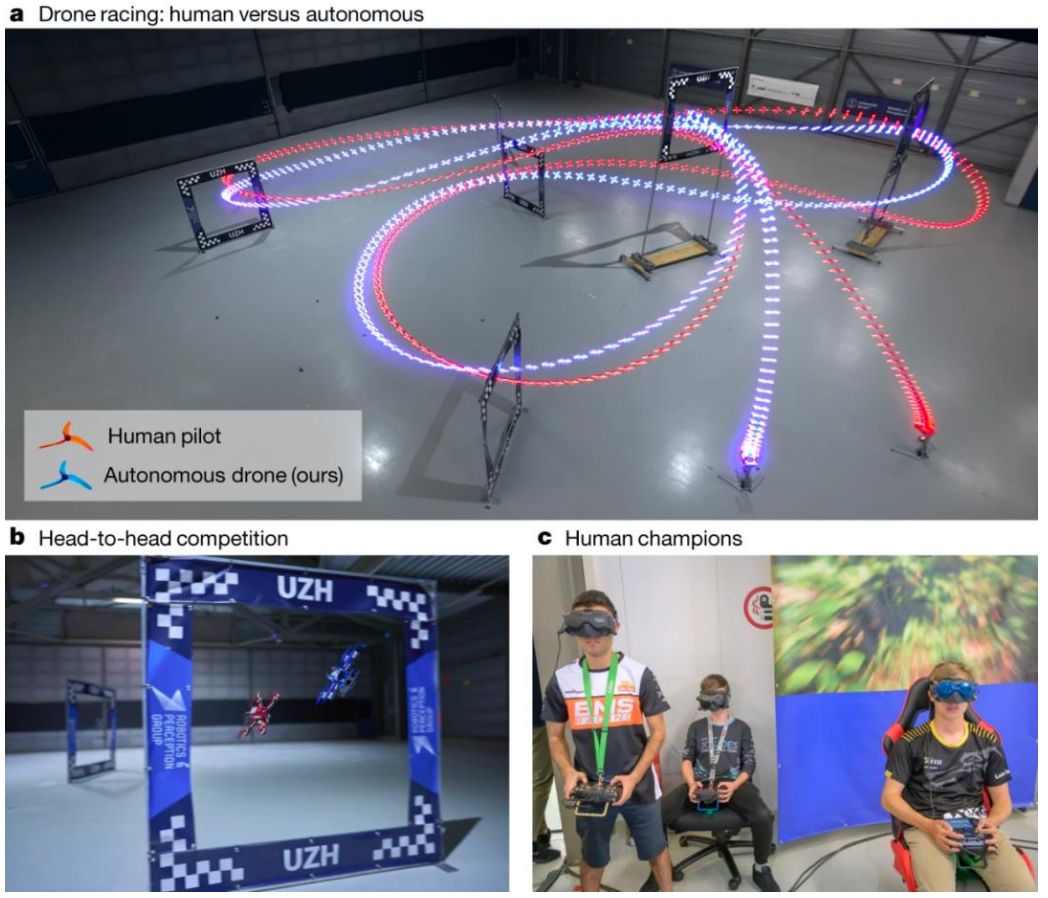
- 利用轨迹采样，但不需等一条轨迹结束就能更新。
- Q-学习

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

- Reference:
  - Sutton 4-6；
- 扩展阅读：
  - https://enhuiz.github.io/flappybird-ql/
    Flappy Bird Q-learning.mht　　flappybird-ql-master.zip
  - https://baijia.baidu.com/s?id=1595349081674644741
    用Python入门不明觉厉的马尔可夫链蒙特卡罗（附案例代码） -百家号.mht
  - https://baijia.baidu.com/s?id=1597978859962737001
    通过 Q-learning 深入理解强化学习
  - https://www.huxiu.com/article/416426.html
    世界上最难的“沙雕”游戏被AI攻破了.mht

- https://www.jiqizhixin.com/articles/2023-08-31-4
  刺激，无人机竞速超越顶级人类玩家，强化学习再登Nature封面.mhtml
- https://wwww.huxiu.com/article/332683.html
  斯坦福爆改了一辆DeLorean，自动驾驶还能漂移.mhtml





谢谢大家！