

# 法律声明

---

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。



关注 小象学院

# 第5讲 时序差分法

---

## 强化学习

主讲人：叶梓

上海交通大学博士

主要研究方向：机器学习、深度学习、人工智能

# 本章内容

---

- TD法与MC法的区别
- 时序差分法详述
- SARSA
- SARSA( $\lambda$ )
- Q-learning
- Double Q-learning
- 案例演示

# 时序差分方法

## □ DP

- 需要环境模型，即状态转移概率  $P_{SS'}^a$ ,
- 状态值函数的估计是自举的(bootstrapping),
- 即当前状态值函数的更新依赖于已知的其他状态值函数。

## □ MC

- 可以从经验中学习不需要环境模型;
- 状态值函数的估计是相互独立的;
- 只能用于episode tasks

## □ 时序差分学习(Temporal-Difference learning, TD learning)

- 结合了DP和MC，并兼具两种算法的优点
- 它不需要环境模型，属于model-free的算法
- 它不局限于episode task，可以用于连续的任务

# 时间差分方法的优点

---

- ❑ 相比于 DP 方法，TD 方法不需要环境的模型，与 MC 相同。
- ❑ 相比于 Monte Carlo 方法，TD 方法可以采用在线的、完全增量式的实现方式：
  - ❑ 在 Monte Carlo 方法中，必须要等到 episode 结束，有了 return 之后才能更新，在有些应用中 episodes 的时间很长，甚至有些应用环境是连续型任务，根本没有 episodes。
  - ❑ 而 TD 方法不需等到最终的真实的 return，并且 TD 方法可以保证收敛到  $V_{\pi}$ 。

# 时序差分法与蒙特卡罗法

- 时序差分法和蒙特卡罗法都使用经验来解决预测问题，给定服从规则  $\pi$  的一些经历，两种方法均可更新经历中的每一个非终止的状态  $S_t$  的  $V_\pi$ 。
- Monte Carlo 方法要等到 return（回报）知道之后才将其设为是  $V(S_t)$  的目标值。
- 回忆一下，基于增量的every-visit的Monte Carlo 方法V值计算。

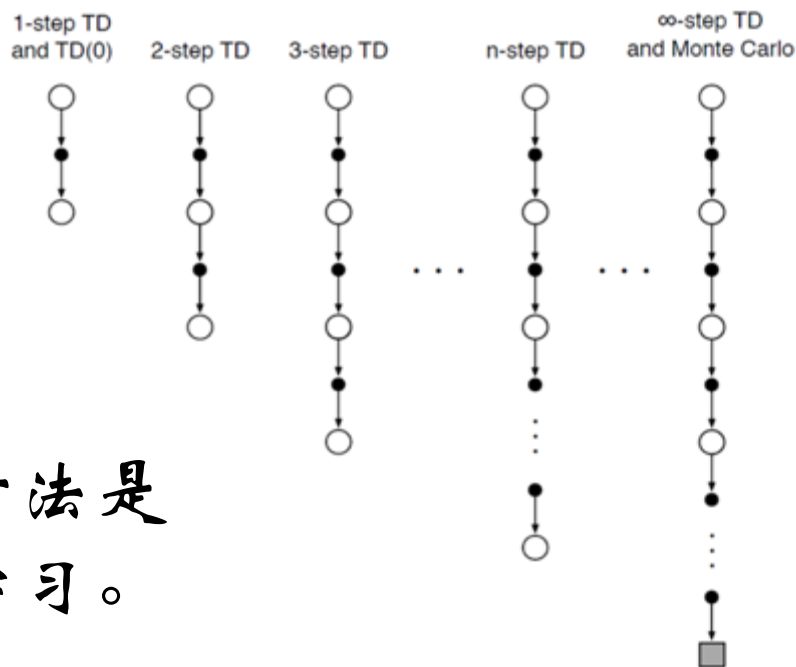
$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$



目标：时间t后的实际回报

# 时序差分法与蒙特卡罗法

- 时序差分学习是模拟（或者经历）一段序列，
- 每行动一步（或者几步），根据新状态的价值，然后估计执行前的状态价值。



- 可以认为蒙特卡洛的方法是最大步数的时序差分学习。

# 时序差分法

$$\mu_k = \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})$$

- TD 方法只需等到下一个 time step 即可，即在时刻  $t+1$ ，TD 方法立刻形成一个 target，并使用观测到的 reward  $R_{t+1}$  和估计的  $V(S_{t+1})$  进行更新。
- 最简单的 TD 方法称为是 TD(0)，其更新方法为：

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

目标：估计回报

TD目标

- $R_{t+1} + \gamma V(S_{t+1})$  is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  is called the *TD error*

TD误差



# 时序差分法

- TD 方法与MC法一样，都是基于已有的估计进行更新，
- 只不过Monte Carlo 更新的目标值为 $G_t$ ，而TD更新的目标值为 $R_{t+1} + \gamma V(S_{t+1})$ ，这两者之间的关系其实可以从下面的式子来更好的理解：

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t | S_t = s]$$

MC法

$$= \mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

$$= \mathbb{E}_{\pi} \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s \right]$$

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

TD法

DP法

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

# 时序差分方法

## □ 用于策略评价的TD(0) 伪代码

Tabular是什么？

### Tabular TD(0) for estimating $v_\pi$

Input: the policy  $\pi$  to be evaluated

Initialize  $V(s)$  arbitrarily (e.g.,  $V(s) = 0, \forall s \in \mathcal{S}^+$ )

Repeat (for each episode):

Initialize  $S$

Repeat (for each step of episode):

$A \leftarrow$  action given by  $\pi$  for  $S$

Take action  $A$ , observe  $R, S'$

$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

until  $S$  is terminal

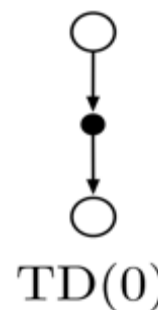
注意：即时奖励R必须有！

[http://blog.csdn.net/coffee\\_cream](http://blog.csdn.net/coffee_cream)

# TD方法的反演

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

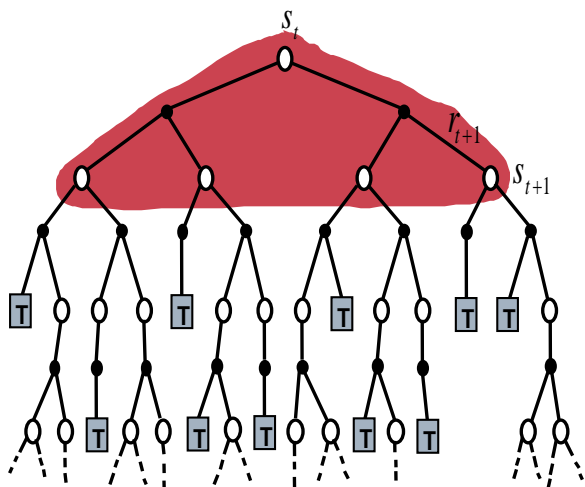
- TD(0) 的 backup diagram 如下图所示，它对最上面的节点的 value 评估值的更新基于的是从它到下一个 state 的一次样本转换
- TD 和 Monte Carlo 更新都被称为是 sample backups，因为他们都涉及到采样的连续的状态或状态对。
- sample back-ups 与 DP 方法的 full backups 的不同在于，它利用的不是所有可能的转换的完全分布，而是一个单一的样本转换。



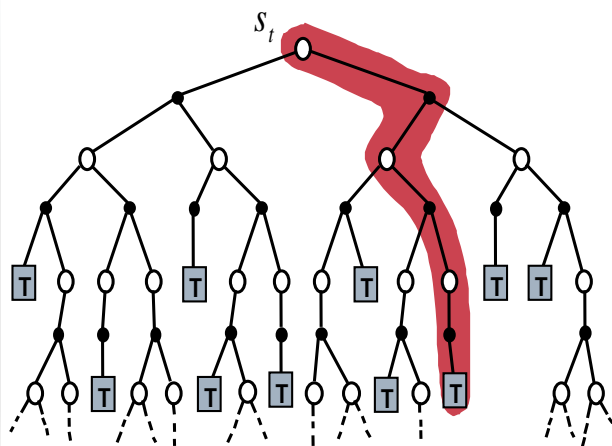
# DP、MC、TD各自的反演图

□ 现在再看这三张图，是否更容易理解了？

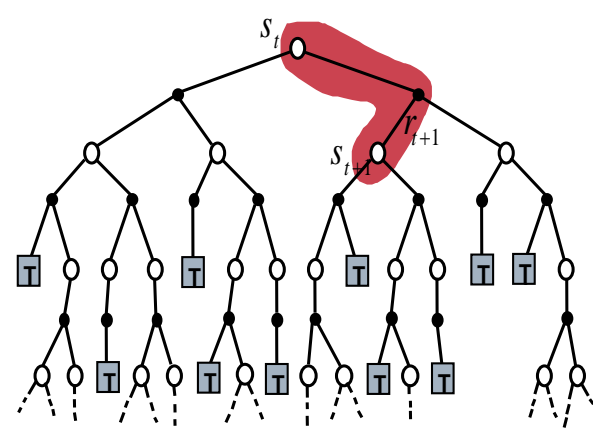
动态规划



蒙特卡罗



TD(0)



# TD误差与MC误差

- 在 TD 学习中有一个重要的概念叫 **TD error**，用  $\delta_t$  表示，它表示的就是在 **该时刻** 估计的误差，在 TD(0) 中它指的就是  $R_{t+1} + \gamma V(S_{t+1})$  与  $V(S_t)$  的差：

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

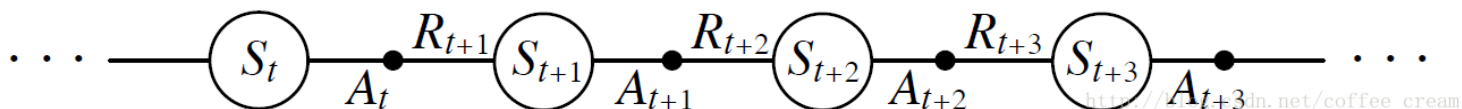
- **Monte Carlo error** 可以写作是一系列 TD errors 的和。

MC error

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t+1}\delta_{T-t+1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t+1}\delta_{T-t+1} + \gamma^{T-t}(0 - 0) \\ &= \sum_{k=0}^{T-t+1} \gamma^k \delta_{t+k} \end{aligned}$$

# SARSA: 同策略的TD

- 与 MC一样，TD control method 也包含 on-policy 和 off-policy。
- 对于 on-policy 方法
  - 首先是要对当前的行为规则  $\pi$  估计所有状态  $s$  和行为  $a$  的  $q_\pi(s,a)$ ,
  - 估计的方法与上面介绍的学习  $v_\pi$  的方法一样，每一个 episode 是由一系列 states 和 state-action 对组成的转换序列：



# SARSA: 同策略的TD

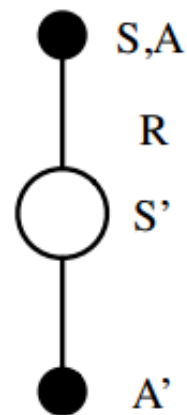
- 这里需要关注的是从一个“state-action对”向另外一个“state-action对”的转换过程，可知，该评估值更新方程为：

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

- 每次从一个非终止状态  $S_t$  进行转换都会利用上面的更新方程来更新  $Q(S_t, A_t)$ ,
- 注意到该方程中一共包含了5个元素： $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ ，因此将这种方法称为是 SARSA。

# SARSA: 同策略的TD控制

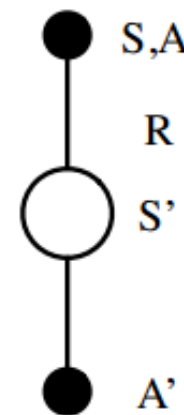
- 一个Agent处在某一个状态 $S$ ，在这个状态下它可尝试各种不同的行为。
- 当遵循某一策略时，会**根据当前策略**选择一个行为 $A$ 。
- 个体**实际执行这个行为**，与环境发生实际交互，环境会根据其行为给出即时奖励 $R$ ，并且进入下一个状态 $S'$ 。





# SARSA: 同策略的TD控制

- 在这个后续状态 $S'$ ，再次依当前策略产生一个行为 $A'$ 。 $S', A'$
- 此时，个体并不执行该行为，而是通过自身当前的状态行为价值函数得到该“状态行为对” $(S', A')$ 的价值 $Q$ ,
- 利用该价值同时结合个体在 $S$ 状态下采取行为 $A$ 所获得的即时奖励来更新个体在 $S$ 状态下采取行为 $A$ 的状态行为价值。



# SARSA的收敛性

- 当行为策略满足GLIE特性，同时学习速率参数 $\alpha$ 满足：

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \text{ 且 } \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

- SARSA算法将收敛至最优策略和最优价值函数。
- GLIE，即在有限的时间内进行无限可能的探索：
  - 一是所有的状态行为对会被无限次探索，即确保探索到了所有的状态和行为：
  - 二是另外随着采样趋向无穷多，策略收敛至一个贪婪策略：

# SARSA的伪码

□ 基于 SARSA 方法，就可以得到 on-policy control algorithm。

■ 对所有 on-policy method，在持续的对行为规则  $\pi$  估计  $q_\pi$  的同时，也依据  $q_\pi$  采用贪婪的方式来修改行为规则  $\pi$ 。

## Sarsa: An on-policy TD control algorithm

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize  $S$

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

Repeat (for each step of episode):

Take action  $A$ , observe  $R, S'$

Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$   
until  $S$  is terminal

同策略

留到下一轮执行

为什么需要A'?

[http://blog.csdn.net/coffee\\_green](http://blog.csdn.net/coffee_green)

# SARSA的特性

---

- ❑ 在 SARSA 算法中， $Q(S,A)$  的值使用一张大表来存储的，不适合解决规模很大的问题；
- ❑ 对于每一个状态序列 Episode，在 S 状态时采取的行为 A 是基于当前行为策略的，同时该行为也是实际 Episode 发生的行为，
- ❑ 在更新状态行为对  $(S,A)$  的价值的循环里，agent 并不实际执行在  $S'$  下的  $A'$  行为，而是将行为  $A'$  留到下一个循环执行。

# SARSA( $\lambda$ )

- 根据前面的n-步收获，类似的可以引出一个n-步 SARSA的概念。

$n=1$	TD 或 TD(o)	$q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1})$
$n=2$		$q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2})$
...		
$n=\infty$	MC	$q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$

n=2时，就用向前用2步的即时奖励，再加上两步后的新状态的价值。

- 这里的对应的是一个状态行为对，表示的是在某个状态下采取某个行为的价值大小。

# n-步Q回报

## □ 第n步Q回报 (Q-return)

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

□ 这里的n-步Q收获，Q是包含行为的，也就是在当前策略下基于某一状态产生的行为。

□ 有了如上定义，可以把n-步SARSA用n-步Q收获来表示，如下式：

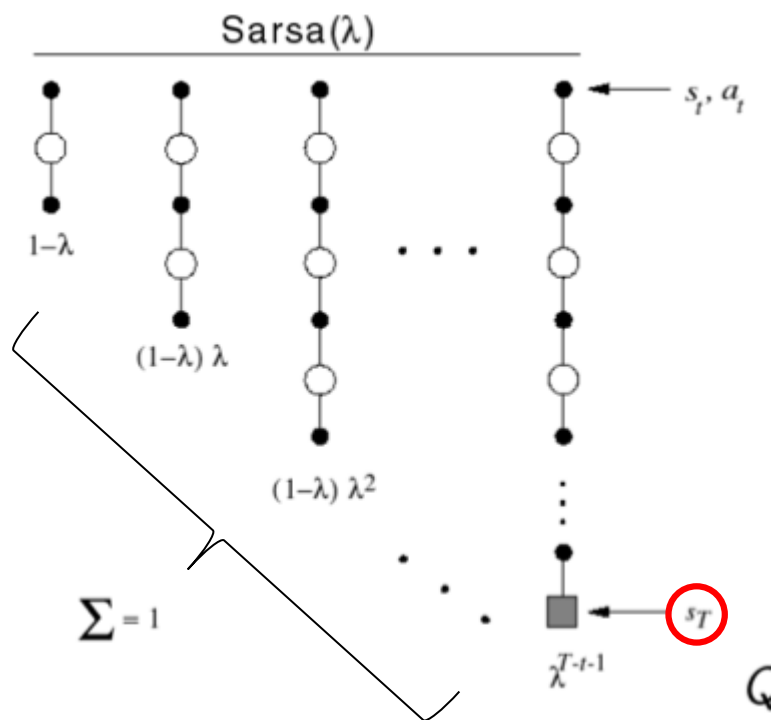
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [q_t^{(n)} - Q(S_t, A_t)]$$

□ 每一步都有一个 $q(n)$ ，可否利用加权的方式来融合这n个估计值呢？

# 加权的Q回报

- 可以给 n-步 Q 回报中的每一步收获分配一个权重 $\lambda$ ，并按权重对每一步 Q 回报求和，那么将得到 $q^\lambda$ 回报：

$$q_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$



# Sarsa( $\lambda$ ) 的前向观点

- 如果用某一状态的 $q^\lambda$ 收获来更新状态行为对的Q值，那么可以表示成如下的形式：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^{(\lambda)} - Q(S_t, A_t))$$

- 这是SARSA( $\lambda$ ) 的前向观点，使用它更新 Q 价值需要遍历完整的状态序列（到终点）。
- 有没有可能不到终点状态就可以更新当前状态的值函数呢？



# Sarsa( $\lambda$ ) 的后向观点

- Sarsa( $\lambda$ )后向观点需要引入Eligibility Trace (适合度轨迹) 概念,
- 不同的是这次的E值针对的不是一个状态, 而是一个状态行为对:
  - $E_0(s,a)=0$
  - $E_t(s,a)=\gamma\lambda E_{t-1}(s,a)+1$  (if  $S_t=s, A_t=a$ )
  - $E_t(s,a)=\gamma\lambda E_{t-1}(s,a)$  (if  $S_t\neq s, A_t\neq a$ )
- 它体现的是一个结果与某一个状态行为对的因果关系, 与得到结果最近的状态行为对, 以及哪些在此之前频繁发生的状态行为对对这个结果的影响最大。

# Sarsa( $\lambda$ ) 的后向观点

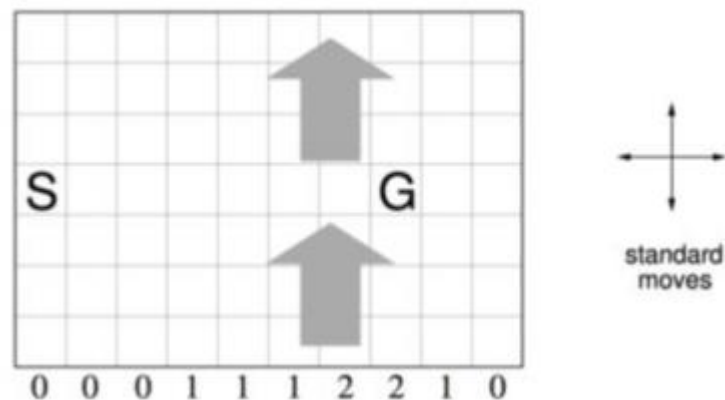
- 下式是引入Eligibility Trace概念的SARSA( $\lambda$ )之后的Q值更新描述：

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$
$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)$$

- 引入Eligibility Trace概念，SARSA( $\lambda$ )将可以更有效的在线学习，因为不必要学习完整的Episode，数据用完即可丢弃。属于增量式的方法。
- 当一个回合结束后，前向观点和后向观点每个状态的值函数更新总量是相等的。

# SARSA示例：有风的格子世界

- 如图，环境是一个10\*7的长方形格子世界，同时有一个起始位置S和一个终止目标位置G，
- 水平下方的数字表示对应的列中有一定强度的风，当该数字是1时，个体进入该列的某个格子时，会按图中箭头所示的方向自动移动一格，当数字为2时，表示顺风移动2格，以此类推模拟风的作用。
- 任何试图离开格子世界的行为都会使得个体停留在移动前的位置。



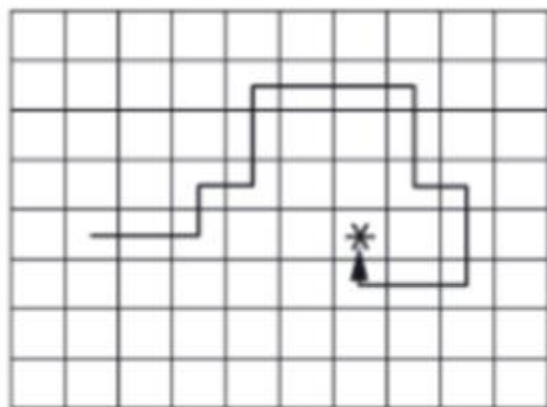
# SARSA示例：有风的格子世界

- 个体从环境观测不到自身位置、起始位置以及终止位置信息的坐标描述，个体在与环境进行交互的过程中学习到自身及其它格子的位置关系。
- 个体的行为空间是离散的四个方向。
- 可设置个体每行走一步获得即时奖励为-1，直到到达终止目标位置的即时奖励为0，借此希望找到最优策略。
- 衰减系数 $\lambda$ 设为1。

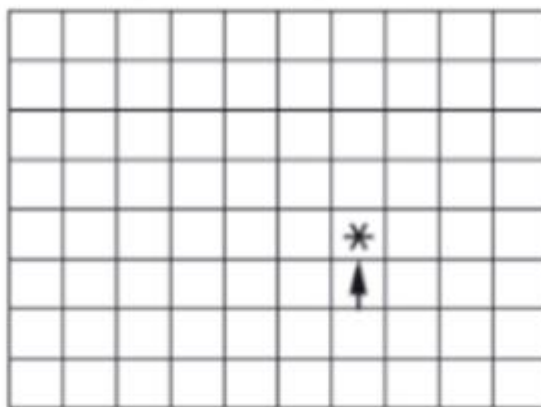
# SARSA(0)与SARSA( $\lambda$ )

- ❑ SARSA(0), 更新后只有最接近终点的那一点改变了;
- ❑ SARSA( $\lambda$ ), 则是整个路径都更新了。

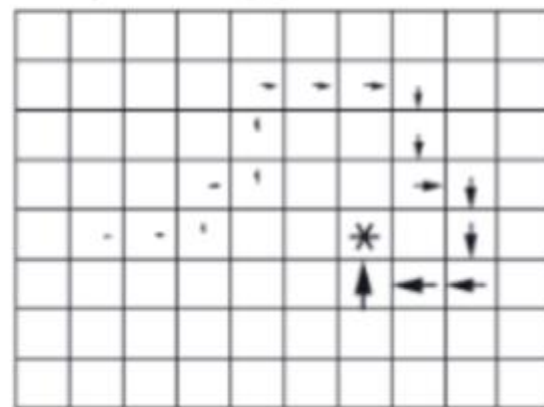
### 个体选择的路线



### 一次SARSA更新后的行为价值的改变



SARSA( $\lambda$ )更新后的  
行为价值的改变( $\lambda=0.9$ )



# SARSA(0)

---

- 一开始个体对环境一无所知，即所有的  $Q$  值均为 0，它将随机选取移步行为。
- 在到达终点前的每一个位置  $S$ ，个体依据当前策略，产生一个移步行为，执行该行为，环境会将其放置到一个新位置，同时给以即时奖励 0；
- 在新的位置  $S'$  上，根据当前的策略它会产生新位置下的一个行为，个体不执行该行为，仅仅在表中查找新状态下新行为的  $Q'$  值；
- 由于  $Q=0$ ，依据更新公式，它将把刚才离开的位置以及对应的行为的状态行为对  $Q(S,A)$  价值更新为 0。

# SARSA(0)

- 如此直到个体最到达终点位置  $S_G$ ，它获得一个即时奖励 1，此时agent会依据公式更新其到达终点位置前所在那个位置（用  $S_H$  表示，级终点位置下方）时采取向上移步的那个状态行为对价值  $Q(S_H, A_{up})$ ，它将不再是 0，这是agent在这个状态序列中唯一一次用非 0 数值来更新  $Q$  值。
- 注意：这里不要误认为 Sarsa 算法只在经历一个完整的状态序列之后才更新，在这个例子中，它每走一步都会更新，只是多数时候更新的数据和原来一样罢了。

# SARSA(0)

- 此时如果要求agent继续学习，则环境将其放入起点。个体的第二次寻路过程一开始与首次一样都是盲目随机的，直到其进入终点位置下方的位置  $S_H$ ，在这个位置，agent更新的策略将使其有非常大的几率选择向上的行为直接进入终点位置  $S_G$ 。
- 同样，经过第二次的寻路，agent了解到到达终点下方的位置  $S_H$  价值比较大，因为在这个位置直接采取向上移步的行为就可以拿到到达终点的即时奖励。因此它会将那些通过移动一步就可以到达  $S_H$  位置的其它位置以及相应的到达该位置位置所要采取的行为对所对应的价值进行提升。



# SARSA(0)

---

- 如此反复，如果采用贪婪策略更新，agent最终将得到一条到达终点的路径，不过这条路径的倒数第二步永远是在终点位置的下方。
- 如果采用 $\epsilon$ -greedy策略更新，那么个体还会尝试到终点位置的左上右等其它方向的相邻位置价值也比较大，此时个体每次完成的路径可能都不一样。

# SARSA( $\lambda$ )

- 该算法同时还针对每一次状态序列维护一个关于状态行为对  $(S,A)$  的 E 表，初始时 E 表值均为 0。当 agent 首次在起点  $S_0$  决定移动一步  $A_0$  (假设向右) 时，它被环境告知新位置为  $S$ ，此时发生如下事情：
- 首先，agent 会做一个标记，使  $E(S_0,A_0)$  的值增加 1，表明 agent 刚刚经历过这个事件  $(S_0,A_0)$ ；
- 其次，它要估计这个事件的对于解决整个问题的价值，也就是估计 TD 误差，此时依据公式结果为 0，说明 agent 认为在起点处向右走没什么价值，这个“没有什么价值”有两层含义：
  - 不仅说明在  $S_0$  处往右目前对解决问题没有积极帮助，
  - 同时表明 agent 认为所有能够到达  $S_0$  状态的状态行为对的价值没有任何积极或消极的变化。

# SARSA( $\lambda$ )

---

- 随后，agent将要更新该状态序列中所有已经经历的  $Q(S,A)$  值，由于存在  $E$  值，那些在  $(S_0, A_0)$  之前近期发生或频繁发生的  $(S,A)$  的  $Q$  值将改变得比其它  $Q$  值明显些，此外agent还要更新其  $E$  值，以备下次使用。
- 对于刚从起点出发的agent，这次更新没有使得任何  $Q$  值发生变化，仅仅在  $E(S_0, A_0)$  处有了一个实质的变化。
- 之后的过程类似，agent有意义的发现就是对路径有一个记忆，体现在  $E$  里，具体的  $Q$  值没发生变化。

# SARSA( $\lambda$ )

- 这一情况直到agent到达终点位置时发生改变。此时agent得到了一个即时奖励 1，它会发现这一次变化（从  $S_H$  采取向上行为  $A_{up}$  到达  $S_G$ ）价值明显；
- 它会计算这个 TD 误差为 1，同时告诉整个经历过程中所有  $(S,A)$ ，根据其 与  $(S_H, A_{up})$  的密切关系更新这些状态行为对的价值  $Q$ ，agent 在这个状态序列中经历的所有状态行为对的  $Q$  值都将得到一个非 0 的更新；
- 但是那些在个体到达  $S_H$  之前就近发生以及频繁发生的状态行为对的价值提升得更加明显。

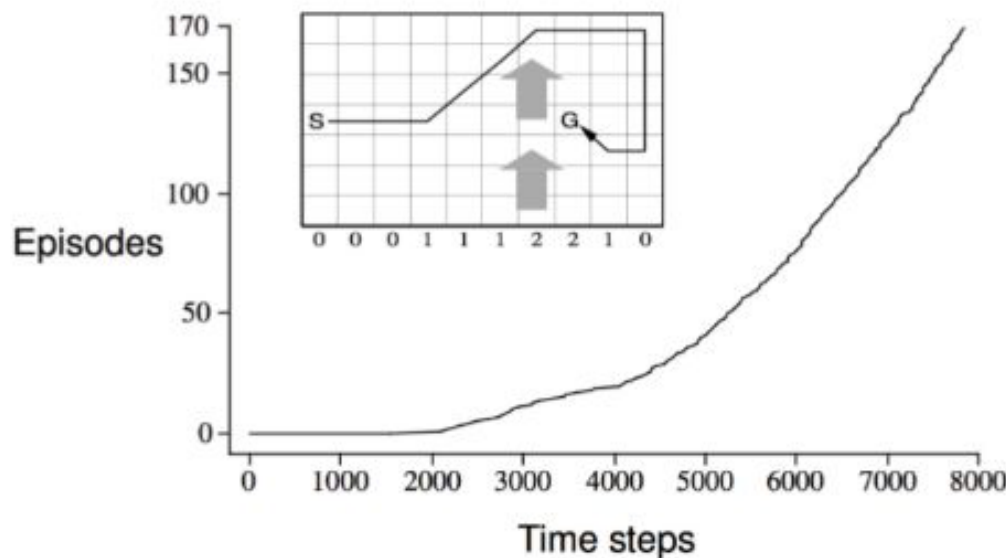
# SARSA( $\lambda$ )

---

- 当agent重新从起点第二次出发时，它会发现起点处向右走的价值不再是0。如果采用贪婪策略更新，agent将根据上次经验得到的新策略直接选择右走，并且一直按照原路找到终点。
- 如果采用 $\epsilon$ -贪婪策略更新，那么agent还会尝试新的路线。
- 由于为了解释方便，做了一些约定，这会导致问题并不要求agent找到最短一条路径，如果需要找最短路径，需要在每一次状态转移时给agent一个负的奖励。

# SARSA示例：有风的格子世界

- 个体通过学习发现下面的行为序列（共15步）  
能够得到最大程度的奖励：-14
- R,R,R,R,R,R,R,R,R,D,D,D,D,L,L



# 异策略的TD法

- 异策略的TD学习任务就是使用TD方法在目标策略  $\pi(a|s)$  的基础上更新行为价值，进而优化行为策略：

$$V(S_t) \leftarrow V(S_t) + \alpha \left[ \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right]$$

- 对于上式，agent处在状态  $S_t$  中，基于行为策略  $\mu$  产生了一个行为  $A_t$ ，执行该行为后进入新的状态  $S_{t+1}$ 。
- 异策略学习要做的事情就是，比较借鉴策略和行为策略在状态  $S_t$  下产生同样的行为  $A_t$  的概率的比值。

# 异策略的TD法

- 如果这个比值接近 1，说明两个策略在状态  $S_t$  下采取的行为  $A_t$  的概率差不多，此次对于状态  $S_t$  价值的更新同时得到两个策略的支持。
- 如果这一概率比值很小，则表明借鉴策略  $\pi$  在状态  $S_t$  下选择  $A_t$  的机会要小一些，此时为了从借鉴策略学习，则认为这一步状态价值的更新不是很符合借鉴策略，因而在更新时打些折扣。
- 类似的，如果这个概率比值大于 1，说明按照借鉴策略，选择行为  $A_t$  的几率要大于当前行为策略产生  $A_t$  的概率，此时应该对该状态的价值更新就可以大胆些。



# Q-learning: 异策略的TD

- 从上一页的分析中可以看出，选大的似乎就是个不错的主意。
- Q-learning 的伪代码如下：

## Q-learning: An off-policy TD control algorithm

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize  $S$

Repeat (for each step of episode):

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

Take action  $A$ , observe  $R, S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

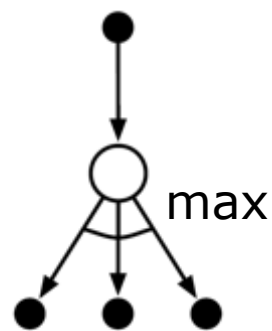
until  $S$  is terminal

直接选Q值最大的动作，是完全贪婪策略，与当前的行动策略无关。

[blog.csdn.net/coffee\\_cream](http://blog.csdn.net/coffee_cream)

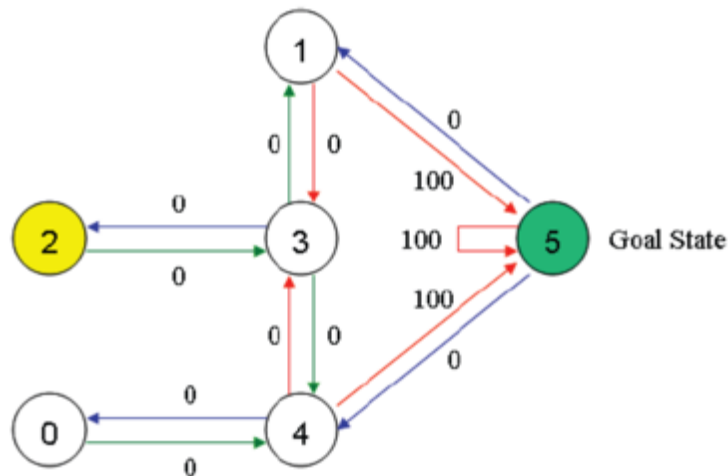
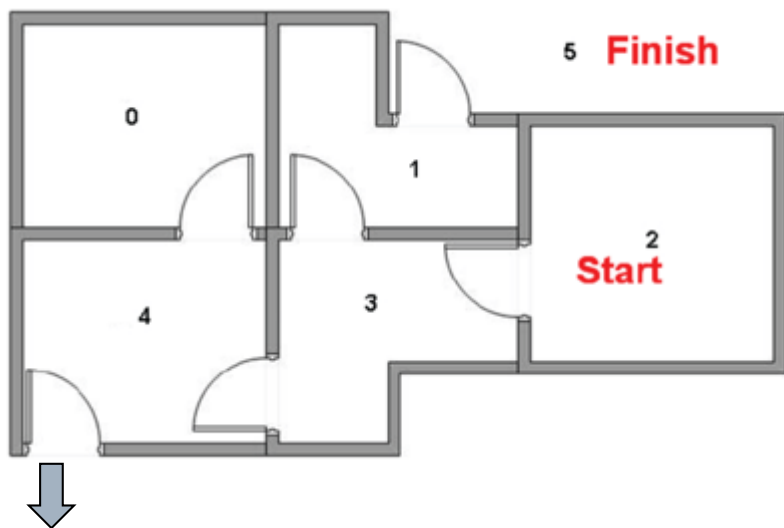
# Q-learning

- Q-learning 的 backup diagram 如图，因为该规则更新的是一个 state-action 对，其顶端是一个 action node，并且该规则是在下个状态所有可能的 actions 中选择 Q 值最大的 action nodes，因此该图的最下面是所有的 action nodes。
- 由于 Q-learning 并不关心 Agent 行动时遵循的是什么 policy，而仅仅是采取最好的 Q-value，其学习的规则与实行的规则不同，因此这是一种 off-policy 学习算法。



Q-learning

# Q-learning的例子



注意：这里出来也是5。

	Action					
State	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

$$\gamma = 0.8$$

# Q-learning的例子

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$Q(\text{state}, \text{action}) = R(\text{state}, \text{action}) + \text{Gamma} \cdot \text{Max}[Q(\text{next state}, \text{all actions})]$$

$$\begin{aligned} Q(1, 5) &= R(1, 5) + 0.8 * \max\{Q(5, 1), Q(5, 4), Q(5, 5)\} \\ &= 100 + 0.8 * \max\{0, 0, 0\} \\ &= 100. \end{aligned}$$

在第一回合，“状态1，去往状态5”的Q值先得到了更新。

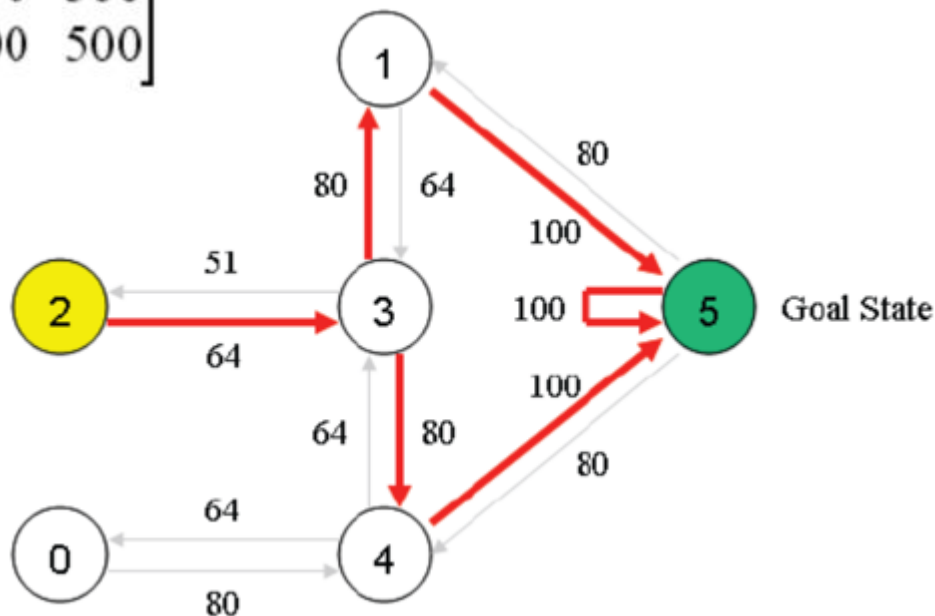
$$\begin{aligned} Q(3, 1) &= R(3, 1) + 0.8 * \max\{Q(1, 3), Q(1, 5)\} \\ &= 0 + 0.8 * \max\{0, 100\} \\ &= 80. \end{aligned}$$

在第一回合，“状态3，去往状态1”的Q值也得到了更新。

# Q-learning的例子

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{matrix}$$

- 最终的结果是归一化之后的。
- 注意：这个例子不体现增量更新！



# Q-learning与SARSA的区别

- Q-learning 是一种 off-policy 的策略，其关键步骤为：

遍历所有动作

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

- 对比一下SARSA：

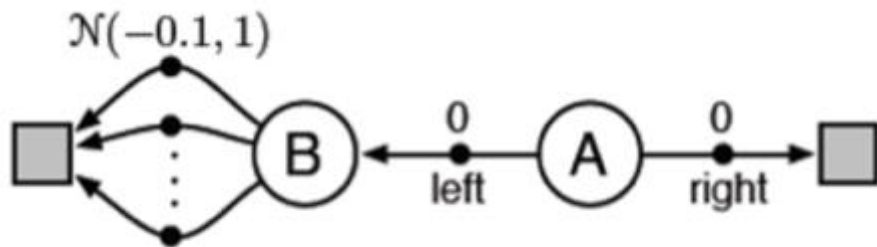
需要A'

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

- Q-learning与SARSA算法最大的不同在于更新Q值的时候，直接使用了最大的 $Q(s_{t+1}, a)$ 值
- 相当于采用了 $Q(s_{t+1}, a)$ 值最大的动作，并且与当前执行的策略，即选取动作 $a_t$ 时采用的策略无关。

# Maximization Bias

- 先看个例子：图中agent从A出发，向右可以直接到达终点，而向B运动，报酬服从均值为-0.1，方差为1的正态分布。
- 由于向B运动时，报酬 $\sim N(-0.1, 1)$ ，则有可能存在大于0的报酬，这样max会使得算法采取向B运动的结果，以期望获得大于0的报酬。这显然不是最佳策略。



# Maximization Bias

---

- ❑ 在 Q-learning 中, target policy 是在当前 action values 中最大化的 greedy policy; 在 Sarsa 中, 常常采用的是  $\epsilon$ -greedy 策略, 其中也包含最大化的操作。
- ❑ 这两种算法, 估计值中的最大值常常作为是最大值的估计, 从而会带来一个显著的 positive bias, 这个偏差就称为是 maximization bias。
- ❑ 可以先考虑一下 maximization bias 产生的原因, 它的原因就在于: 在决定最大化的 action 和在对它的 value 进行评估时采用的是相同的 采样方式。
- ❑ 为了避免 maximization bias, 常采用的策略是 double learning。



# Double Learning

- double learning 同时学习两个独立的估计。
  - 假设它们分别表示为  $Q_1(a)$  和  $Q_2(a)$  ( $a \in A$ ) ,
  - 估计值  $Q_1(a)$  利用  $A^* = \operatorname{argmax}_a Q_1(a)$  来确定最大化的行为,
  - 估计值  $Q_2(a)$  利用  $Q_2(A^*) = Q_2(\operatorname{argmax}_a Q_1(a))$  来进行估计,
- 由  $E[Q_2(A^*)] = q(A^*)$  可知这是一个无偏估计, 可以通过反转两个估计的角色来获得第二个无偏估计  $Q_1(\operatorname{argmax}_a Q_2(a))$ ,
- 虽然它需要两倍的内存, 但每步中的计算量并没有增加。

# Double Q-Learning

- double learning 的思想可以与 Q-learning 相结合就产生了 Double Q-learning, 其更新方程为:

## Double Q-learning

Initialize  $Q_1(s, a)$  and  $Q_2(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily

Initialize  $Q_1(\text{terminal-state}, \cdot) = Q_2(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize  $S$

Repeat (for each step of episode):

Choose  $A$  from  $S$  using policy derived from  $Q_1$  and  $Q_2$  (e.g.,  $\epsilon$ -greedy in  $Q_1 + Q_2$ )

Take action  $A$ , observe  $R, S'$

With 0.5 probability:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left( R + \gamma Q_2(S', \arg\max_a Q_1(S', a)) - Q_1(S, A) \right)$$

else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left( R + \gamma Q_1(S', \arg\max_a Q_2(S', a)) - Q_2(S, A) \right)$$

$S \leftarrow S'$

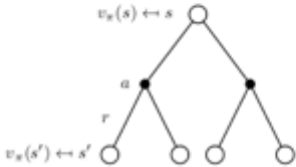

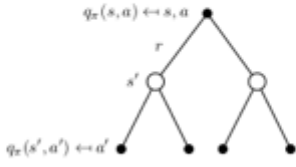
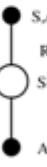
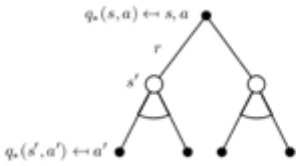
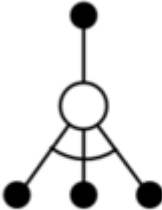
until  $S$  is terminal

一个选动作, 另一个作为更新。

[http://blog.csdn.net/coffee\\_cream](http://blog.csdn.net/coffee_cream)

# 总结一下

## Relationship Between DP and TD

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $v_{\pi}(s)$	 <p>Iterative Policy Evaluation</p>	 <p>TD Learning</p>
Bellman Expectation Equation for $q_{\pi}(s, a)$	 <p>Q-Policy Iteration</p>	 <p>Sarsa</p>
Bellman Optimality Equation for $q_{*}(s, a)$	 <p>Q-Value Iteration</p>	 <p>Q-Learning</p>

# 参考资料

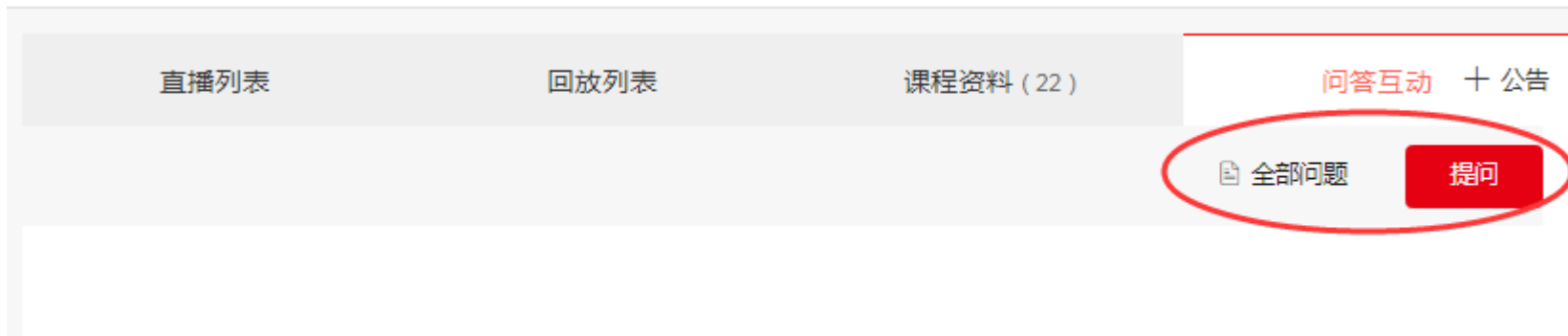
---

- [https://blog.csdn.net/coffee\\_cream/article/details/70194456](https://blog.csdn.net/coffee_cream/article/details/70194456)
- [https://blog.csdn.net/weixin\\_37904412/article/details/81025539](https://blog.csdn.net/weixin_37904412/article/details/81025539)

# 问答互动

在所报课的课程页面，

- 1、点击“全部问题”显示本课程所有学员提问的问题。
- 2、点击“提问”即可向该课程的老师 and 助教提问问题。



# 联系我们

---

小象学院：互联网新技术在线教育领航者

— 微信公众号：**小象学院**

