涉及知识点：
马尔可夫决策过程、基于动态规划的强化学习、
基于模型的强化学习

# 强化学习基础2

田政

https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021

# MDP的平稳性

一个MDP是平稳的，表示五元组中的任意对象都不会随时间变化而变化

- MDP可以由一个五元组表示 $(S, A, \{P_{sa}\}, \gamma, R)$
  - $S$是状态的集合
    - 比如，迷宫中的位置，Atari游戏中的当前屏幕显示
  - $A$是动作的集合
    - 比如，向N、E、S、W移动，手柄操纵杆方向和按钮
  - $P_{sa}$是状态转移概率
    - 对每个状态$s \in S$和动作$a \in A$，$P_{sa}$是下一个状态在S中的概率分布
  - $\gamma \in [0,1]$是对未来奖励的折扣因子
  - $R: S \times A \longmapsto \mathbb{R}$ 是奖励函数
    - 有时奖励只和状态相关



此处请输出出处和引用

# MDP的随机性

通常MDP的随机性来自于两层因素：

- MDP的动态如下所示：
  - 从状态$s_0$开始
  - 智能体选择某个动作$a_0 \in A$
  - 智能体得到奖励$R(s_0, a_0)$
  - MDP随机转移到下一个状态$s_1 \sim P_{s_0 a_0}$
  - 这个过程不断进行

$$s_0 \xrightarrow{a_0, R(s_0, a_0)} s_1 \xrightarrow{a_1, R(s_1, a_1)} s_2 \xrightarrow{a_2, R(s_2, a_2)} s_3 \quad \cdots$$

  - 直到终止状态$s_T$出现为止，或者无止尽地进行下去

# Bellman等式

# 状态值函数

- The value function $v(s)$ gives the long-term value of state $s$

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

- It can be defined recursively:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, \pi\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\right] \\ &= \sum_a \pi(a \mid s) \sum_r \sum_{s'} p(r, s' \mid s, a) \left(r + \gamma v_\pi(s')\right) \end{aligned}$$

- The final step writes out the expectation explicitly

# 动作值函数

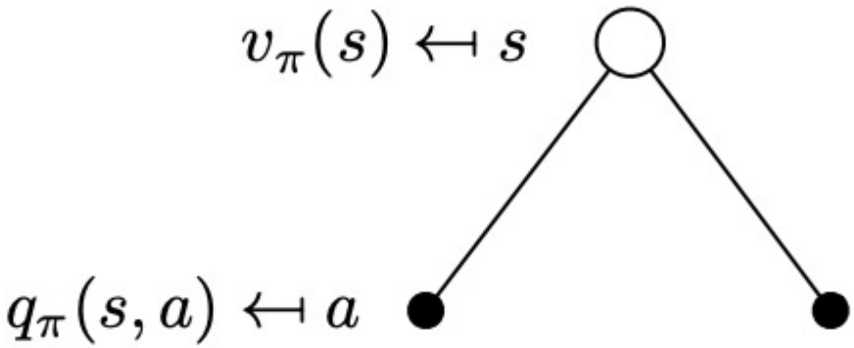
- We can define state-action values

$$q_\pi(s,a) = \mathbb{E}\left[G_t \mid S_t = s, A_t = a, \pi\right]$$

- This implies

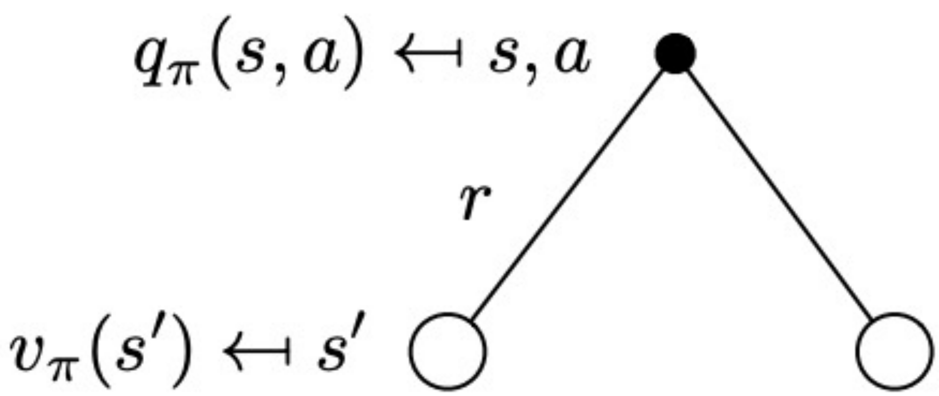
$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \\
&= \sum_r \sum_{s'} p(r, s' \mid s, a) \left( r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right)
\end{aligned}$$

# 状态值函数(1)



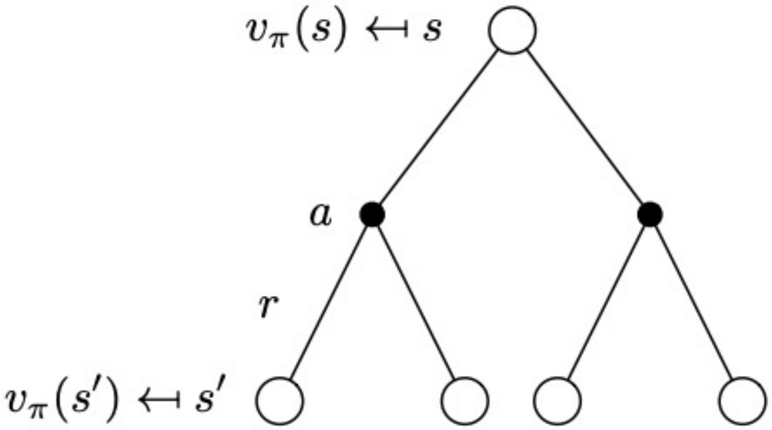

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

# 动作值函数(1)



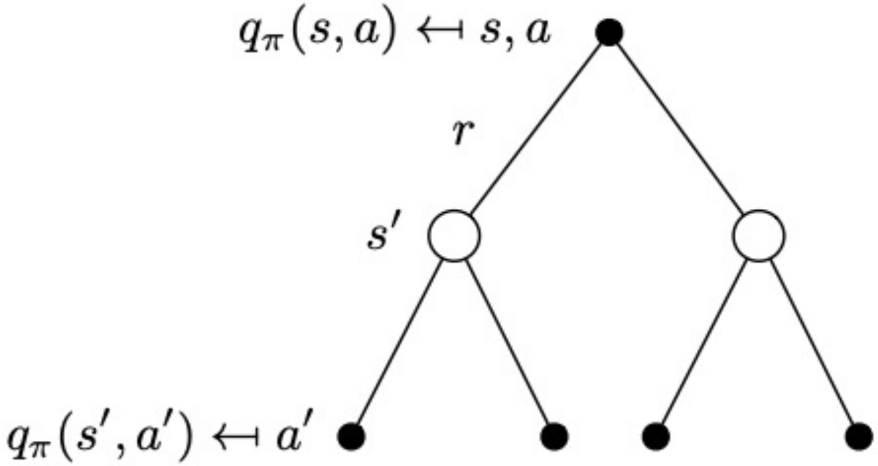

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

# 状态值函数(2)




$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

# 动作值函数(2)




$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

# Bellman期望公式

**Theorem (Bellman Expectation Equations)**

*Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, for any policy $\pi$, the value functions obey the following expectation equations:*

$$v_\pi(s) = \sum_a \pi(s,a) \left[ r(s,a) + \gamma \sum_{s'} p(s'|a,s) v_\pi(s') \right] \quad (5)$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|a,s) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a') \quad (6)$$

# Bellman最优公式

**Theorem (Bellman Optimality Equations)**

*Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, the optimal value functions obey the following expectation equations:*

$$v^*(s) = \max_a \left[ r(s,a) + \gamma \sum_{s'} p(s'|a,s) v^*(s') \right] \tag{7}$$

$$q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|a,s) \max_{a' \in \mathcal{A}} q^*(s',a') \tag{8}$$

There can be no policy with a higher value than $v_*(s) = \max_\pi v_\pi(s), \forall s$

# 最优值函数

**Definition (Optimal value functions)**

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 function

# 最优策略

Define a partial ordering over policies

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

**Theorem (Optimal Policies)**

*For any Markov decision process*

- *There exists an optimal policy* $\pi^*$ *that is better than or equal to all other policies,* $\pi^* \geq \pi, \forall \pi$
  *(There can be more than one such optimal policy.)*
- *All optimal policies achieve the optimal value function,* $v^{\pi^*}(s) = v^*(s)$
- *All optimal policies achieve the optimal action-value function,* $q^{\pi^*}(s,a) = q^*(s,a)$

# 从最优值函数到最优策略

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

$$\pi^*(s, a) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q^*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

Observations:

- If we know q*(s, a), we immediately have the optimal policy ?
- There is always a deterministic optimal policy for any MDP?
- There can be multiple optimal policies ?
- If multiple actions maximize q*(s,.), we can also just pick any of these(including stochastically) ?

# 基于动态规划的强化学习

# MDP目标和策略

- 目标：选择能够最大化累积奖励期望的动作

$$\mathbb{E}[R(s_0)+\gamma R(s_1)+\gamma^2 R(s_2)+\cdots]$$

  - $\gamma \in [0,1]$是未来奖励的折扣因子，使得和未来奖励相比起来智能体更重视即时奖励
    - 以金融为例，今天的\$1比明天的\$1更有价值

- 给定一个特定的策略 $\pi(s): S \rightarrow A$
  - 即，在状态 $s$ 下采取动作 $a=\pi(s)$

- 给策略$\pi$定义价值函数

$$V^{\pi}(s)= \mathbb{E}[R(s_0)+\gamma R(s_1)+\gamma^2 R(s_2)+\cdots|s_0=s,\pi]$$

  - 即，给定起始状态和根据策略$\pi$采取动作时的累积奖励期望

# 价值函数的Bellman等式

- 给策略$\pi$定义价值函数

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}[R(s_0) + \underbrace{\gamma R(s_1) + \gamma^2 R(s_2) + \cdots}_{\gamma V^{\pi}(s_1)} | s_0 = s, \pi] \\
&= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')
\end{aligned}
$$

Bellman等式

立即奖励

状态转移

时间折扣

下一个状态的价值

# 最优价值函数

- 对状态$s$来说的最优价值函数是所有策略可获得的最大可能折扣奖励的和

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

- 最优价值函数的Bellman等式

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- 最优策略

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- 对状态s和策略$\pi$

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

# 价值迭代和策略迭代

- 价值函数和策略相关

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

- 可以对最优价值函数和最优策略执行迭代更新
  - 价值迭代
  - 策略迭代

# 策略迭代

- 对于一个动作空间和状态空间有限的MDP

$$|S| < \infty, |A| < \infty$$

- 策略迭代过程

1. 随机初始化策略 $\pi$
2. 重复以下过程直到收敛{

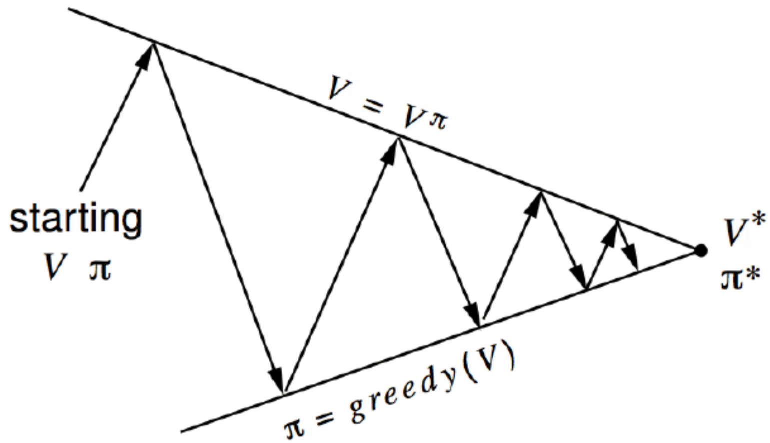
   a) 评估当前策略的价值 $V^{\pi} \quad = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$

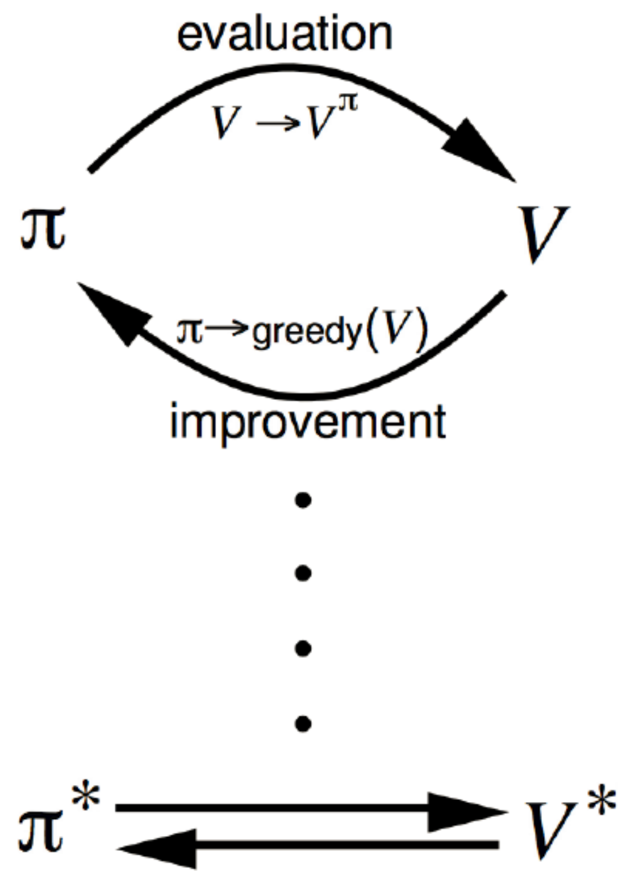
   b) 对每个状态，更新

$$\pi(s) = arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

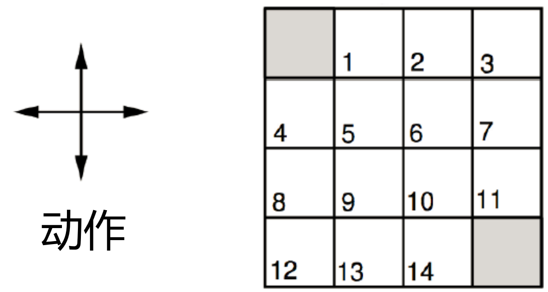
# 策略迭代




- 策略评估
  - 估计$V^{\pi}$
  - 迭代的评估策略
- 策略改进
  - 生成 $\pi' \geq \pi$
  - 贪心策略改进

# 举例：策略评估




- 非折扣MDP（$\gamma = 1$）
- 非终止状态：1, 2, …,14
- 两个终止状态（灰色方格）
- 如果动作指向所有方格以外，则这一步不动
- 奖励均为-1，直到到达终止状态
- 智能体的策略为均匀随机策略

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

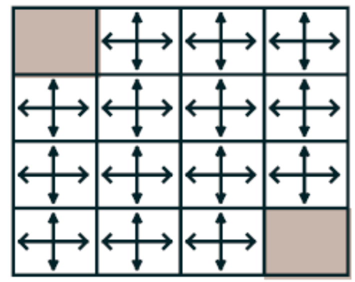
# 举例：策略评估

随机策略的 $V_k$

$V_k$对应的贪心策略

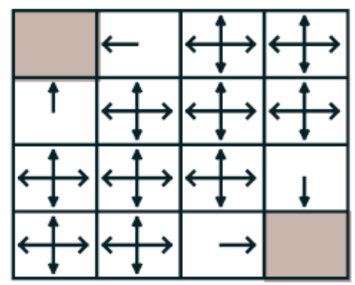
K=0

| | | | |
|---|---|---|---|
| 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 |



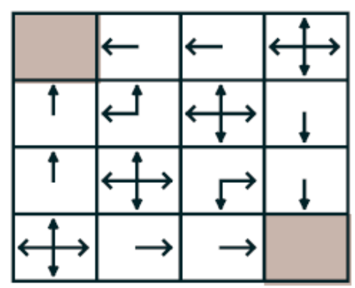
K=1

| | | | |
|---|---|---|---|
| 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 |

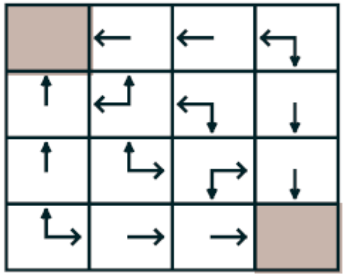


# 举例：策略评估

随机策略的 $V_k$ | $V_k$对应的贪心策略

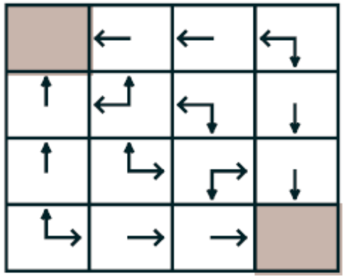
K=3

| | | | |
|---|---|---|---|
| 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 |



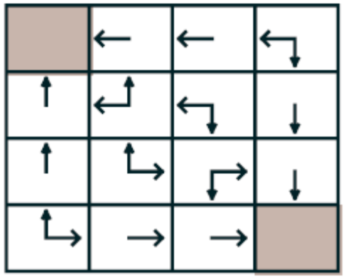
K=10

| | | | |
|---|---|---|---|
| 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=∞

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



$V := V^{\pi}$

最优策略

# 策略迭代

- 对于一个动作空间和状态空间有限的MDP

$$|S| < \infty, |A| < \infty$$

- 策略迭代过程

1. 随机初始化策略 $\pi$
2. 重复以下过程直到收敛{

   a) 评估当前策略的价值 $V^{\pi} = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$

   b) 对每个状态，更新

$$\pi(s) = arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

更新价值函数会很耗时！

# 价值迭代

- 对于一个动作空间和状态空间有限的MDP

$$|S| < \infty, |A| < \infty$$

- 价值迭代过程

1. 对每个状态$s$，初始化 $V(s) = 0$
2. 重复以下过程直到收敛 {

    对每个状态，更新

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$$

}

注意：在以上的计算中没有明确的策略



价值迭代收敛性证明：https://towardsdatascience.com/mathematical-analysis-of-reinforcement-learning-bellman-equation-ac9f0954e19f

# 同步 vs. 异步价值迭代

- 同步的价值迭代会储存两份价值函数的拷贝
  1. 对$S$中的所有状态s

$$V_{new}(s) \leftarrow \max_{a \in A}\left(R(s) + \gamma \sum_{s' \in S} P_{sa}(s') V_{old}(s')\right)$$

  2. 更新 $V_{old}(s) \leftarrow V_{new}(s)$

- 异步价值迭代只储存一份价值函数
  1. 对$S$中的所有状态s

$$V(s) \leftarrow \max_{a \in A}\left(R(s) + \gamma \sum_{s' \in S} P_{sa}(s') V(s')\right)$$

# 价值迭代例子：最短路径

| | | | |
|---|---|---|---|
| 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 |

$V_7$

# 价值迭代 vs. 策略迭代

价值迭代

1. 对每个状态s，初始化 $V(s) = 0$
2. 重复以下过程直到收敛 {
   对每个状态，更新

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$$

}

策略迭代

1. 随机初始化策略 $\pi$
2. 重复以下过程直到收敛 {
   a) 让 $V := V^{\pi}$
   b) 对每个状态，更新

$$\pi(s) = arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

备注：

1. 价值迭代是贪心更新法
2. 策略迭代中，用Bellman等式更新价值函数代价很大
3. 对于空间较小的MDP，策略迭代通常很快收敛
4. 对于空间较大的MDP，价值迭代更实用（效率更高）
5. 如果没有状态转移循环，最好使用价值迭代

# 基于模型的强化学习

# 学习一个MDP模型

- 目前我们关注在给出一个已知MDP模型后：（也就是说，状态转移 $P_{sa}(s')$ 和奖励函数 $R(s)$ 明确给定后）
  - 计算最优价值函数
  - 学习最优策略
- 在实际问题中，状态转移和奖励函数一般不是明确给出的
  - 比如，我们只看到了一些episodes

$$\text{Episode1：}\quad s_0^{(1)} \xrightarrow{a_0^{(1)}, R(s_0)^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}, R(s_1)^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}, R(s_2)^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$$

$$\text{Episode2：}\quad s_0^{(2)} \xrightarrow{a_0^{(2)}, R(s_0)^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}, R(s_1)^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}, R(s_2)^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$$

# 学习一个MDP模型

Episode1：$s_0^{(1)} \xrightarrow{a_0^{(1)}, R(s_0)^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}, R(s_1)^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}, R(s_2)^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode2：$s_0^{(2)} \xrightarrow{a_0^{(2)}, R(s_0)^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}, R(s_1)^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}, R(s_2)^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

$\vdots$ $\qquad\qquad\qquad$ $\vdots$

- 从“经验”中学习一个MDP模型
  - 学习状态转移概率$P_{sa}(s')$

$$P_{sa}(s') = \frac{\text{在}s\text{下采取动作}a\text{并转移到}s'\text{的次数}}{\text{在}s\text{下采取动作}a\text{的次数}}$$

  - 学习奖励函数$R(s)$，也就是立即奖赏期望

$$R(s) = average\{R(s)^{(i)}\}$$

# 学习模型&优化策略

- 算法

1. 随机初始化策略$\pi$

2. 重复以下过程直到收敛 {

   a) 在MDP中执行$\pi$，收集经验数据

   b) 使用MDP中的累积经验更新对$P_{sa}$和$R$的估计

   c) 利用对$P_{sa}$和$R$的估计执行价值迭代，得到新的估计价值函数$V$

   d) 根据$V$更新策略$\pi$为贪心策略

   }

# 实例：AB状态问题

Two states $A, B$; no discounting; 8 episodes of experience

$$\text{A}, 0, \text{B}, 0$$
$$\text{B}, 1$$
$$\text{B}, 1$$
$$\text{B}, 1$$
$$\text{B}, 1$$
$$\text{B}, 1$$
$$\text{B}, 1$$
$$\text{B}, 0$$

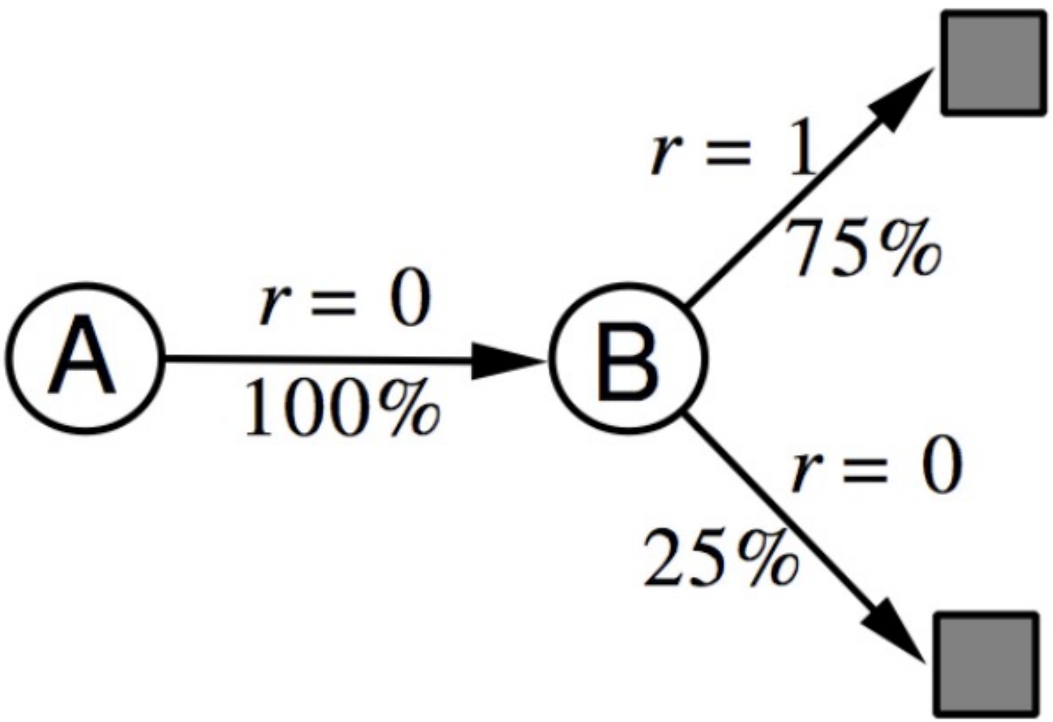
?

We have constructed a table lookup model from the experience

# 实例：AB状态问题

Two states $A, B$; no discounting; 8 episodes of experience

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




We have constructed a table lookup model from the experience

# 学习一个MDP模型

- 在实际问题中，状态转移和奖励函数一般不是明确给出的
  - 比如，我们只看到了一些episodes

Episode1： $s_0^{(1)} \xrightarrow{a_0^{(1)},R(s_0)^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)},R(s_1)^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)},R(s_2)^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode2： $s_0^{(2)} \xrightarrow{a_0^{(2)},R(s_0)^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)},R(s_1)^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)},R(s_2)^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

- 另一种解决方式是不学习MDP，从经验中直接学习价值函数和策略
  - 也就是模型无关的强化学习（Model-free Reinforcement Learning）

# 马尔可夫决策过程总结

- MDP由一个五元组构成 $(S, A, \{P_{sa}\}, \gamma, R)$，其中状态转移$P$和奖励函数$R$构成了动态系统

- 动态系统和策略交互的占用度量

$$\rho^{\pi}(s,a) = \mathbb{E}_{a\sim\pi(s),\, s'\sim p(s,a)}\left[\sum_{t=0}^{T}\gamma^{t}\, p(s_t = s, a_t = a)\right]$$

- 一个白盒环境给定的情况下，可用动态规划的方法求解最优策略
  - 值迭代和策略迭代

- 如果环境是黑盒的，可以根据统计信息来拟合出动态环境$P$和$R$，然后做动态规划求解最优策略

# 无模型的强化学习

# 无模型的强化学习（Model-free RL）

- 在现实问题中，通常没有明确地给出状态转移和奖励函数
  - 例如，我们仅能观察到部分片段（episodes）

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \ldots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \ldots s_T^{(2)}$

- 模型无关的强化学习直接从经验中学习值（value）和策略（policy），而无需构建马尔可夫决策过程模型（MDP）
- 关键步骤：（1）估计值函数；（2）优化策略

# 值函数估计

- 在基于模型的强化学习（MDP）中，值函数能够通过动态规划计算获得

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$
$$= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

- 在模型无关的强化学习中
  - 我们无法直接获得 $P_{sa}$ 和 $R$
  - 但是，我们拥有一系列可以用来估计值函数的经验

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \ldots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \ldots s_T^{(2)}$

# 蒙特卡洛方法

# 收益估计

- 期望收益和采样次数的关系

$$Q_n(a^i) = \frac{r_1 + r_2 + \cdots + r_{n-1}}{n-1}$$

- 缺点：每次更新的空间复杂度是 $O(n)$

**增量实现**

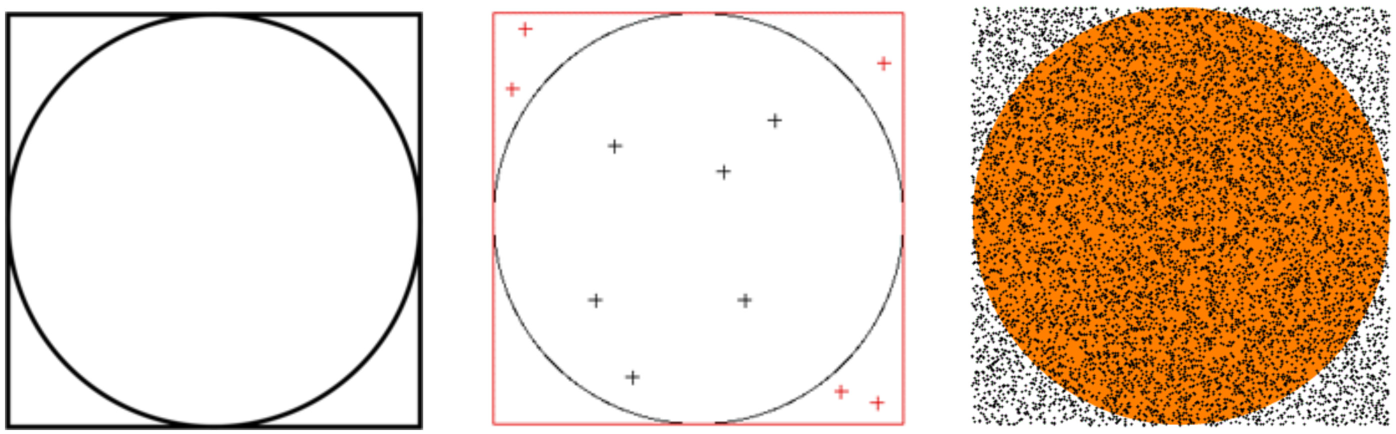
误差项：$\Delta_n^i$

$$Q_{n+1}(a^i) \coloneqq \frac{1}{n}\sum_{i=1}^{n} r_i = \frac{1}{n}\left(r_n + \frac{n-1}{n-1}\sum_{i=1}^{n-1} r_i\right) = \frac{1}{n}r_n + \frac{n-1}{n}Q_n = Q_n(a^i) + \frac{1}{n}(r_n - Q_n)$$

空间复杂度为 $O(1)$

# 蒙特卡洛方法

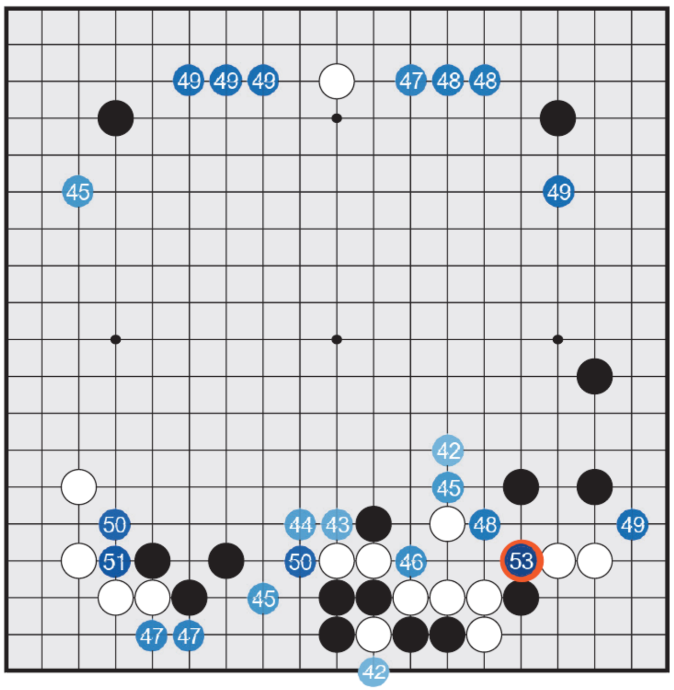
- 蒙特卡洛方法（Monte-Carlo methods）是一类广泛的计算算法。生活中处处都是MC方法。
  - 依赖于重复随机抽样来获得数值结果
- 例如，计算圆的面积

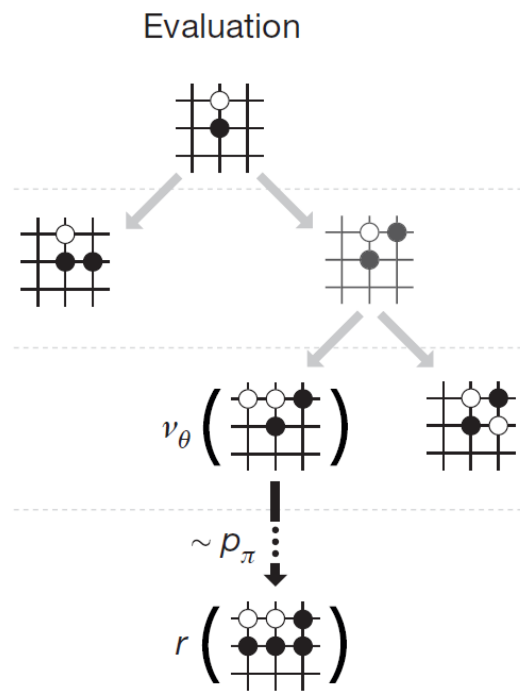


$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

# 蒙特卡洛方法

- 围棋对弈：估计当前状态下的胜率




$$\text{Win Rate}(s) = \frac{\#\text{win simulation cases started from } s}{\#\text{simulation cases started from } s \text{ in total}}$$

# 蒙特卡洛价值预测

# 蒙特卡洛价值估计

- 目标：从策略 $\pi$ 下的经验片段学习 $V^{\pi}$

$$s_0^{(i)} \xrightarrow[R_1^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[R_2^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[R_3^{(i)}]{a_2^{(i)}} s_3^{(i)} \ldots s_T^{(i)} \sim \pi$$

- 回顾：累计奖励（return）是总折扣奖励

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

- 回顾：值函数（value function）是期望累计奖励

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi] \\ &= \mathbb{E}[G_t | s_t = s, \pi] \\ &\simeq \frac{1}{N} \sum_{i=1}^{N} G_t^{(i)} \end{aligned}$$

- 使用策略 $\pi$ 从状态 $s$ 采样 $N$ 个片段
- 计算平均累计奖励

- 蒙特卡洛策略评估使用经验均值累计奖励而不是期望累计奖励

# 蒙特卡洛价值估计

## 实现

- 使用策略$\pi$采样片段

$$s_0^{(i)} \xrightarrow[R_1^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[R_2^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[R_3^{(i)}]{a_2^{(i)}} s_3^{(i)} \ldots s_T^{(i)} \sim \pi$$

- 在一个片段中的每个时间步长$t$的状态$s$都被访问
  - 增量计数器 $N(s) \leftarrow N(s) + 1$
  - 增量总累计奖励 $S(s) \leftarrow S(s) + G_t$
  - 价值被估计为累计奖励的均值 $V(s) = S(s)/N(s)$
  - 由大数定率有

$$V(s) \rightarrow V^{\pi}(s) \ \text{as} \ N(s) \rightarrow \infty$$

# 增量蒙特卡洛更新

- 每个片段结束后逐步更新$V(s)$
- 对于每个状态$S_t$和对应累计奖励$G_t$

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

- 对于非稳定的问题（即，环境会随时间发生变化），我们可以跟踪一个现阶段的平均值（即，不考虑过久之前的片段）

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

# 蒙特卡洛值估计

思路： $$V(S_t) \simeq \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$$

实现： $$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

- 蒙特卡洛方法：直接从经验片段进行学习
- 蒙特卡洛是模型无关的：未知马尔可夫决策过程的状态转移/奖励
- 蒙特卡洛从完整的片段中进行学习：没有使用bootstrapping的方法
- 蒙特卡洛采用最简单的思想：值（value）= 平均累计奖励（mean return）
- 注意：?

# 蒙特卡洛值估计

思路： $$V(S_t) \simeq \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$$

实现： $$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

- 蒙特卡洛方法：直接从经验片段进行学习
- 蒙特卡洛是模型无关的：未知马尔可夫决策过程的状态转移/奖励
- 蒙特卡洛从完整的片段中进行学习：没有使用bootstrapping的方法
- 蒙特卡洛采用最简单的思想：值（value）= 平均累计奖励（mean return）
- 注意：只能将蒙特卡洛方法应用于有限长度的马尔可夫决策过程中
  - 即，所有的片段都有终止状态

# 时序差分学习

# 时序差分学习（Temporal Difference Learning）

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

$$V(S_t) \leftarrow V(S_t) + \alpha\big(\underbrace{R_{t+1}}_{\text{观测值}} + \underbrace{\gamma V(S_{t+1})}_{\text{对未来的猜测}} - V(S_t)\big)$$

- 时序差分方法直接从经验片段中进行学习
- 时序差分是模型无关的
  - 不需要预先获取马尔可夫决策过程的状态转移/奖励
- 通过bootstrapping，时序差分从不完整的片段中学习
- 时序差分更新当前预测值使之接近估计累计奖励（非真实值）

# 蒙特卡洛 vs. 时序差分（MC vs. TD)

相同的目标：从策略$\pi$下的经验片段学习$V^{\pi}$

- 增量地进行每次蒙特卡洛过程（MC）
  - 更新值函数$V(S_t)$使之接近准确累计奖励$G_t$

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

- 最简单的时序差分学习算法（TD）：
  - 更新$V(S_t)$使之接近估计累计奖励$R_{t+1} + \gamma V(S_{t+1})$

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

  - 时序差分目标：$R_{t+1} + \gamma V(S_{t+1})$
  - 时序差分误差：$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

# 驾车回家的例子（MC vs. TD)

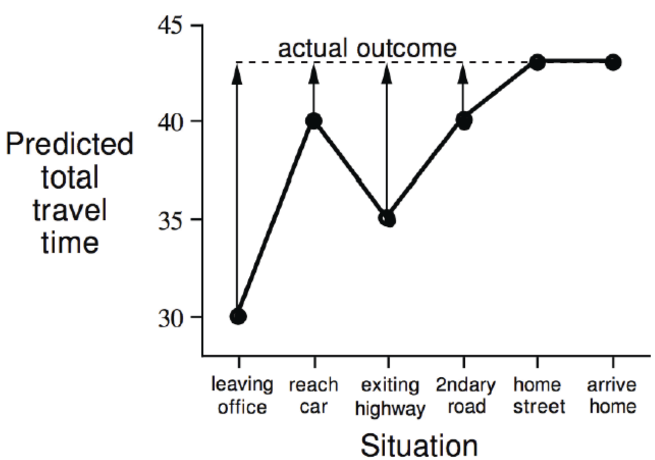
| 状态 | 经过的时间(分钟) | 预计所剩时间 | 预计总时间 |
|---|---|---|---|
| 离开公司 | 0 | 30 | 30 |
| 开始驾车，下雨 | 5 | 35 | 40 |
| 离开高速公路 | 20 | 15 | 35 |
| 卡车后跟车 | 30 | 10 | 40 |
| 到达家所在街道 | 40 | 3 | 43 |
| 直奔家门 | 43 | 0 | 43 |

?

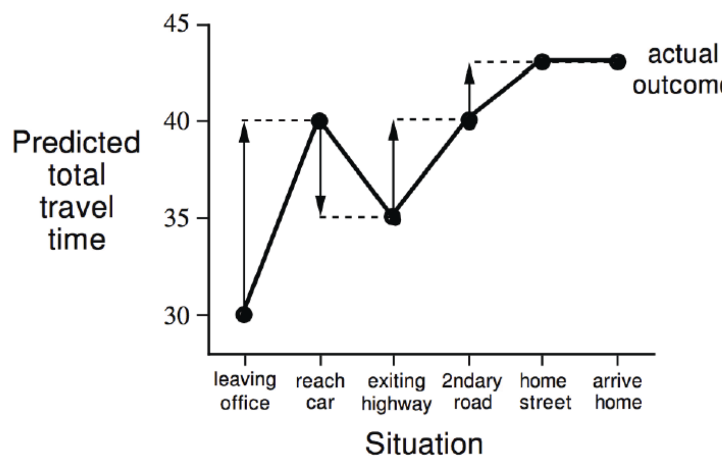
# 驾车回家的例子（MC vs. TD)

| 状态 | 经过的时间 (分钟) | 预计所剩时间 | 预计总时间 |
|---|---|---|---|
| 离开公司 | 0 | 30 | 30 |
| 开始驾车，下雨 | 5 | 35 | 40 |
| 离开高速公路 | 20 | 15 | 35 |
| 卡车后跟车 | 30 | 10 | 40 |
| 到达家所在街道 | 40 | 3 | 43 |
| 直奔家门 | 43 | 0 | 43 |

Changes recommended by Monte Carlo methods ($\alpha$=1)




Changes recommended by TD methods ($\alpha$=1)



# 蒙特卡洛（MC）和时序差分（TD）的优缺点

- 时序差分：能够在知道最后结果之前进行学习
  - 时序差分能够在每一步之后进行在线学习
  - 蒙特卡洛必须等待片段结束，直到累计奖励已知

- 时序差分：能够无需最后结果地进行学习
  - 时序差分能够从不完整的序列中学习
  - 蒙特卡洛只能从完整序列中学习
  - 时序差分在连续（无终止的）环境下工作
  - 蒙特卡洛只能在片段化的（有终止的）环境下工作

# 偏差（Bias）/方差（Variance）的权衡

- 累计奖励$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$是$V^\pi(S_t)$的无偏估计
- 时序差分真实目标$R_{t+1} + \gamma V^\pi(S_{t+1})$是$V^\pi(S_t)$的无偏估计
- 时序差分目标$R_{t+1} + \gamma \underbrace{V(S_{t+1})}_{\text{当前估计}}$是$V^\pi(S_t)$的有偏估计
- 时序差分目标具有比累计奖励更低的方差
  - 累计奖励——取决于多步随机动作，多步状态转移和多步奖励
  - 时序差分目标——取决于单步随机动作，单步状态转移和单步奖励

# 蒙特卡洛（MC）和时序差分（TD）的优缺点（2）

MC:

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

蒙特卡洛具有高方差，无偏差

- 良好的收敛性质
  - 使用函数近似时依然如此
- 对初始值不敏感
- 易于理解和使用

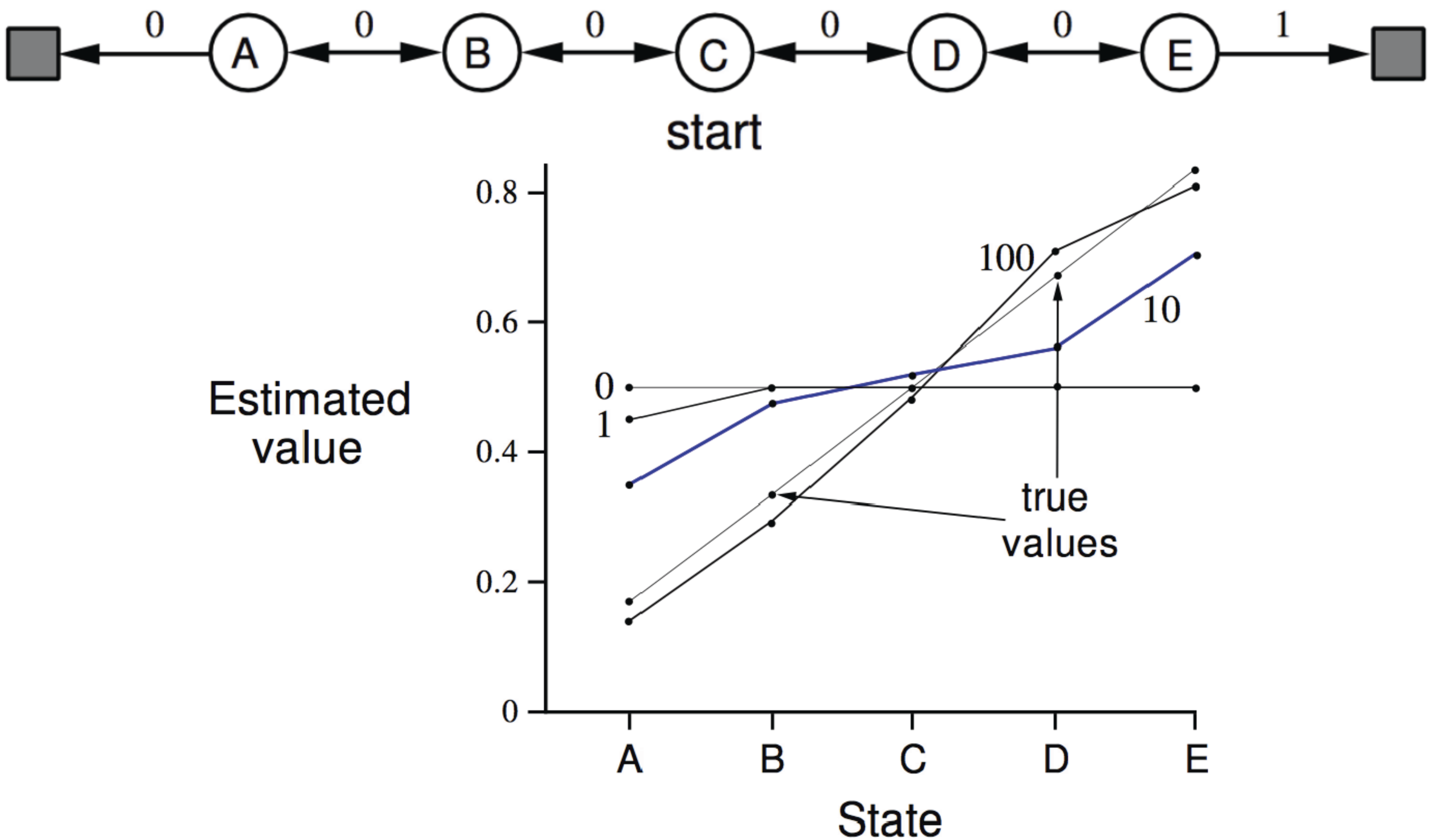
TD:

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

时序差分具有低方差，有偏差

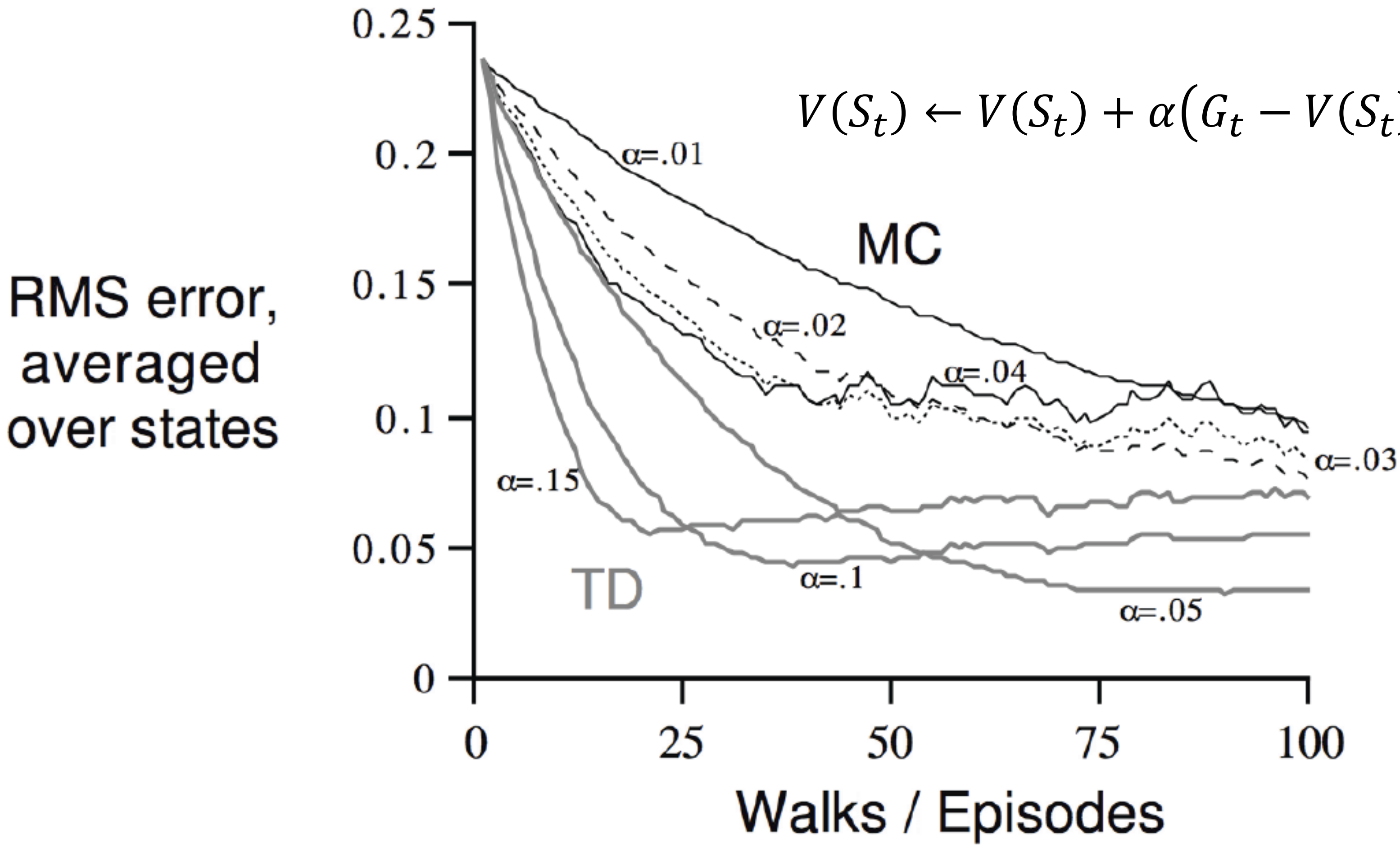
- 通常比蒙特卡洛更加高效
- 时序差分最终收敛到$V^{\pi}(S_t)$
  - 但使用函数近似并不总是如此
- 比蒙特卡洛对初始值更加敏感

# 随机游走的例子



0
A
0
B
0
C
0
D
0
E
1
start
Estimated value
0.8
0.6
0.4
0.2
0
0
1
100
10
true values
A
B
C
D
E
State

# 随机游走的例子

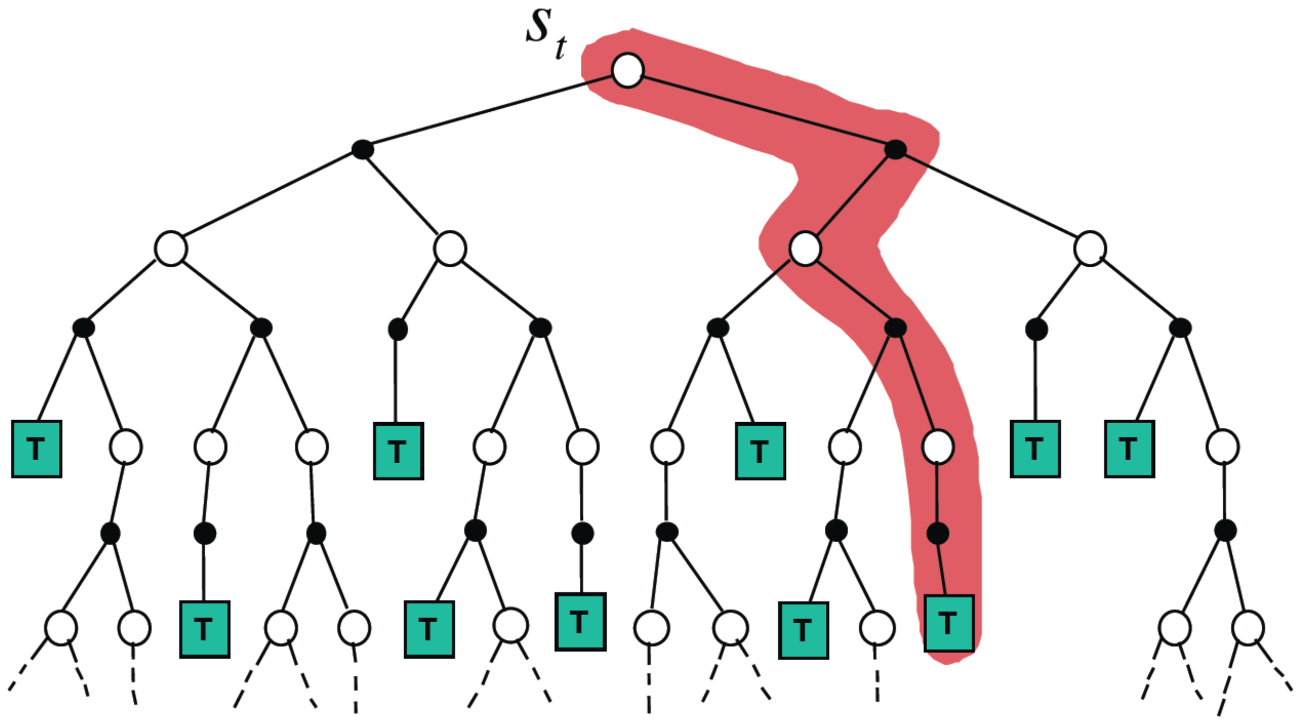



$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

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

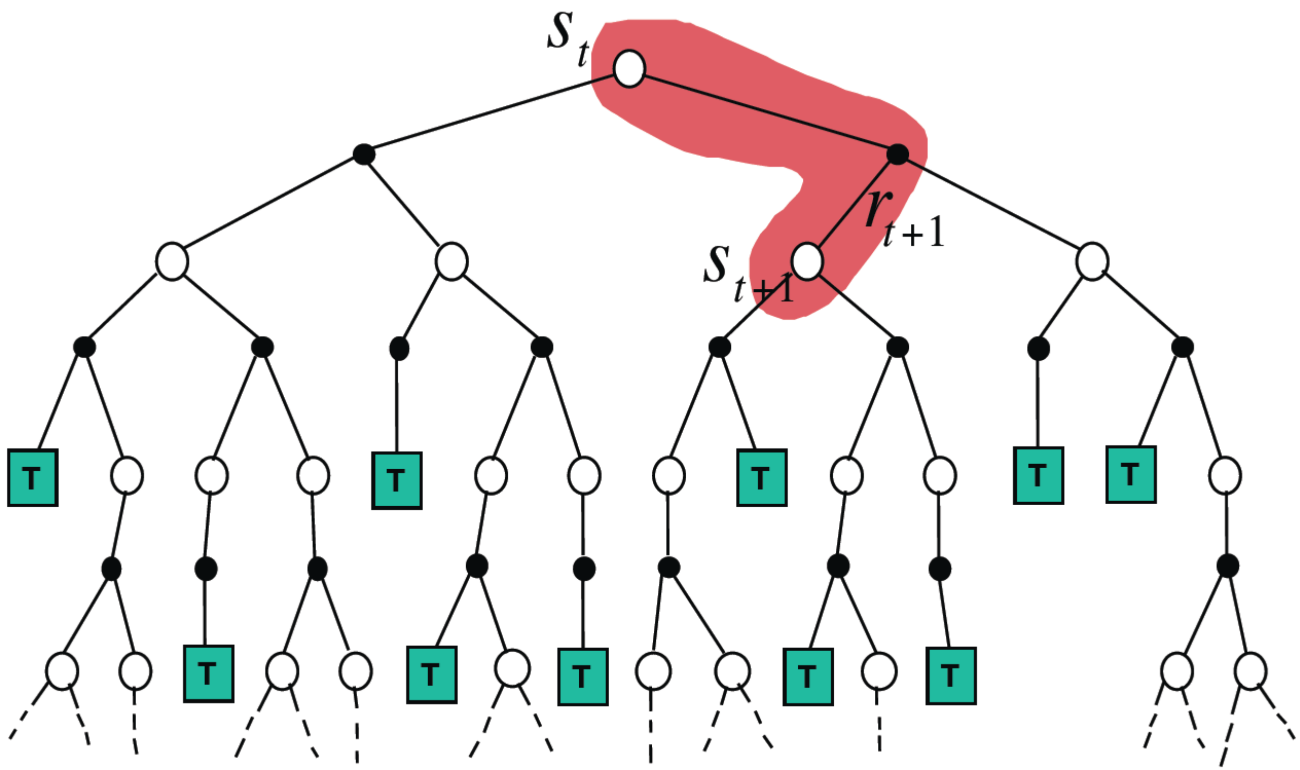
# 蒙特卡洛反向传播（Backup）

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



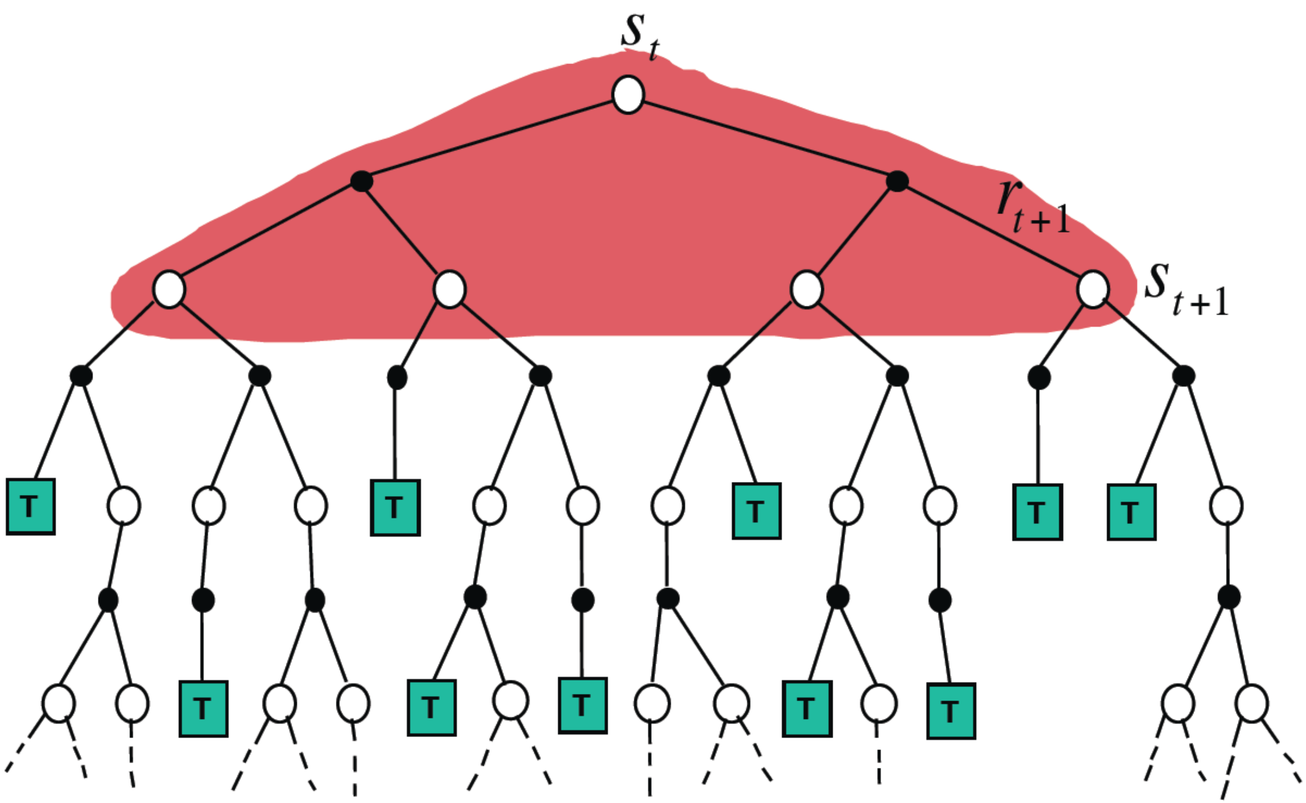
# 时序差分反向传播（Backup）

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



# 动态规划反向传播（Backup）

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



# 多步时序差分学习

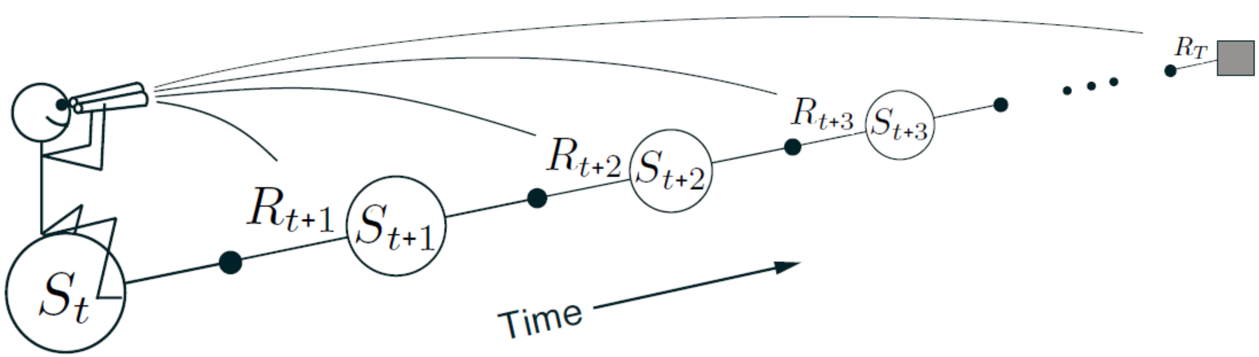
- 对于有时间约束的情况，我们可以跳过$n$步预测的部分，直接进入模型无关的控制

- 定义$n$步累计奖励

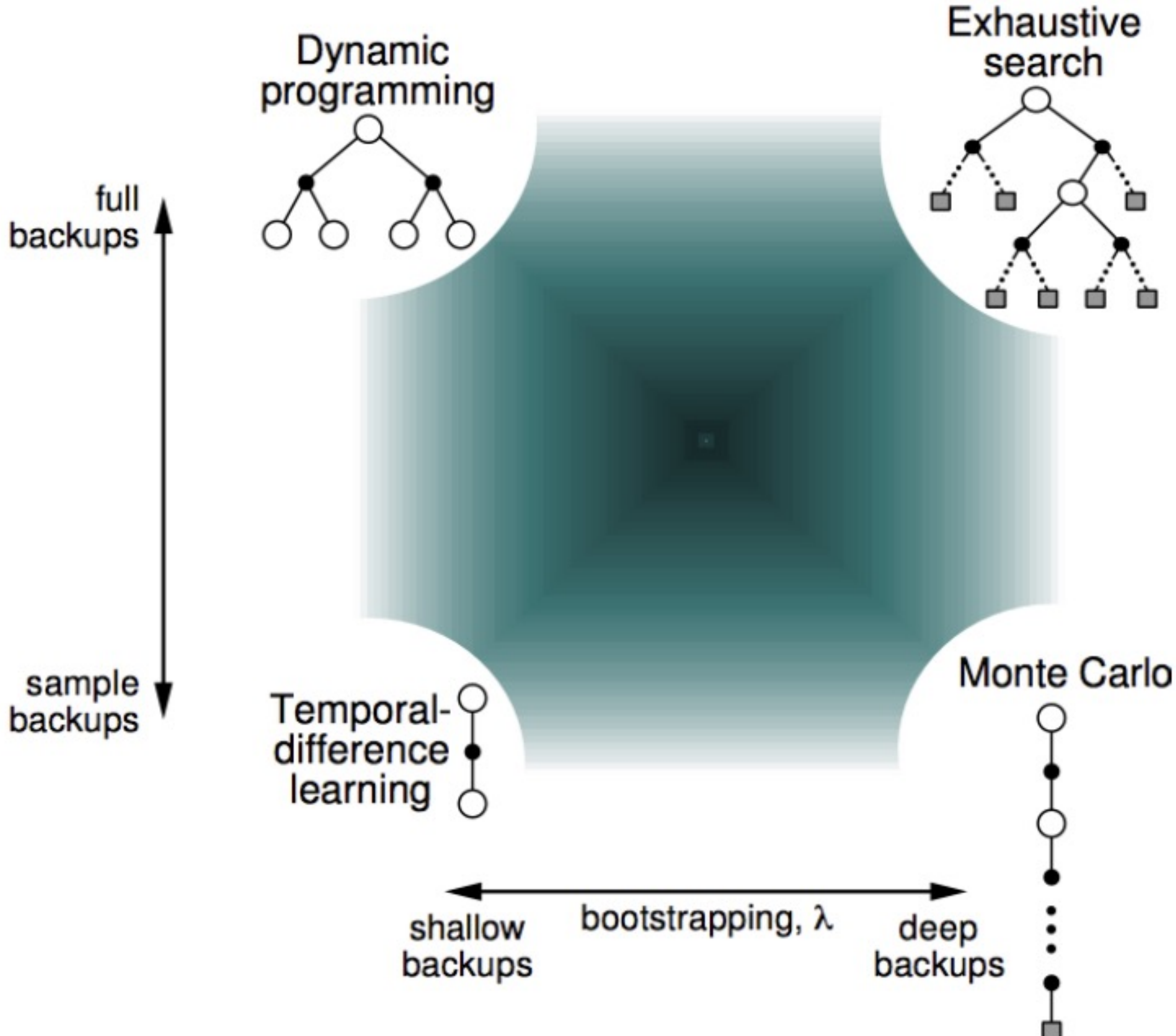
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- $n$步时序差分学习

$$V(S_t) \leftarrow V(S_t) + \alpha \left( G_t^{(n)} - V(S_t) \right)$$



# 总览强化学习值函数估计多种方法




Slide credit: David Silver

# 重要性采样

# 重要性采样

- 估计一个不同分布的期望

$$
\begin{aligned}
\mathbb{E}_{x\sim p}[f(x)] &= \int_x p(x)f(x)dx \\
&= \int_x q(x)\frac{p(x)}{q(x)}f(x)dx \\
&= \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]
\end{aligned}
$$

- 将每个实例的权重重新分配为 $\beta(x)=\frac{p(x)}{q(x)}$

# 使用重要性采样的离线策略蒙特卡洛

- 使用策略$\mu$产生的累计奖励评估策略$\pi$
- 根据两个策略之间的重要性比率（importance ratio）对累计奖励$G_t$加权
- 每个片段乘以重要性比率

$$\{s_1, a_1, r_2, s_2, a_2, \ldots, s_T\} \sim \mu$$

$$G_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \cdots \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} G_t$$

# 使用重要性采样的离线策略蒙特卡洛

- 更新值函数以逼近修正的累计奖励值

$$V(s_t) \leftarrow V(s_t) + \alpha\left(G_t^{\pi/\mu} - V(s_t)\right)$$

无法在$\pi$非零而$\mu$为零时使用

重要性采样将显著增大方差（variance）

# 使用重要性采样的离线策略时序差分

- 使用策略$\mu$产生的时序差分目标评估策略$\pi$
- 根据重要性采样对时序差分目标$r + \gamma V(s')$加权
- 仅需要一步来进行重要性采样修正

$$V(s_t) \leftarrow V(s_t) + \alpha\left(\underbrace{\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}}_{\text{重要性采样修正}}\underbrace{\left(r_{t+1} + \gamma V(s_{t+1})\right)}_{\text{时序差分目标}} - V(s_t)\right)$$

具有比蒙特卡洛重要性采样更低的方差

策略仅需在单步中被近似

# 值函数估计总结

- 无模型的强化学习在黑盒环境下使用
- 要优化智能体策略，首要任务则是精准、高效地估计状态或者(状态、动作)的价值
- 在黑盒环境下，值函数的估计方法主要包括蒙特卡洛方法和时序差分法
- 蒙特卡洛方法通过采样到底的方式直接估计价值函数
- 时序差分学习通过下一步的价值估计来更新当前一步的价值估计
- 实际使用中，时序差分方法更加常见

THANK YOU