

时序差分学习 (Temporal-Difference Learning)

吉建民

USTC

`jianmin@ustc.edu.cn`

2023 年 10 月 16 日

Used Materials

Disclaimer: 本课件大量采用了 Rich Sutton's RL class, David Silver's Deep RL tutorial 和其他网络课程课件, 也采用了 GitHub 中开源代码, 以及部分网络博客内容

Table of Contents

课程回顾

背景

时序差分预测 (TD Prediction)

TD(λ)

时序差分控制 (TD Control)

Sarsa: on-policy

Q-learning : off-policy

MDP and Bellman Equations

- ▶ MDP 模型是一个四元组 $\langle S, A, T, R \rangle$
 - ▶ $T(s, a, s') = P(s' \mid s, a)$
 - ▶ Policy: $\pi : S \times A \rightarrow [0, 1]$, $\pi(a \mid s)$
 - ▶ $V^*(s) = \max_{\pi} V_{\pi}(s)$, $Q^*(s, a) = \max_{\pi} Q_{\pi}(s, a)$
- ▶ Bellman Equations:

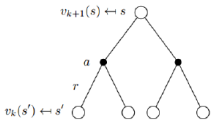
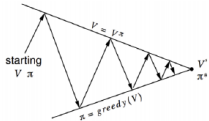
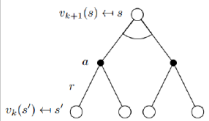
$$V_{\pi}(s) = \sum_{a \in A} \pi(a \mid s) \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{\pi}(s') \right)$$

$$Q_{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \sum_{a' \in A} \pi(a' \mid s') Q_{\pi}(s', a')$$

$$V^*(s) = \max_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right)$$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a' \in A} Q^*(s', a')$$

Dynamic Programming

Algorithm	Iterative Policy Evaluation	Policy Iteration	Value Iteration
			
Bellman Equation	Bellman Expectation Equation	Bellman Expectation Equation Policy Iteration + Greedy Policy Improvement	Bellman Optimality Equation
Problem	Prediction	Control	Control

Monte Carlo Methods

- ▶ 蒙特卡洛方法是一种 Model-free Reinforcement Learning 方法
- ▶ 蒙特卡洛预测：在固定策略 π 下，通过与环境交互的到一系列 episodes，从中提取回报 (return G_t)，根据平均值估计值函数

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t^a - Q(S_t, A_t))$$

- ▶ Exploring Starts Assumption: 所有可能 (s, a) 都经历到
- ▶ 蒙特卡洛控制 On-policy: 生成 episode 的策略和迭代优化的策略是同一个，并且这个策略是一种软 (soft) 策略，即 $\pi(a | s) > 0$ for all $s \in S, a \in A$
 - ▶ ϵ -Greedy Policy Exploration

$$\pi(a | s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \operatorname{argmax}_{a \in A} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

- ▶ UCB (Upper Confidence Bound)

$$a_t = \operatorname{argmax}_{a \in A} \left(Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right)$$

Monte Carlo Methods (con't)

- ▶ 蒙特卡洛控制 Off-policy: 分别在策略估计和策略提升的时候使用两种策略, 一个具有探索性的策略专门用于产生 episode 积累经验, 称为 behavior policy μ , 另一个则是更为贪婪, 用来学习成为最优策略的 target policy π 。要求策略 π 下的所有行为在策略 μ 下被执行过, 也就是要求对所有满足 $\pi(a | s) > 0$ 的 (s, a) 均有 $\mu(a | s) > 0$
 - ▶ 重要性采样 (Importance Sampling)

importance sampling:

$$\begin{array}{ccc} E[f] = \int_x p(x) f(x) dx & = & \int_x q(x) \frac{p(x)}{q(x)} f(x) dx \\ \downarrow \text{sample from } p & & \downarrow \text{sample from } q \\ \frac{1}{m} \sum_{i=1}^m f(x) & & \frac{1}{m} \sum_{i=1}^m \frac{p(x)}{q(x)} f(x) \end{array}$$

- ▶ 加权重要性采样 (Weighted importance sampling)

$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}} \quad V_{n+1} = V_n + \frac{W_n}{C_n} (G_n - V_n)$$
$$C_{n+1} = C_n + W_{n+1}$$

Table of Contents

课程回顾

背景

时序差分预测 (TD Prediction)

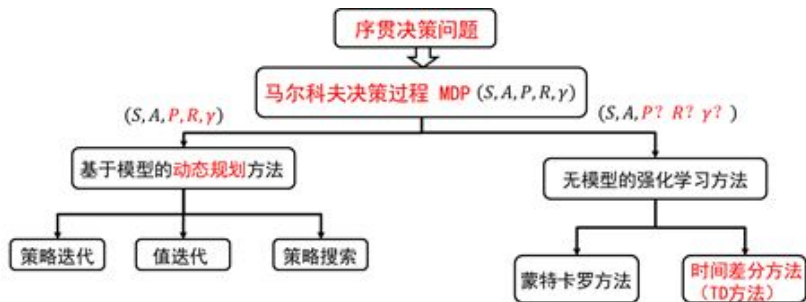
TD(λ)

时序差分控制 (TD Control)

Sarsa: on-policy

Q-learning : off-policy

强化学习方法



时序差分学习 (Temporal-Difference Learning)

时序差分学习 (TD): 融合了 MC 和 DP 的思想

- ▶ TD 像 MC 那样直接从 Episode 学习, 不需要了解模型本身, 是 model-free 方法
- ▶ TD 像 DP 那样, 估计一个状态可以基于已经学习过的状态值, 即 bootstrapping, 而 MC 必须等到一个 episode 结束才能进行学习
- ▶ 它可以学习不完整的 episode, 通过自身的引导 (bootstrapping), 猜测 episode 的结果, 同时持续更新这个猜测

蒙特卡洛方法回顾

- ▶ 回顾蒙特卡洛法中计算状态回报 (return) 的方法是：

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

- ▶ 蒙特卡罗策略评估使用多个 episodes 的平均获得值来代替状态值函数

$$V_{\pi}(s) = E_{\pi}(G_t \mid S_t = s) \approx \frac{\sum_{i: S_t^i = s} G_t^i}{N}$$

- ▶ 把上述公式写成增量式的公式

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

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

- ▶ 更通用表达形式 (可以处理不稳定环境)

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

时序差分学习 (Temporal-Difference Learning)

- ▶ 而对于时序差分法来说，我们没有完整的状态序列，只有部分的状态序列，那么如何可以近似求出某个状态的收获呢？
- ▶ 回顾 Bellman Equations 推导过程中，值函数的展开

$$\begin{aligned}V_{\pi}(s) &= E_{\pi}(G_t \mid S_t = s) \\&= E_{\pi}(R_{t+1} + \gamma G_{t+1} \mid S_t = s) \\&= E_{\pi}(R_{t+1} + \gamma V_{\pi}(S_{t+1}) \mid S_t = s) \\Q_{\pi}(s, a) &= E_{\pi}(G_t \mid S_t = s, A_t = a) \\&= E_{\pi}(R_{t+1} + \gamma Q_{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a)\end{aligned}$$

- ▶ 这启发我们可以用 $R_{t+1} + \gamma V(S_{t+1})$ 来近似的代替收获 G_t
 - ▶ 我们只需要两个连续的状态与对应的奖励，就可以尝试求解强化学习问题了

Table of Contents

课程回顾

背景

时序差分预测 (TD Prediction)

TD(λ)

时序差分控制 (TD Control)

Sarsa: on-policy

Q-learning : off-policy

时序差分预测 (TD Prediction): TD(0)

- ▶ 目标：在一个固定的策略 π 下，从一系列不完整的 episodes 中学习该策略下的状态价值函数 $V_{\pi}(s)$
- ▶ 回顾蒙特卡罗法的迭代式子是：

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

- ▶ 时序差分在预测时，用 $R_{t+1} + \gamma V(S_{t+1})$ 来估计回报 G_t

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

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

- ▶ $R_{t+1} + \gamma V(S_{t+1})$ 称为 TD 目标值 (target)
- ▶ $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ 称为 TD 误差 (error)
- ▶ 将用 TD 目标值近似代替收获 G_t 的过程称为引导 (BootStrapping)
- ▶ MC 每次更新都需要等到 agent 到达终点之后再更新；而对于 TD learning 来说，agent 每走一步它都可以更新一次，不需要等到到达终点之后才进行更新

TD(0) 算法

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

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

Loop for each episode:

 Initialize S

 Loop for each step of episode:

$A \leftarrow$ action given by π 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

示例——驾车返家

- ▶ 用“驾车返家”例子直观解释 MC 策略评估和 TD 策略评估的差别
- ▶ 想象一下你下班后开车回家，需要预估整个行程花费的时间
 - ▶ 假如一个人在驾车回家的路上突然碰到险情：对面迎来一辆车感觉要和你相撞，严重的话他可能面临死亡威胁，但是最后双方都采取了措施没有实际发生碰撞
 - ▶ 如果使用 MC 学习，路上发生的这一险情可能引发的负向奖励不会被考虑进去，不会影响总的预测耗时
 - ▶ 但是在 TD 学习时，碰到这样的险情，这个人会立即更新这个状态的价值，随后会发现这比之前的状态要糟糕，会立即考虑决策降低速度赢得时间
 - ▶ 也就是说你不必像 MC 学习那样直到他死亡后才更新状态价值，那种情况下也无法更新状态价值
- ▶ TD 算法相当于在整个返家的过程中（一个 Episode），根据已经消耗的时间和预期还需要的时间来不断更新最终回家需要消耗的时间

示例——驾车返家 (con't)

状态	已消耗时间 (分)	预计仍需耗 时 (分)	预测总耗时 (分)
离开办公室	0	30	30
取车, 发现下雨	5	35	40
离开高速公路	20	15	35
被迫跟在卡车后面	30	10	40
到达家所在街区	40	3	43
进入家门	43	0	43

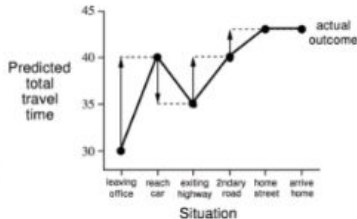
示例——驾车返家 (con't)

MC 学习和 TD 学习两种不同的学习策略来更新价值函数（各个状态的价值）

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



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



示例——驾车返家 (con't)

- ▶ 使用的是从某个状态预估的到家还需耗时来间接反映某状态的价值：
 - ▶ 某位置预估的到家时间越长，该位置价值越低，在优化决策时需要避免进入该状态
- ▶ 对于 MC 学习过程，驾驶员在路面上碰到各种情况时，他不会更新对于回家的预估时间，等他回到家得到了真实回家耗时后，他会重新估计在返家的路上着每一个主要节点状态到家的时间，在下次返家的时候用新估计的时间来帮助决策
- ▶ 对于 TD 学习，在一开始离开办公室的时候你可能会预估总耗时 30 分钟，但是当你取到车发现下雨的时候，你会立刻想到原来的预计过于乐观，因为既往的经验告诉你下雨会延长你的返家总时间，此时你会更新目前的状态价值估计，从原来的 30 分钟提高到 40 分钟。同样当你驾车离开高速公路时，会一路根据当前的状态（位置、路况等）对应的预估返家剩余时间，直到返回家门得到实际的返家总耗时。这一过程中，你会根据状态的变化实时更新该状态的价值

TD vs. MC (1)

- ▶ TD 不需要等到 episode 结束才学习
 - ▶ TD 可以在线更新，每一步后都更新
 - ▶ MC 必须等到一个 episode 结束后，才能更新
- ▶ TD 可以在没有终止状态的环境下学习
 - ▶ TD 可以从不完整的 episode 中学习
 - ▶ MC 只能从完整的 episode 中学习
 - ▶ TD 可以在 continuing（无终止状态）的环境中学习
 - ▶ MC 只能在 episodic（有终止状态）的环境中学习

偏差 (Bias) 和方差 (Variance)

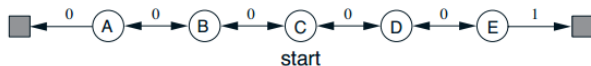
- ▶ 回报 (return) $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$ 是对 $V_\pi(S_t)$ 的无偏估计 (unbiased estimate)
 - ▶ 因为其期望便是值函数的定义
- ▶ 真实的 TD target $R_{t+1} + \gamma V_\pi(S_{t+1})$ 是对 $V_\pi(S_t)$ 的无偏估计
- ▶ TD target $R_{t+1} + \gamma V(S_{t+1})$ 是对 $V_\pi(S_t)$ 的有偏估计 (biased estimate)
 - ▶ 若 $V(S_{t+1})$ 采用真实值, 则 TD 估计也是无偏估计, 然而在试验中 $V(S_{t+1})$ 用的也是估计值, 因此时间差分估计方法属于有偏估计
- ▶ TD target 比回报 (return) 的方差 (Variance) 更小
 - ▶ 回报 G_t 的计算依赖更多的随机动作、环境状态转移和效用
 - ▶ TD target 的计算只依赖一次随机动作、环境状态转移和效用
 - ▶ TD target 的随机性比回报 G_t 要小, 因此其方差也比蒙特卡罗方法的方差小

TD vs. MC (2)

- ▶ MC 没有偏差 (Bias), 但有着较高的方差 (Variance)
 - ▶ 更好的收敛性质 (even with function approximation)
 - ▶ 对初始值不太敏感
 - ▶ 使用简单
- ▶ TD 较低的方差, 但有一定程度的偏差
 - ▶ 通常比 MC 更高效
 - ▶ TD(0) 收敛到 $V_{\pi}(s)$ (but not always with function approximation)
 - ▶ 对初始值更加敏感

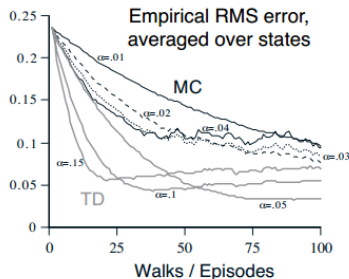
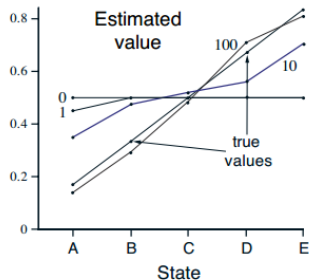
示例——随机行走

- ▶ 状态空间：如下图：A、B、C、D、E 为中间状态，C 同时作为起始状态。灰色方格表示终止状态；



- ▶ 行为空间：除终止状态外，任一状态可以选择向左、向右两个行为之一；
- ▶ 即时奖励：右侧的终止状态得到即时奖励为 1，左侧终止状态得到的即时奖励为 0，在其他状态间转化得到的即时奖励是 0；
- ▶ 状态转移：100% 按行为进行状态转移，进入终止状态即终止；
- ▶ 衰减系数：1；
- ▶ 给定的策略：随机选择向左、向右两个行为。

示例——随机行走 (con't)



- ▶ 左图，分别显示了 TD(0) 在 0、1、10、100 个 episodes 学习的结果
- ▶ 右图，显示了 TD 方法比 MC 更快地收敛到真实值

TD vs. MC (3)

- ▶ MC 和 TD 谁学的更快？这还是一个 open 问题
- ▶ 但实践中，TD 学的更快

示例——AB

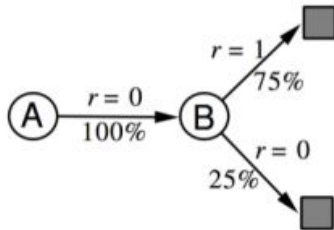
- 已知：现有两个状态 (A 和 B)，MDP 未知，衰减系数为 1，有如下表所示 8 个完整 Episode 的经验及对应的即时奖励，其中除了第 1 个 Episode 有状态转移外，其余 7 个均只有一个状态。

Episode	状态转移及奖励
1	A:0, B:0
2	B:1
3	B:1
4	B:1
5	B:1
6	B:1
7	B:1
8	B:0

- 问题：依据仅有的 Episode，计算状态 A，B 的价值分别是多少，即 $V(A) = ?$, $V(B) = ?$

示例——AB (con't)

- ▶ 答案: $V(B) = 6/8$, $V(A)$ 根据不同算法结果不同, 用 MC 算法结果为 0, TD 则得出 $6/8$
- ▶ 应用 MC 算法, 由于需要完整的 Episode, 因此仅 Episode 1 可以用来计算 A 的状态价值, 很明显是 0; 同时 B 的价值是 $6/8$ 。应用 TD 算法时, TD 算法试图利用现有的 Episode 经验构建一个 MDP (如下图), 由于存在一个 Episode 使得状态 A 有后继状态 B, 因此状态 A 的价值是通过状态 B 的价值来计算的, 同时经验表明 A 到 B 的转移概率是 100%, 且 A 状态的即时奖励是 0, 并且没有衰减, 因此 A 的状态价值等于 B 的状态价值



示例——AB 进一步解释

- ▶ MC 算法试图收敛至一个能够最小化状态价值与实际收获的均方差的解决方案，这一均方差用公式表示为：

$$\sum_{k=1}^K \sum_{t=1}^{T_k} \left(G_t^k - V(S_t^k) \right)^2$$

其中， k 表示的是 Episode 序号， K 为总的 Episode 数量， t 为一个 Episode 内状态序号（第 1, 2, 3, ... 个状态等）， T_k 表示的是第 k 个 Episode 总的状态数， G_t^k 表示第 k 个 Episode 里 t 时刻状态 S_t 获得的最终收获， $V(S_t^k)$ 表示的是第 k 个 Episode 里算法估计的 t 时刻状态 S_t 的价值

- ▶ 在 AB 例子中， $V(A) = 0$

示例——AB 进一步解释 (con't)

- ▶ TD 算法则收敛至一个根据已有经验构建的最大可能的马尔科夫模型的状态价值，也就是说 TD 算法将首先根据已有经验估计状态间的转移概率：

$$\hat{p}_{s,s'}^a = \frac{1}{N(s,a)} \sum_{k=1}^K \sum_{t=1}^{T_k} \mathbf{1}(S_t^k, A_t^k, S_{t+1}^k = s, a, s')$$

同时估计某一个状态的即时奖励：

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s,a)} \sum_{k=1}^K \sum_{t=1}^{T_k} \mathbf{1}(S_t^k, A_t^k = s, a) R_t^k$$

最后计算该 MDP 的状态函数

- ▶ 在 AB 例子中， $V(A) = 0.75$

TD vs. MC (4)

- ▶ MC 方法并不利用马尔科夫性质，故在非马尔科夫环境中更有效率
- ▶ TD(0) 利用马尔科夫性质，在马尔科夫环境中更有效率

TD vs. MC

Monte-Carlo	Temporal Difference
要等到 episode 结束才能获得 return	每一步执行完都能获得一个 return
只能使用完整的 episode	可以使用不完整的 episode
高 variance, 零 bias	低 variance, 有 bias
没有体现出马尔科夫性质	体现出了马尔科夫性质

三种强化学习算法

- ▶ Monte-Carlo, Temporal-Difference 和 Dynamic Programming 都是计算状态价值的一种方法，区别在于，前两种是在不知道 Model 的情况下的常用方法，这其中又以 MC 方法需要一个完整的 Episode 来更新状态价值，TD 则不需要完整的 Episode；DP 方法则是基于 Model 的计算状态价值的方法，它通过计算一个状态 s 所有可能的转移状态 s' 及其转移概率以及对应的即时奖励来计算这个状态 s 的价值
- ▶ 关于是否 Bootstrap：MC 没有引导数据，只使用实际收获；DP 和 TD 都有引导数据
- ▶ 关于是否用样本来计算：MC 和 TD 都是应用样本来估计实际的价值函数；而 DP 则是利用模型直接计算得到实际价值函数，没有样本或采样之说

DP, MC, TD 的异同

我们从原始公式给出动态规划 (DP), 蒙特卡罗方法 (MC), 和时间差分方法 (TD) 的不同之处

$$\begin{aligned} v_{\pi}(s) &= E_{\pi}[G_t | S_t = s] \\ &= E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t,k+1} | S_t = s\right] \\ &= E_{\pi}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t,k+2} | S_t = s\right] \\ &= E_{\pi}\left[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s\right] \end{aligned}$$

MC: 利用采样平均回报逼近期望

TD: 联合了MC和DP, 采样期望值, 并利用真值的当前估计值 $V(S_{t+1})$

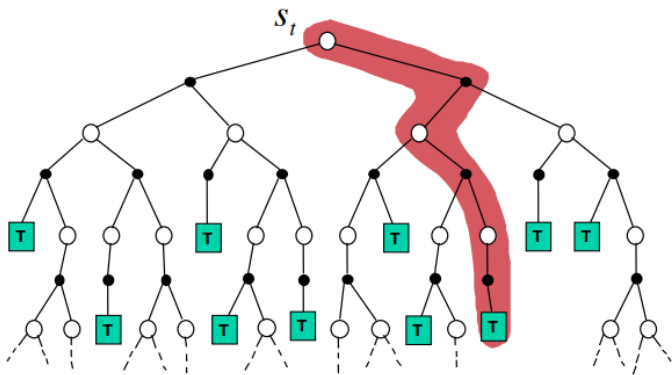
DP: 期望值由模型来提供, 但是利用真值的当前估计值 $V(S_{t+1})$

从图中可以看到, 蒙特卡罗的方法使用的是值函数最原始的定义, 该方法利用所有回报的累积和估计值函数。DP 方法和 TD 方法则利用一步预测方法计算当前状态值函数。其共同点是利用了 bootstrapping 方法, 不同的是, DP 方法利用模型计算后继状态, 而 TD 方法利用试验得到后继状态。

Monte Carlo Backup

MC: 采样，一次完整经历，用实际收获更新状态预估价值

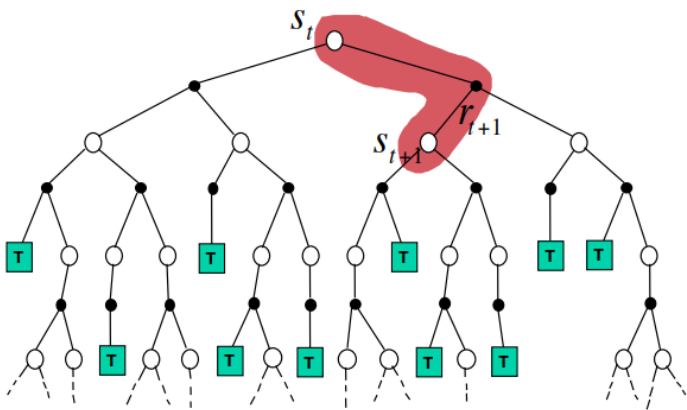
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



Temporal-Difference Backup

TD: 采样, 经历可不完整, 用后继状态的预估状态价值预估收获再更新预估价值

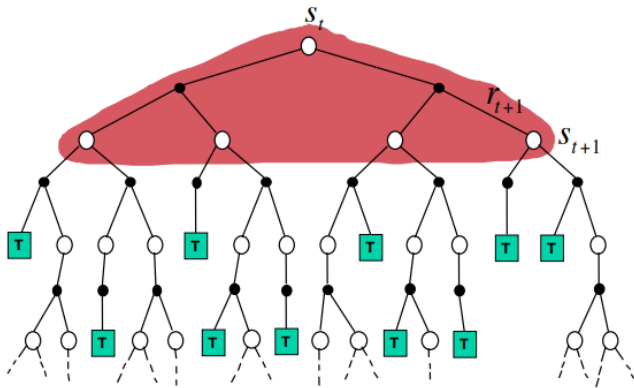
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



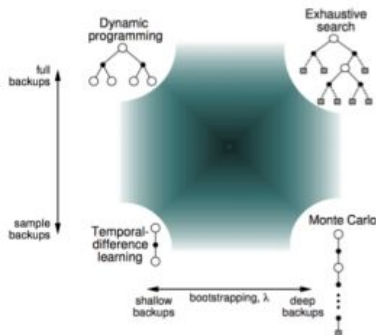
Dynamic Programming Backup

DP: 没有采样，根据完整模型，依靠预估数据更新状态价值

$$V(s) \leftarrow E_{\pi} [R_{t+1} + \gamma V(S_{t+1})]$$



Unified View of Reinforcement Learning



上图从两个维度解释了四种算法的差别，多了一个穷举法。这两个维度分别是：采样深度和广度。当使用单个采样，同时不走完整整个 Episode 就是 TD；当使用单个采样但走完整整个 Episode 就是 MC；当考虑全部样本可能性，但对每一个样本并不走完整整个 Episode 时，就是 DP；当既考虑所有 Episode 又把 Episode 从开始到终止遍历完，就变成了穷举法。

Table of Contents

课程回顾

背景

时序差分预测 (TD Prediction)

TD(λ)

时序差分控制 (TD Control)

Sarsa: on-policy

Q-learning: off-policy

n-step 预测

- ▶ MC 和 TD 方法都过于极端
 - ▶ TD 方法只走一步就更新，它的回报公式如下：

$$G_t = R_{t+1} + \gamma V(s_{t+1})$$

- ▶ tip: 把上式看做 1-step TD target
- ▶ MC 方法需要 episode 走到终止状态才能更新，把它看做，它的回报公式如下：

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

- ▶ tip: 把上式看做 ∞ -step TD target!!

在 1-step TD 和 $+\infty$ -step TD 之间的空间是什么 ??

- ▶ 通常好的方法都是在两个极端之间进行选择...
- ▶ 考虑如下 n-step TD 回报的公式 ($n = 1, 2, 3, \dots, \infty$)

- ▶ $n = 1$ (TD) $G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$
- ▶ $n = 2$ $G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$
- ▶ . .
- ▶ . .
- ▶ . .
- ▶ $n = \infty$ (MC) $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T$

- ▶ 定义 n-step TD 回报公式如下:

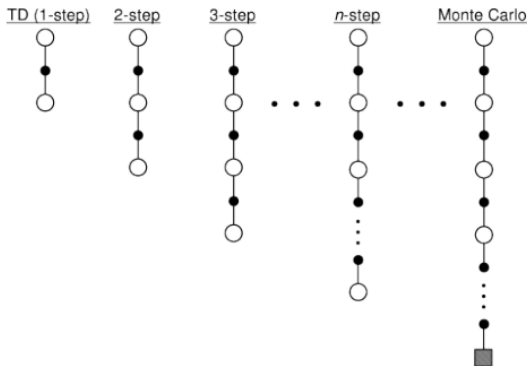
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- ▶ 故 n-step TD 值函数更新公式如下:

$$V(s_t) = V(s_t) + \alpha(G_t^{(n)} - V(s_t))$$

n-step TD 预测

在当前状态往前行动 n 步，计算 n 步的 return，同样 TD target 也由 2 部分组成，已走的步数使用确定的即时 reward，剩下的使用估计的状态价值替代。



注：图中空心大圆圈表示状态，实心小圆圈表示行为

n-step Backups

- ▶ n-step TD:

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

- ▶ n-step backup:

$$\Delta V_t(S_t) = \alpha \left(G_t^{(n)} - V_t(S_t) \right)$$

- ▶ on-line: the updates are done during the episode, as soon as the increment is computed

$$V_{t+1}(s) \leftarrow V_t(s) + \Delta V_t(s)$$

- ▶ off-line: the increments are accumulated “on the side” and are not used to change value estimates until the end of the episode

$$V(s) \leftarrow V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$$

Error reduction property of n-step returns

For any V , the expected value of the n -step return using V is guaranteed to be a better estimate of V^π than V is: the worst error under the new estimate is guaranteed to be less than or equal to γ^n times the worst error under V .

$$\underbrace{\max_s |E_\pi \{R_t^{(n)} \mid s_t = s\} - V^\pi(s)|}_{\text{Maximum error using n-step return}} \leq \gamma^n \underbrace{\max_s |V(s) - V^\pi(s)|}_{\text{Maximum error using } V}$$

Using this, you can show that TD prediction methods using n -step backups converge.

n-step TD prediction algorithm

n-step TD for estimating $V \approx v_\pi$

Input: a policy π

Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n

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

All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:

 Initialize and store $S_0 \neq \text{terminal}$

$T \leftarrow \infty$

 Loop for each step of episode, $t = 0, 1, 2, \dots$:

 If $t < T$, then:

 Take an action according to $\pi(\cdot|S_t)$

 Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

 If S_{t+1} is terminal, then $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)

 If $\tau \geq 0$:

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

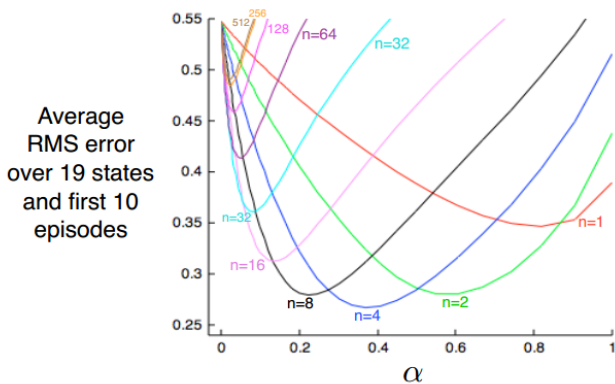
 If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ ($G_{\tau:\tau+n}$)

$V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$

 Until $\tau = T - 1$

n-step TD prediction algorithm

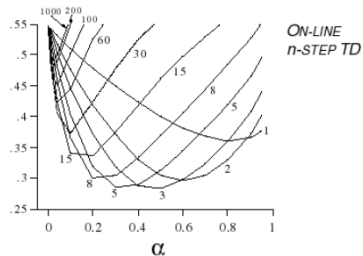
- 下图是利用 n-step TD learning 算法学习之前的 random walk 例子的结果图，可以发现 $n=1$ (TD) 和 $n=\infty$ (MC) 都不是学的最快的，更好的选择是 $n=4$ 时。



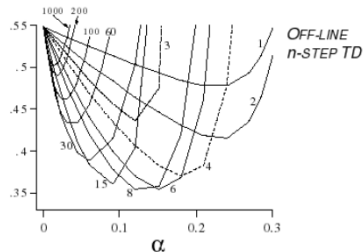
Large Random Walk Example

A larger example:

*RMS error,
averaged over
first 10 episodes*



*RMS error,
averaged over
first 10 episodes*



Task:

19 state random walk

λ -收获 (λ -return)

- ▶ 这里我们引入了一个新的参数： λ 。通过引入这个新的参数，可以做到在不增加计算复杂度的情况下综合考虑所有步数的预测
- ▶ λ -收获 G_t^λ 综合考虑了从 1 到 ∞ 的所有步收获，它给其中的任意一个 n -步收获施加一定的权重 $(1 - \lambda)\lambda^{n-1}$ 。通过这样的权重设计，得到如下的公式：

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

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t^{(T-t)}$$

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

- ▶ 对应的 λ -预测写成 TD(λ):

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

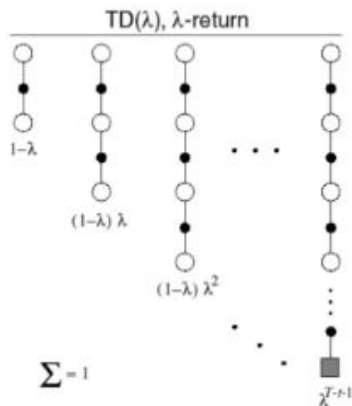
λ -return

- ▶ 我们在 $G_t^{(n)}$ 前乘以加权因子 $(1 - \lambda)\lambda^{n-1}$ 因为

$$\begin{aligned} G_t^\lambda &= (1 - \lambda)G_t^{(1)} + (1 - \lambda)\lambda G_t^{(2)} + \cdots + (1 - \lambda)\lambda^{n-1}G_t^{(n)} + \cdots \\ &\approx [(1 - \lambda) + (1 - \lambda)\lambda + \cdots + (1 - \lambda)\lambda^{n-1} + \cdots]V(S_t) \\ &= (1 - \lambda)\frac{1}{1 - \lambda}V(S_t) = V(S_t) \end{aligned}$$

λ -return

下图是各步收获的权重分配图，图中最后一列 λ 的指数是 $T - t - 1$ 。 T 为终止状态的时刻步数， t 为当前状态的时刻步数，所有的权重加起来为 1。



Relation to TD(0) and MC

- ▶ λ -return 可以写成:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t^{(T-t)}$$

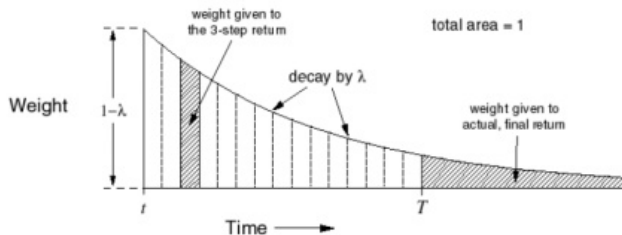
- ▶ 当 $\lambda = 1$, 得到 MC:

$$G_t^\lambda = (1 - 1) \sum_{n=1}^{T-t-1} 1^{n-1} G_t^{(n)} + 1^{T-t-1} G_t^{(T-t)} = G_t^{(T-t)}$$

- ▶ 当 $\lambda = 0$, 得到 TD(0):

$$G_t^\lambda = (1 - 0) \sum_{n=1}^{T-t-1} 0^{n-1} G_t^{(n)} + 0^{T-t-1} G_t^{(T-t)} = G_t^{(1)}$$

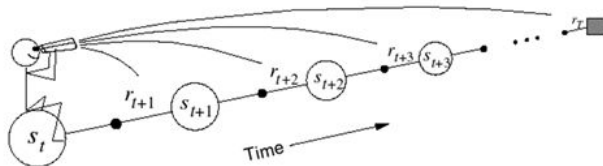
TD(λ) 对于权重分配的图解



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

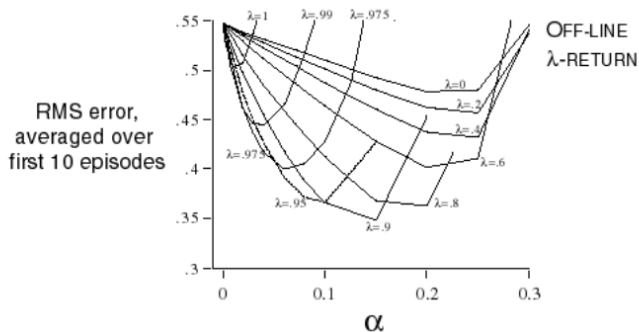
对于 $n = 3$ 的 3-步收获，赋予其在 λ 收获中的权重如左侧阴影部分面积，对于终止状态的 T -步收获， T 以后的所有阴影部分面积。而所有节段面积之和为 1。这种几何级数的设计也考虑了算法实现的计算方便性。

前向 TD(λ)



- ▶ 利用 TD(λ) 的前向观点估计值函数时, G_t^λ 的计算用到了将来时刻的值函数, 因此需要等到整个试验结束之后。这和蒙特卡罗法的要求一样, 因此 TD(λ) 有着和蒙特卡罗法一样的劣势
- ▶ 前向 TD(λ) 构成 off-line 预测算法
- ▶ 前向 TD(λ) 提供理论解释, 在实际应用中也很少
- ▶ 有没有一种更新方法不需要等到试验结束就可以更新当前状态的值函数呢?

λ -return on the Random Walk



Same 19 state random walk as before

示例——被电击的原因

- ▶ 老鼠在连续接受了 3 次响铃和 1 次亮灯信号后遭到了电击，那么在分析遭电击的原因时，到底是响铃的因素较重要还是亮灯的因素更重要呢？



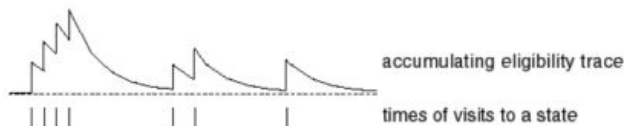
- ▶ 频率启发 (Frequency heuristic): 将原因归因于出现频率最高的状态
- ▶ 就近启发 (Recency heuristic): 将原因归因于较近的几次状态
- ▶ 效用追踪 (Eligibility Traces) 结合两种启发:

$$E_0(s) = \mathbf{1}(S_t = s)$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s) \quad \lambda, \gamma \in [0, 1]$$

$$E_t(s) = \sum_{k=0}^t (\gamma\lambda)^{t-k} \mathbf{1}(S_k = s)$$

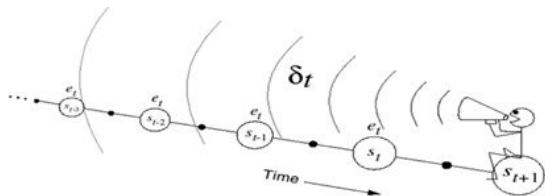
效用追踪 (Eligibility Traces)



- ▶ 上图横坐标是时间，横坐标下有竖线的位置代表当前进入了状态 s ，纵坐标是效用追踪值 E 。可以看出当某一状态连续出现， E 值会在一定衰减的基础上有一个单位数值的提高，此时将增加该状态对于最终收获贡献的比重，因而在更新该状态价值的时候可以较多地考虑最终收获的影响。同时如果该状态距离最终状态较远，则其对最终收获的贡献越小，在更新该状态时也不需要太多的考虑最终收获。
- ▶ E 值并不需要等到完整的 Episode 结束才能计算出来，它可以每经过一个时刻就得到更新。 E 值存在饱和现象，有一个瞬时最高上限：

$$E_{max} = 1/(1 - \gamma\lambda)$$

后向 TD(λ)



- ▶ 对每个状态 s 维护效用追踪 (Eligibility Trace) $E_t(s)$
- ▶ 利用 TD-error δ_t 和 eligibility trace $E_t(s)$ 更新 $V(s)$

$$E_0(s) = \mathbf{1}(S_t = s)$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$

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

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- ▶ 后向 TD(λ) 可以构成 on-line 或 off-line 预测算法

On-line Tabular TD(λ)

Initialize $V(s)$ arbitrarily

Repeat (for each episode):

$e(s) = 0$, for all $s \in S$

Initialize s

Repeat (for each step of episode):

$a \leftarrow$ action given by π for s

Take action a , observe reward, r , and next state s'

$\delta \leftarrow r + \gamma V(s') - V(s)$

$e(s) \leftarrow e(s) + 1$

For all s :

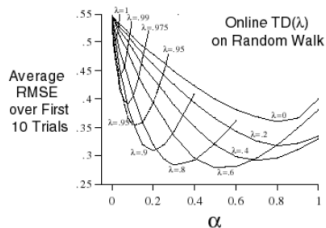
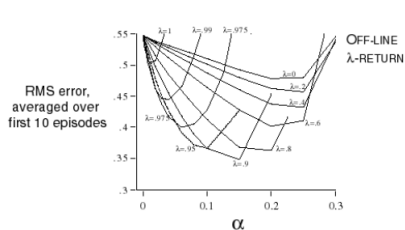
$V(s) \leftarrow V(s) + \alpha \delta e(s)$

$e(s) \leftarrow \gamma \lambda e(s)$

$s \leftarrow s'$

Until s is terminal

On-line vs. Off-line on Random Walk



- Same 19 state random walk
- On-line performs better over a broader range of parameters

后向 TD(0)

- ▶ 回顾 TD-error 定义

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

- ▶ 当 $\lambda = 0$, 只有当前状态进行更新

$$\begin{aligned} E_t(s) &= \mathbf{1}(S_t = s) \\ V(s) &\leftarrow V(s) + \alpha \delta_t E_t(s) \end{aligned}$$

- ▶ 上述结果与 TD(0) 更新等价

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

- ▶ 后向 TD(1) 与 MC 是否等价 ?

前向 TD(λ) 与后向 TD(λ) 等价

$$\begin{aligned} G_t^\lambda - V(S_t) &= -V(S_t) + (1-\lambda)\lambda^0(R_{t+1} + \gamma V(S_{t+1})) \\ &\quad + (1-\lambda)\lambda^1(R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})) \\ &\quad + (1-\lambda)\lambda^2(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})) \\ &\quad + \dots \\ &= -V(S_t) + (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1})) \\ &\quad + (\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2})) \\ &\quad + (\gamma\lambda)^2(R_{t+3} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})) \\ &\quad + \dots \\ &= 0 + (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \\ &\quad + (\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1})) \\ &\quad + (\gamma\lambda)^2(R_{t+3} + \gamma V(S_{t+3}) - V(S_{t+2})) \\ &\quad + \dots \\ &= \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots \approx \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t}\delta_k \end{aligned}$$

前向 TD(λ) 与后向 TD(λ) 等价 (con't)

- ▶ $G_t^\lambda - V(S_t) \approx \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t} \delta_k$
 - ▶ 当采用 off-line 更新时, 对同一 t , 函数 V_t 是不变的, 上式 \approx 是 =
- ▶ 回顾 $E_t(s) = \sum_{k=0}^t (\gamma\lambda)^{t-k} \mathbf{1}(S_k = s)$
- ▶ 考虑一个 episode 从 0 时刻到 T 时刻终止, 采用 off-line 更新方式:
 - ▶ 前向 TD(λ) 对应的更新结果 ($V'(s) - V(s)$) 为

$$\sum_{t=0}^{T-1} \alpha \mathbf{1}(S_t = s) (G_t^\lambda - V(S_t))$$

- ▶ 后向 TD(λ) 对应的更新结果为

$$\sum_{t=0}^{T-1} \alpha \delta_t E_t(s)$$

前向 TD(λ) 与后向 TD(λ) 等价 (con't)

可以证明 $\sum_{t=0}^{T-1} \alpha (G_t^\lambda - V(S_t)) \mathbf{1}(S_t = s) = \sum_{t=0}^{T-1} \alpha \delta_t E_t(s)$

$$\begin{aligned} \sum_{t=0}^{T-1} \alpha (G_t^\lambda - V(S_t)) \mathbf{1}(S_t = s) &= \sum_{t=0}^{T-1} \alpha \mathbf{1}(S_t = s) \sum_{k=t}^{T-1} (\gamma \lambda)^{k-t} \delta_k \\ \sum_{t=0}^{T-1} \alpha \delta_t E_t(s) &= \sum_{t=0}^{T-1} \alpha \delta_t \sum_{k=0}^t (\gamma \lambda)^{t-k} \mathbf{1}(S_k = s) \\ &= \sum_{k=0}^{T-1} \alpha \sum_{t=0}^k (\gamma \lambda)^{k-t} \mathbf{1}(S_t = s) \delta_k \\ &= \sum_{t=0}^{T-1} \alpha \sum_{k=t}^{T-1} (\gamma \lambda)^{k-t} \mathbf{1}(S_t = s) \delta_k \\ &= \sum_{t=0}^{T-1} \alpha \mathbf{1}(S_t = s) \sum_{k=t}^{T-1} (\gamma \lambda)^{k-t} \delta_k \end{aligned}$$

后向 TD(1) 与 MC

- ▶ 当 $\lambda = 1$, 后向 TD(1) 更新公式为

$$E_t(s) = \sum_{k=0}^t \gamma^{t-k} \mathbf{1}(S_k = s)$$
$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- ▶ 回顾 MC 更新公式

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1-t} R_T$$

后向 TD(1) 与 MC (con't)

- 已知 off-line 情况下，前向 TD(λ) 与后向 TD(λ) 等价

$$\begin{aligned} G_t - V(S_t) &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \\ \sum_{t=0}^{T-1} \alpha (G_t - V(S_t)) \mathbf{1}(S_t = s) &= \sum_{t=0}^{T-1} \alpha \delta_t \sum_{k=0}^t \gamma^{t-k} \mathbf{1}(S_k = s) \\ &= \sum_{t=0}^{T-1} \alpha \left(\sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \right) \mathbf{1}(S_t = s) \end{aligned}$$

- 粗略看，后向 TD(1) 与 every-visit MC 等价 (off-line 情况下严格等价)
- 后向 TD(1) 支持 on-line 算法，在线更新时，状态价值差每一步都会有积累

前向与后向 TD(λ)

- ▶ 前向观点需要等到一次试验之后再更新当前状态的值函数；而后向观点不需要等到值函数结束后再更新值函数，而是每一步都在更新值函数，是增量式方法
- ▶ 前向观点在一次试验结束后更新值函数时，更新完当前状态的值函数后，此状态的值函数就不再改变。而后向观点，在每一步计算完当前的 TD 误差后，其他状态的值函数需要利用当前状态的 TD 误差进行更新
- ▶ 在一次试验结束后（off-line 方式），前向观点和后向观点每个状态的值函数的更新总量是相等的，都是 G_t^λ

Summary of Forward and Backward TD(λ)

下表给出了 取各种值时，不同算法在不同情况下的关系

Offline updates	$\lambda = 0$	$\lambda \in (0, 1)$	$\lambda = 1$
Backward view	TD(0) 	TD(λ) 	TD(1)
Forward view	TD(0)	Forward TD(λ)	MC
Online updates	$\lambda = 0$	$\lambda \in (0, 1)$	$\lambda = 1$
Backward view	TD(0) 	TD(λ) \nparallel	TD(1) \nparallel
Forward view	TD(0) 	Forward TD(λ) 	MC
Exact Online	TD(0)	Exact Online TD(λ)	Exact Online TD(1)

相较于 MC 算法，TD 算法应用更广，是一个非常有用的强化学习方法

Exact online TD(λ) 由 Sutton and von Seijen, ICML 2014 给出证明

Table of Contents

课程回顾

背景

时序差分预测 (TD Prediction)

TD(λ)

时序差分控制 (TD Control)

Sarsa: on-policy

Q-learning: off-policy

MC vs. TD Control

- ▶ TD learning 比 MC 有多种好处:
 - ▶ 低方差
 - ▶ 在线算法
 - ▶ 可以处理不完全 episode 情况, 应用连续学习问题
- ▶ TD control: 使用 TD 替换 MC 构造 Control 算法
 - ▶ 采用 TD Prediction 估计 $Q(S, A)$
 - ▶ 采用 ϵ -greedy policy improvement
 - ▶ 每个 time-step 进行更新

Table of Contents

课程回顾

背景

时序差分预测 (TD Prediction)

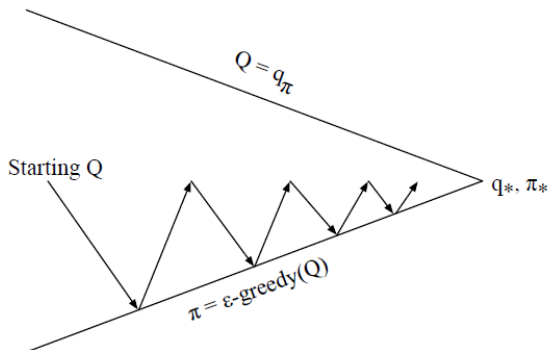
TD(λ)

时序差分控制 (TD Control)

Sarsa: on-policy

Q-learning : off-policy

On-policy Control with Sarsa



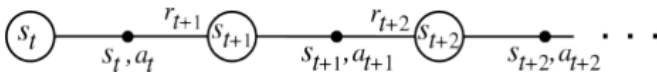
Every **time-step**:

Policy evaluation **Sarsa**, $Q \approx q_{\pi}$

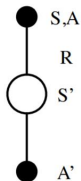
Policy improvement ϵ -greedy policy improvement

Sarsa 算法

- ▶ 现在我们讨论把 TD 预测的方法用于解决控制问题。
 - ▶ 首先，像 MC 那样，遵循 GPI 的框架，即先做策略估计，然后做策略提升。
 - ▶ 然后，把 MC 方法里面的 G_t 改为 TD target = $R_{t+1} + \gamma V(S_{t+1})$
 - ▶ 最后，我们像 MC 那样讨论 on-policy 和 off-policy, 现在先讨论 on-policy 即 Sarsa
- ▶ 在 Sarsa 中，要学习的是动作值函数，而不是值函数。
 - ▶ 也就是说，在一个策略 π 下，要估计所有状态以及行为的值 $Q_{\pi}(s, a)$ 。
 - ▶ 回想一个包含交替的状态和状态-动作对的序列。



Sarsa 算法



- ▶ 计算 $Q_{\pi}(s, a)$ ，由 TD(0) 的得到如下的公式：

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

- ▶ Sarsa 算法的名字来自 S, A, R, S, A 字母的组合

Sarsa 算法

- ▶ 设计一个基于 Sarsa 预测方法的 on-policy 控制策略如下：
 - ▶ 首先，对策略 π 持续估计 $Q_{\pi}(S, A)$
 - ▶ 然后，利用 ϵ -greedy 提高策略 π

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

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

 Loop for each step of episode:

 Take action A , observe R, S'

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

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

$S \leftarrow S'; A \leftarrow A';$

 until S is terminal

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname{argmax}_{a' \in \mathcal{A}} Q_k(s, a'))$$

- For example, ϵ -greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

Sarsa 算法收敛性

Theorem

Sarsa converges to the optimal action-value function, $Q(s, a) \rightarrow q_(s, a)$, under the following conditions:*

- *GLIE sequence of policies $\pi_t(a|s)$*
- *Robbins-Monro sequence of step-sizes α_t*

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

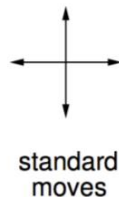
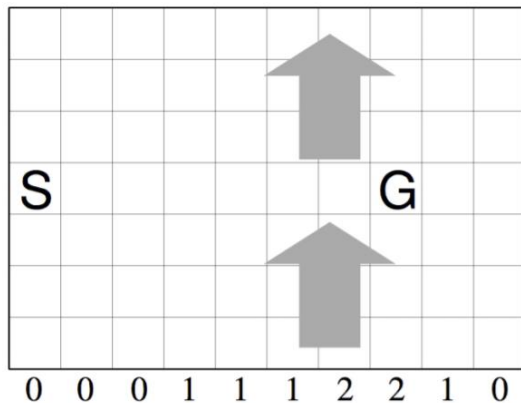
$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

例如: $\alpha_t = \frac{a}{t}$ for some $a > 0$

Sarsa 算法实例: Windy GridWorld

- ▶ 在一个 10×7 的长方形格子世界，标记有一个起始位置 S 和一个终止目标位置 G，格子下方的数字表示对应的列中一定强度的风。
- ▶ 当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数字表示的格数，借此来模拟世界中风的作用。同样格子世界是有边界的，个体任意时刻只能处在世界内部的一个格子中。
- ▶ 个体并不清楚这个世界的构造以及有风，也就是说它不知道格子是长方形的，也不知道边界在哪里，也不知道自己在里面移动移步后下一个格子与之前格子的相对位置关系，当然它也不清楚起始位置、终止目标的具体位置。但是个体会记住曾经经过的格子，下次在进入这个格子时，它能准确的辨认出这个格子曾经什么时候来过。
- ▶ 格子可以执行的行为是朝上、下、左、右移动一步，每移动一步只要不是进入目标位置都给予一个 -1 的惩罚，直至进入目标位置后获得奖励 0 同时永久停留在该位置。
- ▶ 现在要求解的问题是个体应该遵循怎样的策略才能尽快的从起始位置到达目标位置。

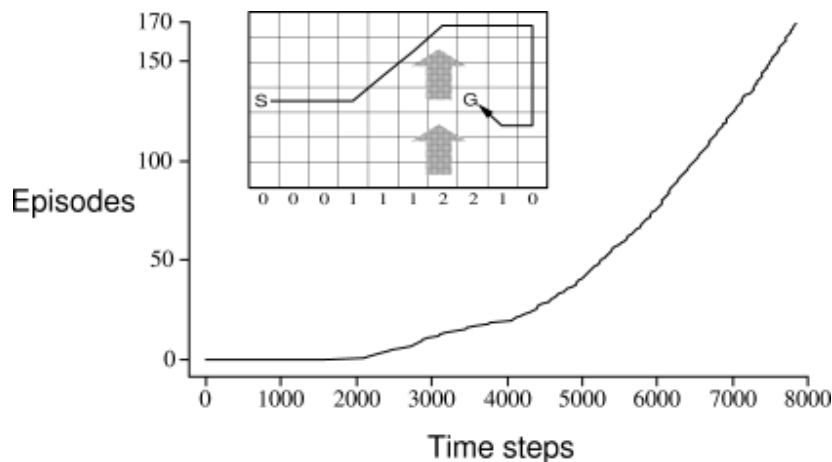
Sarsa 算法实例: Windy GridWorld (con't)



- Reward = -1 per time-step until reaching goal

Sarsa 算法实例: Windy GridWorld (con't)

下图是使用 ϵ -greedy Sarsa 的效果, 其中 $\epsilon = 0.1$, $\alpha = 0.1$, 同时初始值 $Q(s, a) = 0$ for all s, a



MC 方法较困难, 因为有些 policy 在这个例子中是不终止的

Sarsa(λ)

- ▶ Sarsa(λ): 用 TD(λ) 替换 Sarsa 中 TD(0) 来估计策略行动值函数
- ▶ 行动值函数 $Q(s, a)$ 的 n -step 版本

- Consider the following n -step returns for $n = 1, 2, \infty$:

$$n = 1 \quad (\text{Sarsa}) \quad q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1})$$

$$n = 2 \quad q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2})$$

$$\vdots$$
$$\vdots$$

$$n = \infty \quad (MC) \quad q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

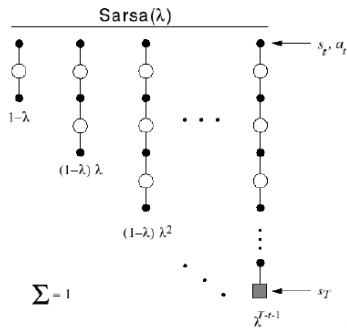
- Define the n -step Q-return

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

- n -step Sarsa updates $Q(s, a)$ towards the n -step Q-return

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

前向 Sarsa(λ)



- The q_t^λ return combines all n -step Q-returns $q_t^{(n)}$
- Using weight $(1 - \lambda)\lambda^{n-1}$

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

- Forward-view Sarsa(λ)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^\lambda - Q(S_t, A_t) \right)$$

后向 Sarsa(λ)

- ▶ 与 TD(λ) 相同，采用 eligibility traces 实现 online 算法
- ▶ 针对 state-action pair 的 eligibility trace

$$E_0(s, a) = \mathbf{1}(S_t = s, A_t = a)$$

$$E_t(s, a) = \gamma\lambda E_{t-1}(s, a) + \mathbf{1}(S_t = s, A_t = a)$$

$$E_t(s, a) = \sum_{k=0}^t (\gamma\lambda)^{t-k} \mathbf{1}(S_t = s, A_t = a)$$

- ▶ 针对 state-action pair (s, a) 更新 $Q(s, a)$ ，更新公式如下：

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

$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)$$

Sarsa(λ) 算法

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

Repeat (for each episode):

$E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Initialize S, A

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)

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$E(S, A) \leftarrow E(S, A) + \delta$

For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

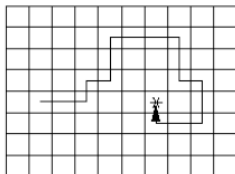
$E(s, a) \leftarrow \gamma \lambda E(s, a) + \delta$

$S \leftarrow S'; A \leftarrow A'$

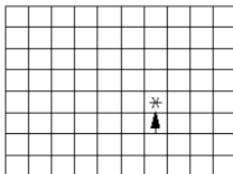
until S is terminal

Sarsa(λ) Windy GridWorld Example

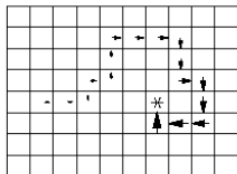
Path taken



Action values increased
by one-step Sarsa



Action values increased
by Sarsa(λ) with $\lambda=0.9$



- With one trial, the agent has much more information about how to get to the goal
 - not necessarily the *best* way
- Can considerably accelerate learning

Sarsa 算法小结

- ▶ SARSA 算法和动态规划法比起来，不需要环境的状态转换模型，和蒙特卡罗法比起来，不需要完整的状态序列，因此比较灵活。在传统的强化学习方法中使用比较广泛。
- ▶ 但是 SARSA 算法也有一个传统强化学习方法共有的问题，就是无法求解太复杂的问题。在 SARSA 算法中， $Q(S, A)$ 的值使用一张大表来存储的，如果我们的状态和动作都达到百万乃至千万级，需要在内存里保存的这张大表会超级大，甚至溢出，因此不是很适合解决规模很大的问题。当然，对于不是特别复杂的问题，使用 SARSA 还是很不错的一种强化学习问题求解方法。

Table of Contents

课程回顾

背景

时序差分预测 (TD Prediction)

TD(λ)

时序差分控制 (TD Control)

Sarsa: on-policy

Q-learning: off-policy

Importance Sampling for Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target $R + \gamma V(S')$ by importance sampling
- Only need a single importance sampling correction

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

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Q-Learning 背景

- ▶ 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

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

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

Off-Policy Control with Q-Learning

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

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

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

$$\pi(A' | S_{t+1}) = \begin{cases} 1 & \text{if } A' = \operatorname{argmax}_{a'} Q(S_{t+1}, a') \\ 0 & \text{otherwise} \end{cases}$$

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

Q-Learning Control

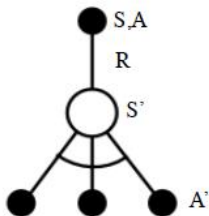
- ▶ 回顾 Sarsa

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

- ▶ Q-Learning

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

- ▶ 上面公式中，直接取最大值 $\max_{a'} Q(S_{t+1}, a')$ 进行学习，这样学习的行为值函数 Q 直接近似到最优 Q^* ；从而不需要重要性采样。



Q-Learning Control (con't)

- ▶ Q-Learning 是 off-policy 算法，因为： $\max_{a'} Q(S_{t+1}, a')$ 中选择的动作只会参与价值函数的更新，不会真正的执行。价值函数更新后，新的执行动作需要基于状态 S_{t+1} ，用 ϵ -greedy 方法重新选择得到。
 - ▶ Q-Learning learns an optimal Q-value function, even if it does not always choose optimal actions.
- ▶ 这一点也和 Sarsa 算法不同。对于 Sarsa，价值函数更新使用的 A' 会作为下一阶段开始时候的执行动作。所以 Sarsa 是 on-policy 算法。

Theorem

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

Q-learning 算法

- ▶ Q-Learning 算法跟 Sarsa 很像，只需改下 Sarsa 的 Q 公式，就可以得到如下 Q-Learning 算法：

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

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

 Take action A , observe R, S'

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

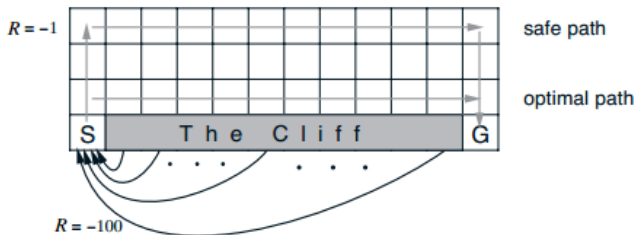
$S \leftarrow S'$

 until S is terminal

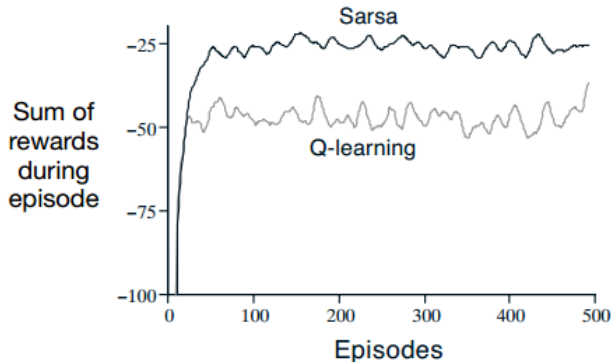
Sarsa 和 Q-learning 对比, 谁学习更快?

► 例子: Cliff Walking

- S 是起点, G 是终点;
- reward: 如果 agent 落入 The Cliff 区域, $R=-100$, 并重置于 S; 在 G 的 $R=0$; 其他区域 $R=-1$



Sarsa 和 Q-learning 对比，谁学习更快？



- ▶ 从上图可知，Q-learning 比 Sarsa 更快
- ▶ 但是，Q-learning 会选择 optimal path，这样掉入 the cliff 的概率更大；而 Sarsa 更倾向于选择 safe path。故 Q-learning 也比 Sarsa 更危险。

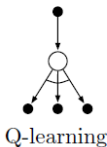
Sarsa 和 Q-learning 对比

- ▶ Q-Learning 直接学习的是最优策略，而 SARSA 在学习最优策略的同时还在做探索。这导致我们在学习最优策略的时候，如果用 SARSA，为了保证收敛，需要制定一个策略，使 ϵ -greedy 方法的超参数 ϵ 在迭代的过程中逐渐变小。Q-Learning 没有这个烦恼。
- ▶ Q-Learning 直接学习最优策略，但是最优策略会依赖于训练中产生的一系列数据，所以受样本数据的影响较大，因此受到训练数据方差的影响很大，甚至会影响 Q 函数的收敛。Q-Learning 的深度强化学习版 Deep Q-Learning 也有这个问题。
- ▶ 在学习过程中，SARSA 在收敛的过程中鼓励探索，这样学习过程会比较平滑，不至于过于激进，导致出现像 Q-Learning 可能遇到一些特殊的最优“陷阱”。比如经典的强化学习问题“Cliff Walk”。
- ▶ 在实际应用中，如果我们是在模拟环境中训练强化学习模型，推荐使用 Q-Learning，如果是在线生产环境中训练模型，则推荐使用 SARSA。

Expected Sarsa

- Instead of the *sample* value-of-next-state, use the expectation!

$$\begin{aligned}Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right] \\&\leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]\end{aligned}$$



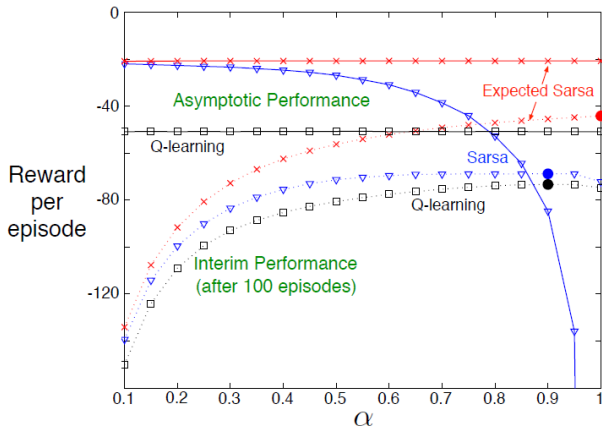
- Expected Sarsa's performs better than Sarsa (but costs more)

Expected Sarsa

Algorithm 1 Expected Sarsa

- 1: Initialize $Q(s, a)$ arbitrarily for all s, a
 - 2: **loop** {over episodes}
 - 3: Initialize s
 - 4: **repeat** {for each step in the episode}
 - 5: choose a from s using policy π derived from Q
 - 6: take action a , observe r and s'
 - 7: $V_{s'} = \sum_a \pi(s', a) \cdot Q(s', a)$
 - 8: $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma V_{s'} - Q(s, a)]$
 - 9: $s \leftarrow s'$
 - 10: **until** s is terminal
 - 11: **end loop**
-

Performance on the Cliff-walking Task



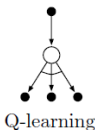
Asymptotic performance is an average over 100,000 episodes, interim performance is an average over the first 100 episodes.

Off-policy Expected Sarsa

- Expected Sarsa generalizes to arbitrary behavior policies μ
 - in which case it includes Q-learning as the special case in which π is the greedy policy



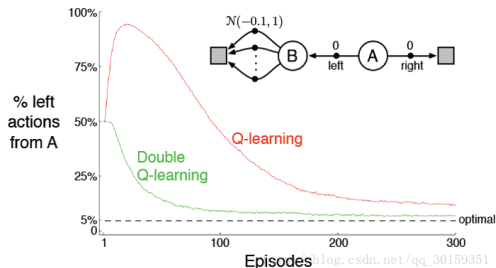
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right]$$
$$\leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$



- ▶ Expected Sarsa can be considered as an on-policy version of Q-learning.
- ▶ When π is the behavior policy, Expected Sarsa is on-policy.

Maximization Bias

- ▶ Q-learning 使用了 \max ，会引起一个最大化偏差 (Maximization Bias, 估计的值函数比真实值要大) 问题。
- ▶ Maximization Bias: 考虑如下的 MDP, 状态 B 到达左边的终态有多个 action, 它们的 reward 是均值为-0.1, 方差为 1 的正态分布。



A 选择 left 的 action 的 reward 期望为-0.1, 向右是 0, 所以不应该选择 left。但是, Q-learning 更倾向于 left, 因为这些 control 算法都有一个 maximization 操作, 有时 B 会产生大于 0 的 reward, 而算法做了 \max 操作, 会让算法觉得 B 状态采取 left 会有正的 value。

Double Q-Learning

- ▶ 为了避免 maximization bias, 我们将 experience 分成两个部分, 用来学习两个 estimates。在对 action value 做估计时, 用其中一个决定最优的行动, 另一个策略决定这个最优行动的值的估计, 这样得到的估计是无偏的。这就是 Double Q-Learning。更新规则为下面两式子各有一半概率更新:

$$\begin{aligned} Q_1(S_t, A_t) &\leftarrow Q_1(S_t, A_t) \\ &\quad + \alpha (R_{t+1} + \gamma Q_2(S_{t+1}, \operatorname{argmax}_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)) \\ Q_2(S_t, A_t) &\leftarrow Q_2(S_t, A_t) \\ &\quad + \alpha (R_{t+1} + \gamma Q_1(S_{t+1}, \operatorname{argmax}_a Q_2(S_{t+1}, a)) - Q_2(S_t, A_t)) \end{aligned}$$

- ▶ 分别独立的学习两个估计, $A^* = \operatorname{argmax}_a Q_1(a)$, $Q_2(A^*) = Q_2(\operatorname{argmax}_a Q_1(a))$; $E[Q_2(A^*)] = q(A^*)$.
- ▶ 虽然我们学习了两个估计, 但是每个 step 中, 只更新其中一个估计。Double Q-Learning 用了双倍的内存, 但是计算次数没有变。

Double Q-Learning 算法

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, such that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

Initialize S

Loop for each step of episode:

Choose A from S using the policy ε -greedy in $Q_1 + Q_2$

Take action A , observe R, S'

With 0.5 probability:

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

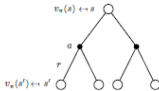

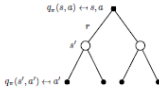

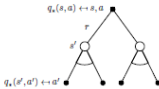
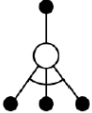
else:

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

$S \leftarrow S'$

until S is terminal

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>

Relationship Between DP and TD

<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	TD Learning $V(S) \stackrel{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration $Q(s, a) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$	Sarsa $Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma Q(S', A')$
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 $Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$

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

$Q(\lambda)$

- ▶ A problem occurs for off-policy methods such as Q-learning when exploratory actions occur, since you backup over a non-greedy policy. This would violate GPI.
 - ▶ Q-learning learns about the greedy policy while it typically follows a policy involving exploratory actions—occasional selections of actions that are suboptimal according to Q_t .
 - ▶ Suppose that on the next two time steps the agent selects the greedy action, but on the third, at time $t+3$, the agent selects an exploratory, nongreedy action. We can use the one-step and two-step returns, but not, in this case, the three-step return. The n -step returns for all $n \geq 3$ no longer have any necessary relationship to the greedy policy.

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, A_{t+2}, R_{t+3}, S_{t+3}, \dots$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(\frac{\pi(A_{t+1} | S_{t+1}) \pi(A' | S_{t+2})}{\mu(A_{t+1} | S_{t+1}) \mu(A' | S_{t+2})} (R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A')) - Q(S_t, A_t) \right)$$

► Three approaches to $Q(\lambda)$:

- **Watkins**: Zero out eligibility trace after a non-greedy action.
Do max when backing up at first non-greedy choice (a_{t+n}).

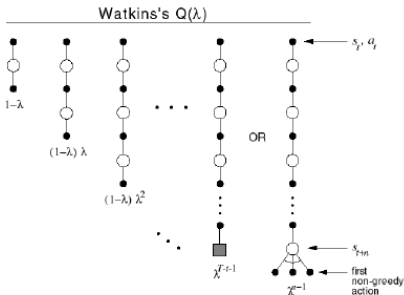
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \max_a Q_t(S_{t+n}, a)$$

- **Peng**: No distinction between exploratory and greedy actions (for each n).
- **Naïve**: Similar to Watkins's method, except that the traces are not set to zero on exploratory actions.

Watkins's $Q(\lambda)$

Watkins's $Q(\lambda)$

Watkins: Zero out eligibility trace after a non-greedy action.
Do max when backing up at first non-greedy choice.



$$e_t(s, a) = \begin{cases} 1 + \gamma \lambda e_{t-1}(s, a) & \text{if } s = s_t, a = a_t, Q_{t-1}(s_t, a_t) = \max_a Q_{t-1}(s_t, a) \\ 0 & \text{if } Q_{t-1}(s_t, a_t) \neq \max_a Q_{t-1}(s_t, a) \\ \gamma \lambda e_{t-1}(s, a) & \text{otherwise} \end{cases}$$

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \delta_t e_t(s, a)$$

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$$

- Disadvantage to Watkins's method:
 - Early in learning, the eligibility trace will be "cut" frequently resulting in short traces

Watkins's $Q(\lambda)$

Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$, for all s, a

Repeat (for each episode):

 Initialize s, a

 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)

$a^* \leftarrow \arg \max_b Q(s', b)$ (if a' ties for the max, then $a^* \leftarrow a'$)

$\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$

$e(s, a) \leftarrow e(s, a) + \delta$

 For all s, a :

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$

 If $a' = a^*$, then $e(s, a) \leftarrow \gamma \lambda e(s, a)$

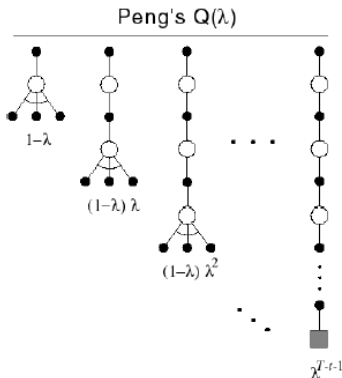
 else $e(s, a) \leftarrow 0$

$s \leftarrow s'; a \leftarrow a'$

 until s is terminal

Peng's $Q(\lambda)$

- No distinction between exploratory and greedy actions
- Backup max action except at end
- Never cut traces
- The earlier transitions of each are on-policy, whereas the last (fictitious) transition uses the greedy policy



- Disadvantage:
 - Complicated to implement
 - Theoretically, no guarantee to converge to the optimal value

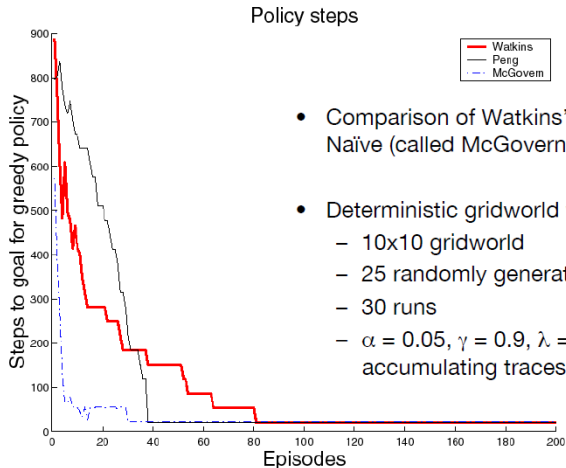
Naïve $Q(\lambda)$

- ▶ Idea: is it really a problem to backup exploratory actions?
 - ▶ Never zero traces
 - ▶ Always backup max at current action (unlike Peng or Watkins's)
 - ▶ Naïve $Q(\lambda)$ 等价与将 Watkins's $Q(\lambda)$ 算法中 traces 公式替换为 Eligibility traces:

$$e_t(s, a) = \begin{cases} \gamma \lambda e_{t-1}(s, a) + 1, & \text{if } s = s_t, a = a_t \\ \gamma \lambda e_{t-1}(s, a), & \text{otherwise} \end{cases}$$

- ▶ Works well in preliminary empirical studies

三种 $Q(\lambda)$ 比较



- Comparison of Watkins's, Peng's, and Naïve (called McGovern's here) $Q(\lambda)$
- Deterministic gridworld with obstacles
 - 10x10 gridworld
 - 25 randomly generated obstacles
 - 30 runs
 - $\alpha = 0.05$, $\gamma = 0.9$, $\lambda = 0.9$, $\epsilon = 0.05$, accumulating traces

From McGovern and Sutton (1997). Towards a better $Q(\lambda)$

小结

- ▶ TD(0) 的更新 $R_t + \gamma V(S_{t+1})$ 是对 MC 和 DP 的一种折衷
- ▶ MC 和 TD 对比
 - ▶ TD: on-line update; MC: off-line update
 - ▶ TD: 有偏、低方差、差收敛; MC: 无偏、高方差、收敛好
 - ▶ 一般而言, TD 比 MC 学的更快
 - ▶ TD 依赖马尔科夫性质; MC 不依赖马尔科夫性质
- ▶ TD(0) 是 1-step bootstrapping; MC 是 ∞ -step bootstrapping
- ▶ MC 与 TD(1) 在 off-line 情况下等价
- ▶ 前向 TD(λ) 和后向 TD(λ) 在 off-line 情况下等价
- ▶ TD 控制算法的两种实现: SARSA 和 Q-learning
 - ▶ Sarsa: on-policy; Q-learning: off-policy
 - ▶ Q-learning 比 Sarsa 学习的更快, 但更危险, 同时还有过估计问题