强化学习—免模型预测

> 作者：YJLAugus 博客： https://www.cnblogs.com/yjlaugus 项目地址： https://github.com/YJLAugus/Reinforcement-Learning-Notes，如果感觉对您有所帮助，烦请点个☆Star。

前言

在第二章**强化学习-动态规划-DP**中，我们讨论了用动态规划来求解强化学习预测问题和控制问题的方法。但是由于动态规划法需要在每一次回溯更新某一个状态的价值时，回溯到该状态的所有可能的后续状态。导致对于复杂问题计算量很大。同时很多时候，我们连环境的**动态特性** P 都无法知道，这时动态规划法根本没法使用。这时候我们如何求解强化学习问题呢？

从本章开始将花连续两讲的时间讨论解决一个可以被认为是MDP、但却不掌握MDP具体细节的问题，也就是讲述**如何直接从Agent与环境的交互来得得到一个估计的最优价值函数和最优策略**。这部分内容同样分为两部分，第一部分也就是本章的内容，聚焦于策略评估，也就是预测，直白的说就是在给定的策略同时不清楚MDP细节的情况下，估计Agent会得到怎样的最终奖励。下一讲将利用本章的主要观念来进行控制进而找出最优策略，最大化Agent的奖励。

本章内容分为三个小部分，分别是**蒙特卡洛强化学习**、**时序差分强化学**习和介于两者之间的**λ时序差分强化学习**。相信读者在阅读本讲内容后会对这三类学习算法有一定的理解。

其中在第三章**蒙特卡洛（Markov Chain & Monte Carlo, MCMC）方法**对蒙特卡洛方法进行了一个简单的介绍，这样对于这一章节问题的解决会有很多帮助。

免模型的强化学习问题定义

在动态规划法中，强化学习的两个问题是这样定义的：

- 预测问题，即给定强化学习的6个要素：状态集 S, 动作集 A, 模型状态转化概率矩阵P, 即时奖励 R，衰减因子γ, 给定策略 π，求解该策略的状态价值函数 v_π。
- 控制问题，也就是求解最优的价值函数和策略。给定强化学习的5个要素：状态集 S, 动作集 A, 模型状态转化概率矩阵P, 即时奖励 R，衰减因子γ, 给定策略 π， 求解最优的状态价值函数v_*和最优策略 π_*

可见，**模型状态转化概率矩阵P始终是已知的，即MDP已知**，对于这样的强化学习问题，我们一般称为**基于模型的强化学习问题**。

不过有很多强化学习问题，我们没有办法事先得到模型状态转化概率矩阵P，这时如果仍然需要我们求解强化学习问题，那么这就是不基于模型的强化学习问题了——**免模型的强化学习**。它的两个问题一般的定义是：

- 预测问题，即给定强化学习的5个要素：状态集S, 动作集A 即时奖励R，衰减因子γ，给定策略π，求解该策略的状态价值函数v_π
- 控制问题，也就是求解最优价值函数和策略。给定强化学习的5个要素：状态集S, 动作集A, 即时奖励R，衰减因子γ, **探索率** ϵ , 求解最优的动作价值函数q_*和最优策略π_*

本章节要讨论的蒙特卡洛方法就是上述免模型的强化学习问题。

蒙特卡洛强化学习 (Monte-Carlo Reinforcement Learning)

蒙特卡洛强化学习概念

蒙特卡洛强化学习：是在不清楚MDP状态转移及即时奖励的情况下，直接从经历完整的`Episode`来学习状态价值，通常情况下某状态的价值等于在多个Episode中以该状态算得到的所有收获的平均。

> Episode： agent根据某个策略执行一系列action到结束就是一个episode。

注：**收获**不是针对Episode的，它存在于Episode内，针对于Episode中某一个状态。从这个状态开始经历完Episode时得到的有衰减的即时奖励的总和。从一个Episode中，我们可以得到该Episode内所有状态的收获。当一个状态在Episode内出现多次，该状态的收获有不同的计算方法，下文会讲到。

完整的Episode 指必须从某一个状态开始，Agent与Environment交互直到**终止状态**，环境给出终止状态的即时收获为止。

蒙特卡洛强化学习特点

蒙特卡洛强化学习有如下特点：不基于模型本身，直接从经历过的Episode中学习，必须是**完整的Episode**，使用的思想就是用平均收获值代替价值。理论上Episode越多，结果越准确。

蒙特卡罗法通过采样若干经历完整的`状态序列(episode)`来估计状态的真实价值。所谓的经历完整，就是这个序列必须是达到终点的。比如下棋问题分出输赢，驾车问题成功到达终点或者失败。有了很多组这样经历完整的状态序列，我们就可以来近似的估计状态价值，进而求解预测和控制问题了。

蒙特卡洛策略评估 (Monte-Carlo Policy Evaluation)

> **目标：** 在给定策略下，从一系列的完整Episode经历中学习,最后求得到该策略下的状态价值函数。

在解决问题过程中主要使用的信息是一系列完整Episode。其包含的信息有：状态的转移、使用的行为序列、中间状态获得的即时奖励以及到达终止状态时获得的即时奖励。其特点是使用有限的、完整Episode产生的这些经验性信息经验性地推导出每个状态的平均收获，以此来替代收获的期望，而后者就是状态价值。通常需要掌握完整的MDP信息才能准确计算得到。

数学描述如下：

基于特定策略 π 的一个Episode信息可以表示为如下的一个序列：

$$S_1, A_1, R_2, \ldots, S_k \backsim \pi$$

t 时刻，S_t 的收获：

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

其中，T 为终止时刻。

该策略下某一状态 s 的价值：

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

> 注： R_{t+1} 表示的是 t时刻agent在状态 S_t 获得的即时奖励，下文都使用这种下标来表示即时奖励。更准确的表述为：个体在状态 S_t 执行一个行为a 后离开该状态获得的即时奖励。

很多时候，即时奖励只出现在Episode结束状态时，但不能否认在中间状态也可能有即时奖励。公式里的 R_t 指的是**任何状态**得到的即时奖励，这一点尤其要注意。

在状态转移过程中，可能发生一个状态经过一定的转移后又一次或多次返回该状态，此时在一个Episode里如何计算这个状态发生的次数和计算该Episode的收获呢？可以有如下两种方法：

首次访问蒙特卡洛策略评估

在给定一个策略，使用一系列完整Episode评估某一个状态s时，对于每一个Episode，仅当该状态**第一次**出现在一个 episode中时：

- 状态出现的次数加1： $N(s) \leftarrow N(s)+1$
- 总的收获更新： $S(s) \leftarrow S(s)+G_t$
- 状态 s 的价值： $V(s)=S(s)/N(s)$

当 $N(s) \rightarrow \infty$ 时， $V(s) \rightarrow v_\pi(s)$

每次访问蒙特卡洛策略评估

在给定一个策略，使用一系列完整Episode评估某一个状态s时，对于每一个Episode，状态 s **每次**出现在一个epospde中时：

- 状态出现的次数加1： $N(s) \leftarrow N(s)+1$
- 总的收获更新： $S(s) \leftarrow S(s)+G_t$
- 状态 s 的价值： $V(s)=S(s)/N(s)$

当 $N(s) \rightarrow \infty$ 时， $V(s) \rightarrow v_\pi(s)$。计算的公式与 **首次访问蒙特卡洛策略评估** 的公式相同，但是具体的意义却不同，下一以一个简单的例子进行说明。

二十一点 二十一点又名黑杰克（Blackjack），是一种流行于赌场的游戏，其目标是使得你的扑克牌点数之和不超过21的情况下越大越好。K、Q、J和10牌都算作10点（一般记作T，即ten之意）；A 牌（ace）既可算作1点也可算作11点，由玩家自己决定（当玩家停牌时，点数一律视为最大而尽量不爆，如A+9为20，A+4+8为13，A+3+A视为15）。游戏开始时，会给玩家和庄家各发两张牌。庄家的牌一张正面朝上，一张背面朝上，玩家两张都是明牌（都正面朝上）如果玩家的两张牌分别是一张A，一张10点（可能是10，J， Q，K），这种情况称为**天和**，玩家直接获胜。除非庄家也是天和，那就是平局。如果玩家不是天和，那么他可以一张一张地继续要牌，直到他主动停止（停牌）或者牌的点数和超过21点（爆牌）。如果玩家选择停牌，就轮到庄家行动。庄家根据一个固定的策略进行游戏：他一直要牌，直到点数等于或超过17时停牌。如果庄家爆牌，那么玩家获胜，否则根据谁的点数更靠近21决定胜负或者平局。

根据以上游戏规则，我们得到如下信息：

状态空间： （多达200种，根据对状态的定义可以有不同的状态空间，这里采用的定义是牌的分数，不包括牌型）

- 当前牌的分数（12 - 21），低于12时，你可以安全的再叫牌，所以没意义。
- 庄家出示的牌（A - 10），庄家会显示一张牌面给玩家，另一张为暗牌。
- 我有“useable” ace吗？（是或否）A既可以当1点也可以当11点。

行为空间：

- 停止要牌 stick
- 继续要牌 twist

奖励（停止要牌）：

- +1：如果你的牌分数大于庄家分数
- 0：如果两者分数相同
- -1：如果你的牌分数小于庄家分数

奖励（继续要牌）：

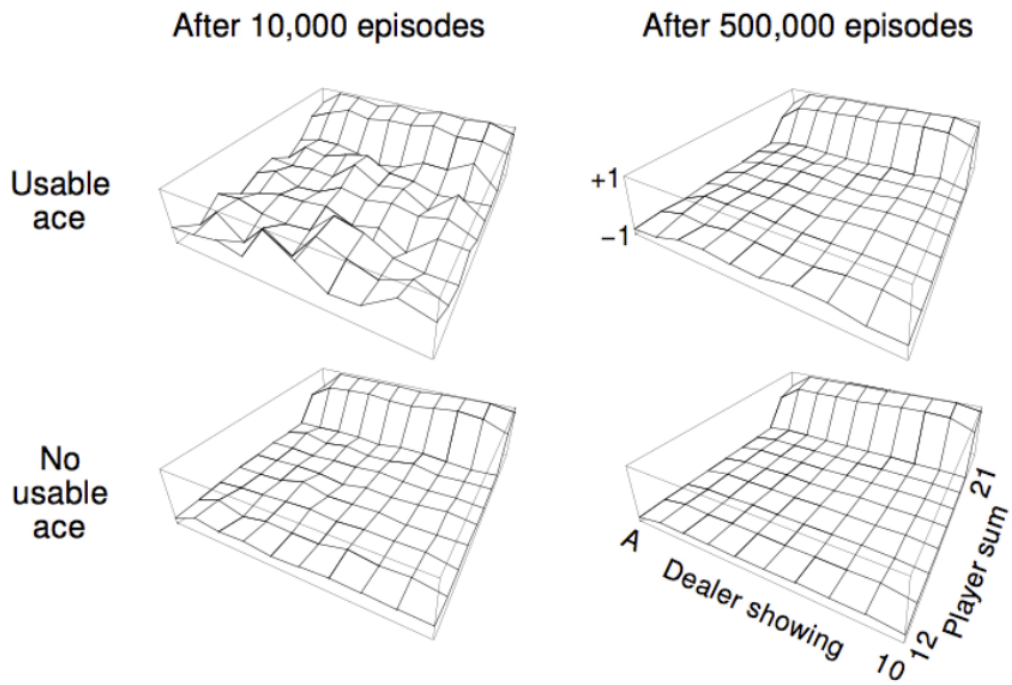
- -1：如果牌的分数>21，并且进入终止状态
- 0：其它情况

状态转换（Transitions）：如果牌分小于12时，自动要牌

当前策略：牌分只要小于20就继续要牌。

求解问题：评估该策略的好坏。

求解过程：使用庄家显示的牌面值、玩家当前牌面总分值来确定一个二维状态空间，区分手中有无A分别处理。统计每一牌局下决定状态的庄家和玩家牌面的状态数据，同时计算其最终收获。通过模拟多次牌局，计算每一个状态下的平均值，得到如下图示。



Policy: stick if sum of cards $\geq$ 20, otherwise twist

最终结果：无论玩家手中是否有A牌，该策略在绝大多数情况下各状态价值都较低，只有在玩家拿到21分时状态价值有一个明显的提升。

这个例子只是使读者对蒙特卡洛策略评估方法有一个直观的认识。

为了尽可能使读者对MC方法有一个直接的认识，我们尝试模拟多个二十一点游戏牌局信息，假设我们仅研究初始状态下庄家一张明牌为4，玩家手中前两张牌和为15的情形，不考虑A牌。在给定策略下，玩家势必继续要牌，根据蒙特卡洛策略评估则可能会出现如下多种情形：

庄家总序列	玩家最终序列	玩家获得奖励	当前估计的状态价值
4，10，3（17点）	Q，5，5（20点）	+1	+1
4，J，7（21点）	9，6，8（23点，爆牌）	-1	0
4，10，2，8（24点，爆牌）	7，8，6（21点）	+1	0.333
4，4，2，7（17点）	J，5，Q（15点）	-1	0
4，9（13点）	5，K，4，8（27点，爆牌）	-1	-0.2
4，3，K（17点）	8，7，5（20点）	+1	0
…	…	…	…

以上状态价值由 以下公式得出：

- 状态出现的次数加1：$N(s) \leftarrow N(s)+1$
- 总的收获更新：$S(s) \leftarrow S(s)+G_t$
- 状态 s 的价值：$V(s)=S(s)/N(s)$

以前三个序列为例

1. 序列一： N = 1， S = 1，V = 1 / 1 = 1;
2. 序列二： N = 2， S = 1 - 1 = 0，V = 0 / 2 = 0;
3. 序列三： N = 3， S = 0 + 1 = 1，V = 1 / 3 = 0.333

可以看到，使用只有当牌不小于20的时候才停止叫牌这个策略（注：庄家不需要遵从这个策略），**前6次平均价值为0**，如果玩的牌局足够多，按照这样的方法可以针对每一个状态（庄家第一张明牌，玩家手中前两张牌分值合计）都可以制作这样一张表，进而计算玩家奖励的平均值。通过结果，可以发现这个策略并不能带来很高的玩家奖励。

这里给出表中第一个对局对应的信息序列（Episode）：

$$S_0<4,15>, A_0<\text{要牌}>, R_1<0>, S_1<4,20>, A_1<\text{停止要牌}>, \ R_2<+1>$$

可以看出，这个完整的Episode中包含两个状态，其中第一个状态的即时奖励为0，后一个状态是终止状态，根据规则，玩家赢得对局，获得终止状态的即时奖励+1。读者可以加深对即时奖励、完整Episode的理解。

通过上面的例子，我们使用蒙特卡洛方法求解的是**平均收获**，根据上面的例子，可以很清楚的知道这点。通常计算平均值就需要预先存储所有的数据，最后加和取平均，这样真的在计算机中计算时，会浪费很多空间。下面介绍一种更简单的方法——**累进更新平均值（Incremental Mean）**

累进更新平均值（Incremental Mean）

这里提到了在实际操作时常用的一个实时更新均值的办法，使得在计算平均收获时不需要存储所有既往收获，而是每得到一次收获，就计算其平均收获。

理论公式如下：

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

这个公式比较简单。把这个方法应用于蒙特卡洛策略评估，就得到下面的蒙特卡洛累进更新。

蒙特卡洛累进更新

对于一些列 Episodes中的每一个： $S_1, A_1, R_2, \ldots, S_k \backsim \pi$，对于Episode里的每一个状态$S_t$ ，有一个收获G_t，每碰到一次S_t ，使用下面的公式计算状态的平均价值$V(S_t)$：

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

其中：

$$N(S_t) = N(S_t) + 1$$

这样我们无论数据量是多还是少，算法需要的内存基本是固定的 。

有时候，尤其是海量数据做分布式迭代的时候，我们可能无法准确计算当前的次数$N(S_t)$，这时我们可以用一个系数α来代替，即：

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

以上就是蒙特卡罗方法求解预测问题的整个过程。由于蒙特卡洛学习方法有许多缺点（后文会细说），因此实际应用并不多。接下来介绍实际常用的时序差分学习方法。

时序差分学习 Temporal-Difference Learning

时序差分学习特点

时序差分学习简称TD学习，它的特点如下：和蒙特卡洛学习一样，它也从Episode学习，不需要了解模型本身；但是它可以学习**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。

时序差分策略评估

我们已经学过，在Monte-Carlo学习中，使用实际的收获（return）G_t来更新价值（Value）：

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t)) \tag{1}$$

由第一章的收获公式(注：收获/回报都是G_t，以后两者不会特别区分)可得：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots + \gamma^{T-1} R_T = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \qquad \gamma \in [0,1], \quad (T \to \infty)$$

> 注：T 用来指示 R 的下标，表明是第几个 R

如果用G_t 来更新价值的话，就可以写成如下：

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

在TD学习中，算法在估计某一个状态的价值时，用的是离开该状态的即刻奖励 R_{t+1} 与下一个状态 S_{t+1} 的预估价值乘以衰减系数γ 组成，这符合Bellman方程的描述，故把（2）式带入（1）式得：

$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{3}$$

其中（3）式中：

- $R_{t+1} + \gamma V(S_{t+1})$ 称为**TD的目标值**
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ 称为**TD误差**

BootStrapping 指的就是TD目标值 $R_{t+1} + \gamma V(S_{t+1})$ 代替收获G_t的过程，暂时把它翻译成“**引导**”。

蒙特卡洛和TD策略评估差别

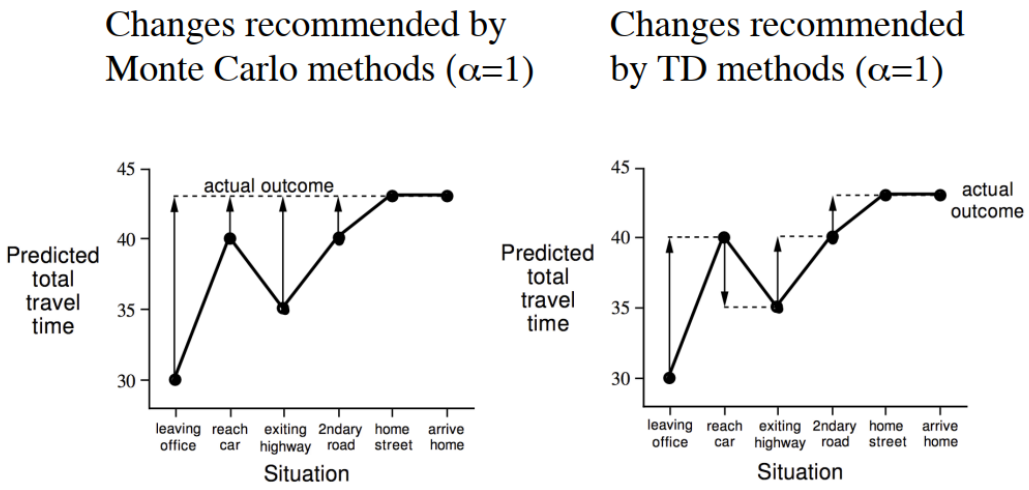
例子——驾车返家：想象一下你下班后开车回家，需要预估整个行程花费的时间。假如一个人在驾车回家的路上突然碰到险情：对面迎来一辆车感觉要和你相撞，严重的话他可能面临死亡威胁，但是最后双方都采取了措施没有实际发生碰撞。如果使用蒙特卡洛学习，路上发生的这一险情可能引发的负向奖励不会被考虑进去，不会影响总的预测耗时；但是在TD学习时，碰到这样的险情，这个人会立即更新这个状态的价值，随后会发现这比之前的状态要糟糕，会立即考虑决策降低速度赢得时间，也就是说你不必像蒙特卡洛学习那样直到他死亡后才更新状态价值，那种情况下也无法更新状态价值。

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。

状态	已消耗时间（分钟）	预计仍需耗时（分钟）	预计总耗时（分钟）
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

基于上表所示的数据，下图展示了蒙特卡洛学习和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）。这里使用的是**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。对于蒙特卡洛学习过程，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，在下一次返家的时候用新估计的时间来帮助决策；而对于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。

Driving Home Example: MC vs. TD



MC vs. TD (—)

通过上面的例子我们可以分析得出MC和TD的不同如下

- TD 在知道结果之前可以学习，MC必须等到最后结果才能学习
- TD 可以在没有结果时学习，可以在持续进行的环境里学习, MC必须要等到完整的Episdo结束后才能学习

Advantages and Disadvantages of MC vs. TD

- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

MC vs. TD (二)

G_t : 实际收获，是基于某一策略状态价值的**无偏**估计：

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

`TD target` ： TD目标值，是基于下一状态**预估价值**计算当前预估收获，是当前状态实际价值的**有偏**估计：

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

`True TD target` ： 真实TD目标值，是基于下一状态的**实际价值**对当前状态实际价值的**无偏**估计：

$$R_{t+1} + \gamma v_\pi(S_{t+1})$$

Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$ is *unbiased* estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is *unbiased* estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is *biased* estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return:
  - Return depends on *many* random actions, transitions, rewards
  - TD target depends on *one* random action, transition, reward

- MC 没有偏倚（bias），但有着较高的变异性（Variance），且对初始值不敏感
- TD 低变异性（Variance）,但有一定程度的偏倚（bias），对初始值较敏感，通常比 MC 更高效

这里的**偏倚**指的是距离期望的距离，预估的平均值与实际平均值的偏离程度；**变异性**指的是方差，评估单次采样结果相对于与平均值变动的范围大小。基本就是统计学上均值与方差的概念。

Advantages and Disadvantages of MC vs. TD (2)

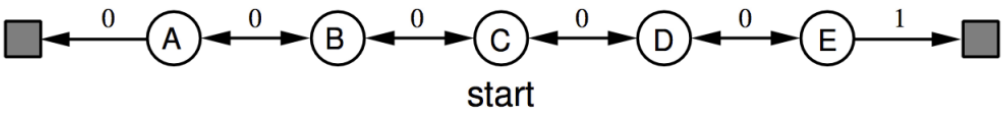
- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

对于MC和TD的区别，还可以用下面的例子来加深理解：

例子——随机行走：

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

Random Walk Example



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

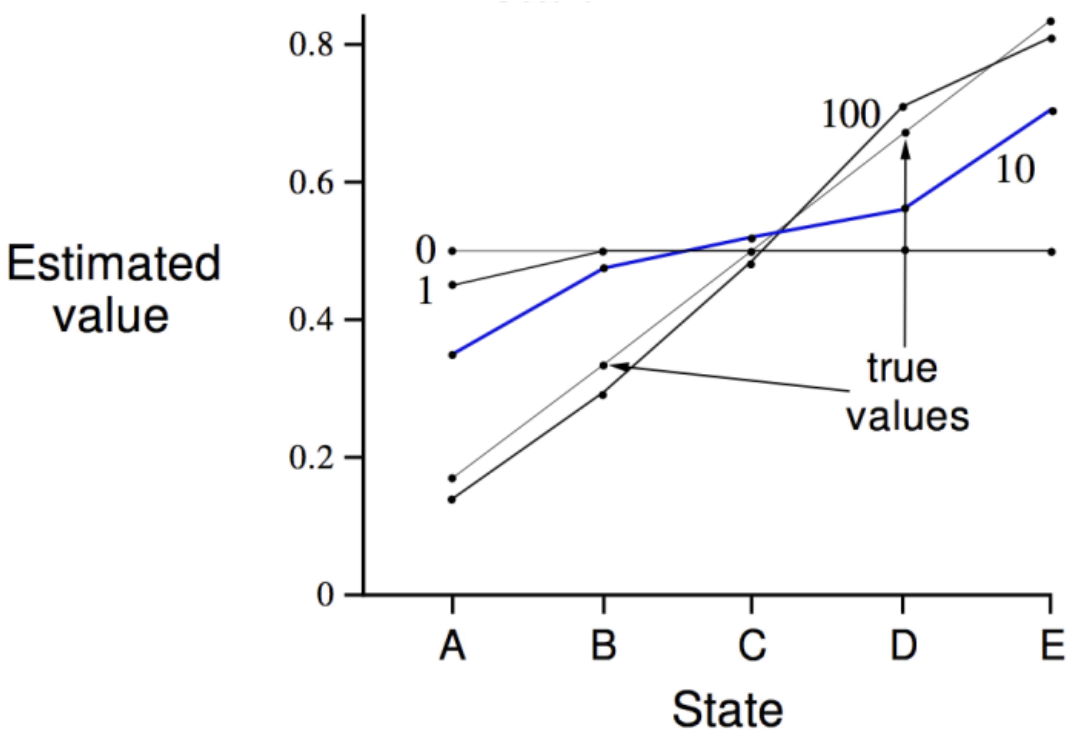
状态转移：100%按行为进行状态转移，进入终止状态即终止

衰减系数：1

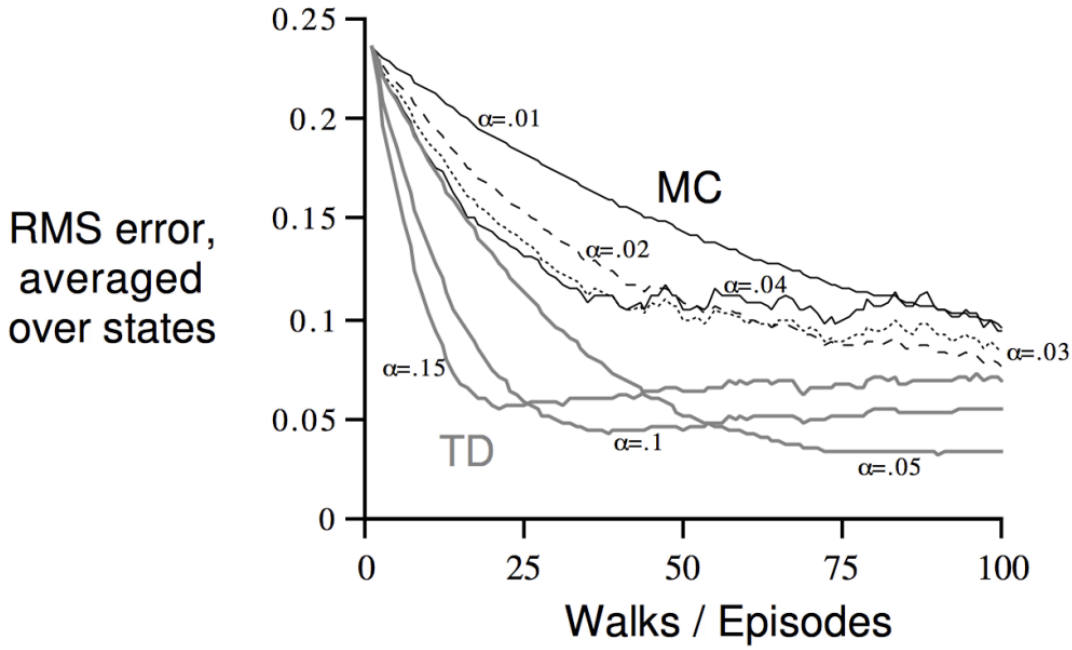
给定的策略：随机选择向左、向右两个行为

问题：对这个MDP问题进行预测，也就是评估随机行走这个策略的价值，也就是计算该策略下每个状态的价值，也就是确定该MDP问题的状态价值函数

求解：下图是使用TD算法得到的结果。横坐标显示的是状态，纵坐标是各状态的价值估计，一共5条折线，数字表明的是实际经历的Episode数量，true value所指的那根折线反映的是各状态的实际价值。第0次时，各状态的价值被初始化为0.5，经过1次、10次、100次后得到的价值函数越来越接近实际状态价值函数。



下图比较了MC和TD算法的**效率**。横坐标是经历的Episode数量，纵坐标是计算得到的状态函数和实际状态函数下各状态价值的均方差。黑色是MC算法在不同step-size下的学习曲线，灰色的曲线使用TD算法。可以看出TD较MC更高效。此图还可以看出当step-size不是非常小的情况下，TD有可能得不到最终的实际价值，将会在某一区间震荡。



例子——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：1

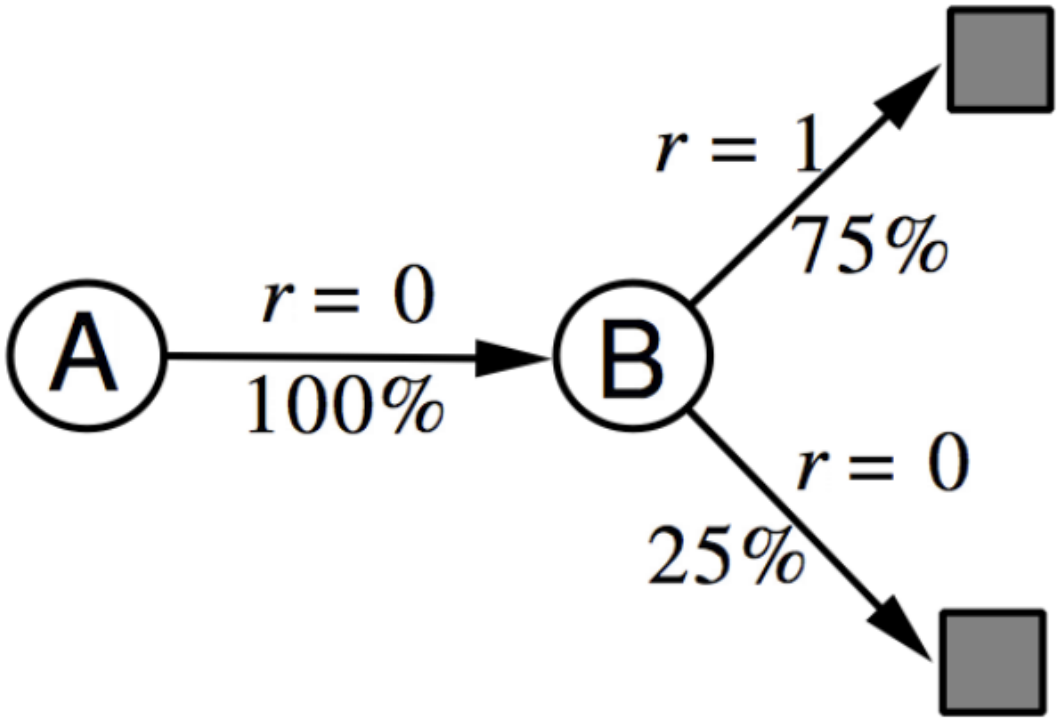
问题：依据仅有的Episode，衰减系数为1（即无衰减），计算状态A，B的价值分别是多少，即V(A)=？，V(B)=？

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。

解释：

应用MC算法，由于需要完整的Episode，因此仅Episode1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。



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：

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

上式中，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1，2，3，4...个状态等），T_k 表示的是第k 个Episode中总的状态数，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。

在例子**AB**中，利用MC算法，V(A)=0。

TD算法则收敛至一个根据已有经验构建的最大可能的马儿可夫模型（MDP<S, A, P, R, γ>）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**：

$$P_{ss'}^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}1(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

同时估计某一状态的**即时奖励**：

$$P_s^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}1(s_t^k, a_t^k = s, a)r_t^k$$

最后计算该MDP的状态函数。

在例子**AB**中，利用TD算法，V(A)=6/8。

Certainty Equivalence

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

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

  - In the AB example, $V(A) = 0$
- TD(0) converges to solution of max likelihood Markov model
  - Solution to the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data

$$\hat{\mathcal{P}}_{s,s'}^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

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

  - In the AB example, $V(A) = 0.75$

> 注：公式符号有所不同，但是表达意义一致。

MC vs. TD (三)

通过比较可以看出，TD算法使用了MDP问题的马儿可夫性，在Markov 环境下更有效；但是MC算法并不利用马儿可夫性，通常在非Markov环境下更有效。

DP，MC，TD 总结

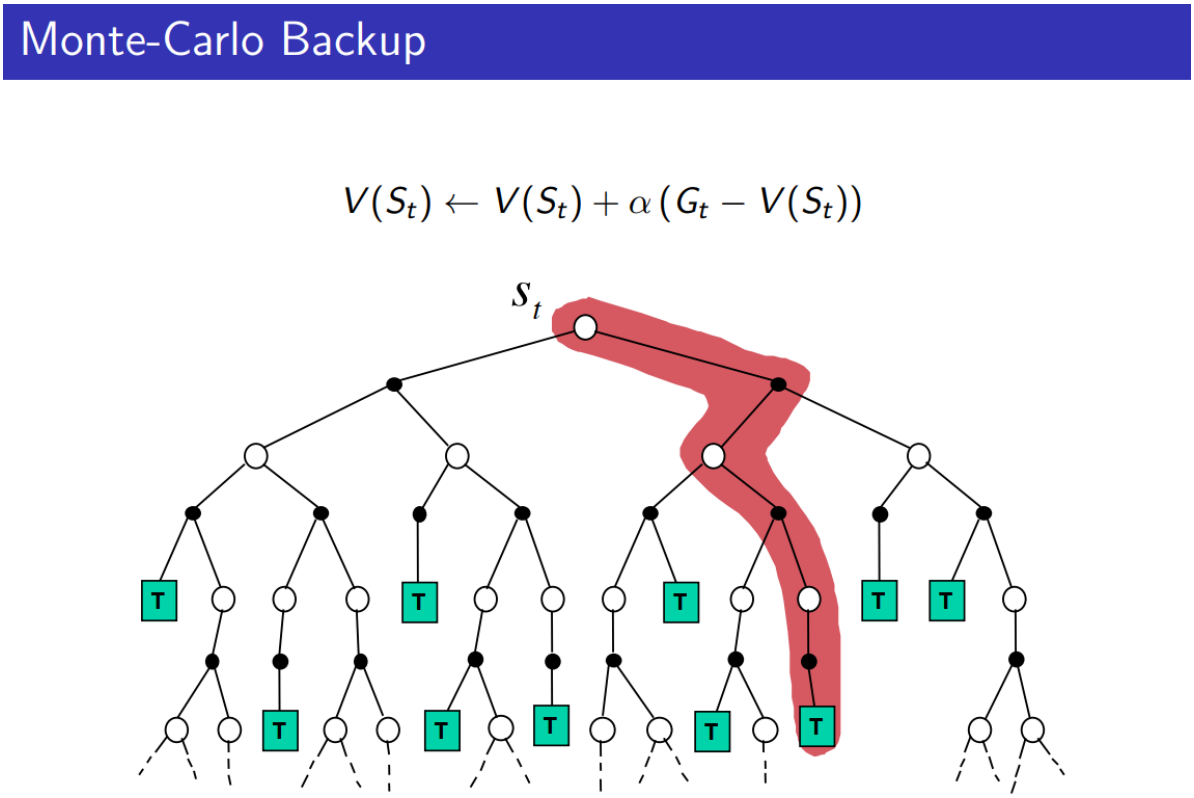
Monte-Carlo，Temporal-Difference 和 Dynamic Programming 都是**计算状态价值**的一种方法，区别在于，前两种是在不知道Model的情况下的常用方法，这其中又以MC方法需要一个**完整**的Episode来更新状态价值，TD则不需要完整的Episode；DP方法则是基于Model（知道模型的运作方式）的计算状态价值的方法，它通过计算一个状态S所有可能的转移状态 S' 及其转移概率以及对应的即时奖励来计算这个状态S的价值。

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。

下面的几张图直观地体现了三种算法的区别：

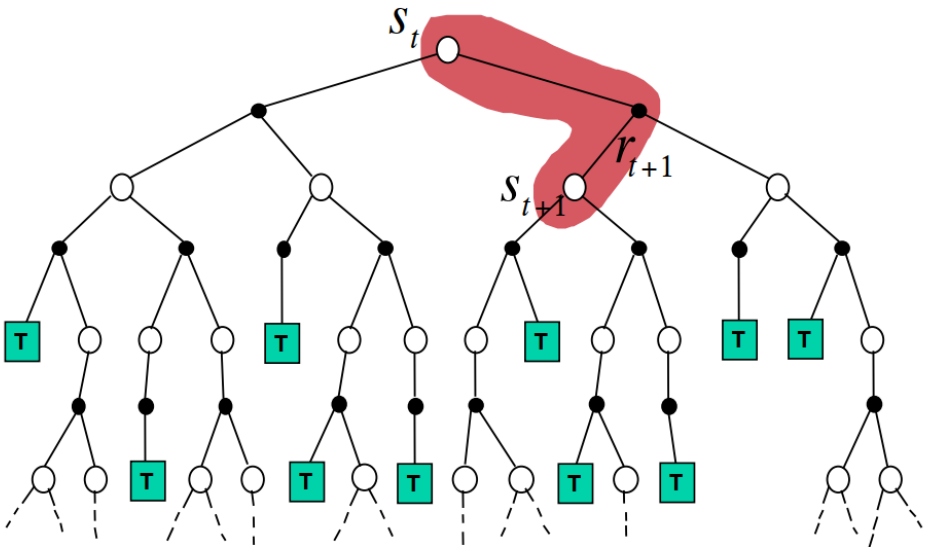
1. MC：采样，一次完整经历，用实际收获更新状态预估价值。



2. TD：采样，经历可不完整，用喜爱状态（已有的状态）的预估状态价值预估收获，再更新预估价值。

Temporal-Difference Backup

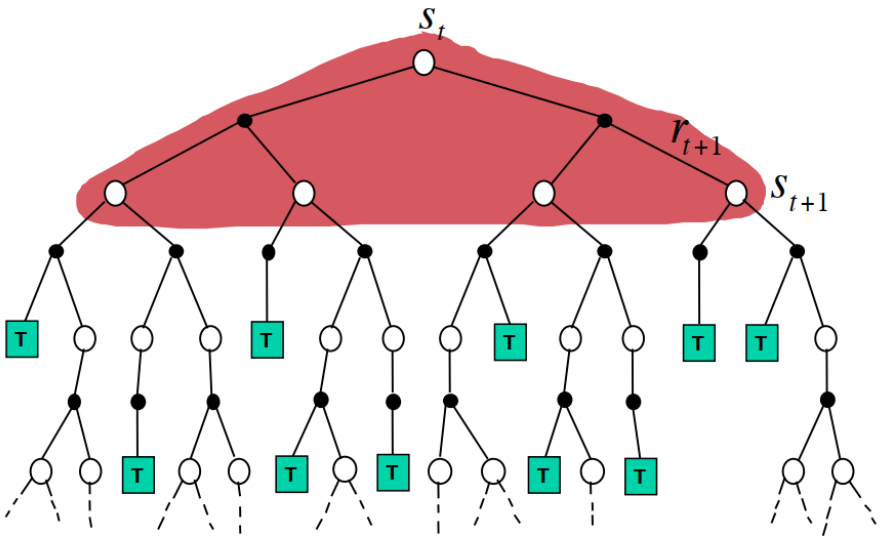
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



3. DP：没有采样，根据完整模型，依靠预估数据更新状态价值。

Dynamic Programming Backup

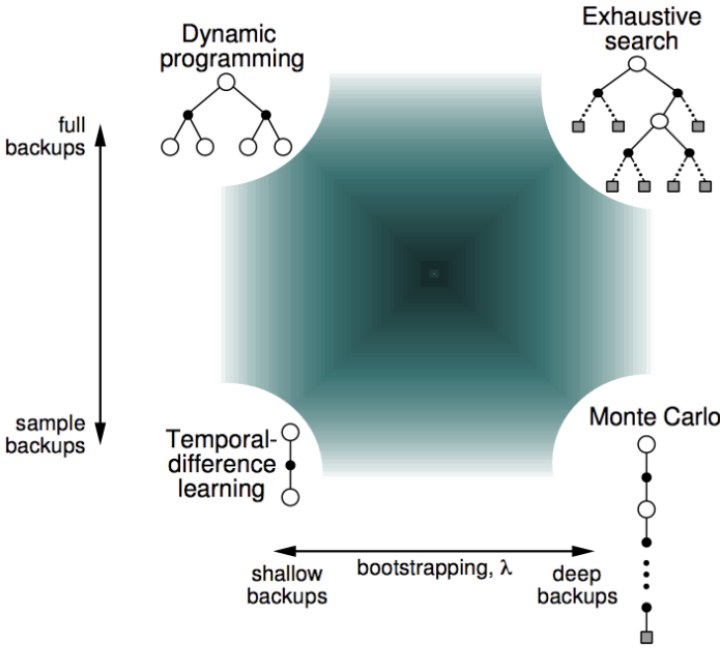
$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



下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，同时不走完整个Episode就是TD；当使用单个采样但走完整个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。

需要提及的是：DP利用的是整个MDP问题的模型，也就是状态转移概率，虽然它并不实际利用样本，但是它利用了整个模型的规律，因此认为是Full Width的。

Unified View of Reinforcement Learning



λ时序差分强化学习TD(λ)

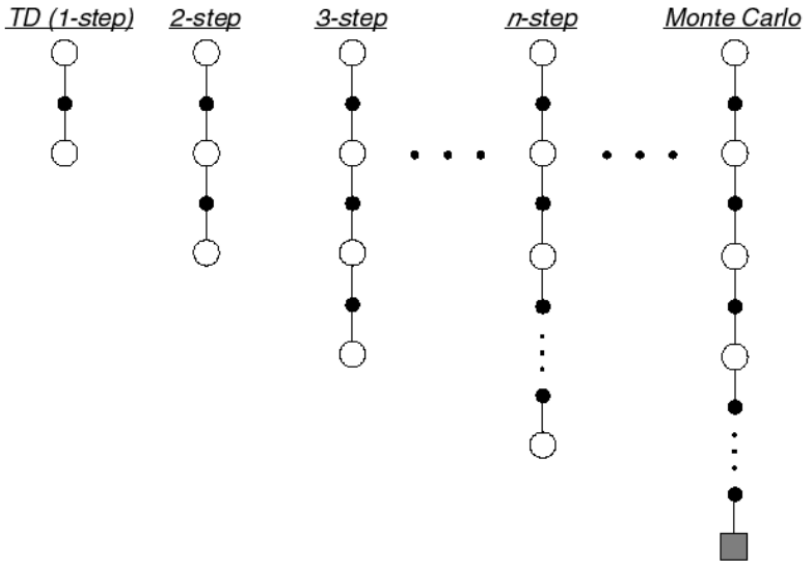
先前所介绍的TD算法实际上都是TD(0)算法，括号内的数字0表示的是在当前状态下往前多看1步，要是往前多看2步更新状态价值会怎样？这就引入了 `n-step` 的概念。

n-步预测 (n-Step Prediction)

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。

n-Step Prediction

- Let TD target look *n* steps into the future



> 注：图中空心大圆圈表示状态，实心小圆圈表示行为

n-步收获(n-Step Return)

TD或TD(0)是基于 1-步 预测的，MC则是基于 ∞-步 预测的。

n=1	TD 或TD(0)	$G_t^{(1)} = R_{t+1} + \gamma(S_{t+1})$
n=2		$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$
…		
n= ∞	MC	$G_t^{\infty} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots + \gamma^{T-1} R_T$

> 注： n = 2时不写成TD(2)

n-步收获：

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

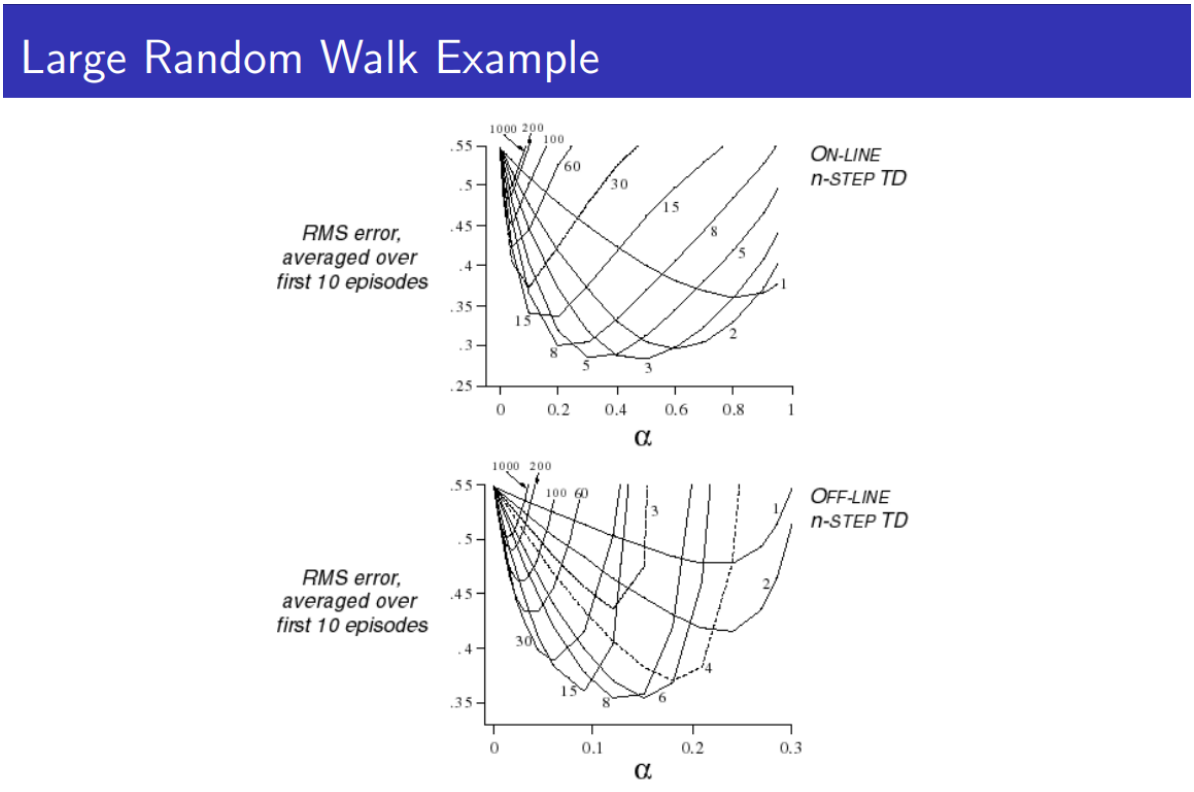
那么，n步TD学习状态**价值函数**的更新公式为：

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

既然存在n-步预测，那么n=？时效果最好呢，下面的例子试图回答这个问题：

例子：大规模随机行走

这个示例研究了使用多个不同步数预测联合不同步长(step-size，公式里的系数α）时，分别在在线和离线状态时状态函数均方差的差别。所有研究使用了10个Episode。离线与在线的区别在于，离线是在经历所有10个Episode后进行状态价值更新；而在线则至多经历一个Episode就更新依次状态价值。



结果如图表明，离线和在线之间曲线的形态差别不明显；从步数上来看，步数越长，越接近MC算法，均方差越大。对于这个大规模随机行走示例，在线计算比较好的步数是3-5步，离线计算比较好的是6-8步。但是不同的问题其对应的比较高效的步数不是一成不变的。因此选择多少步数作为一个较优的计算参数也是一个问题。

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。这就是**λ预测**和**λ收获。**

λ收获

λ-收获G_t^λ综合考虑了从1到∞的所有收获，它给其中的任意一个 `n-step` 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：

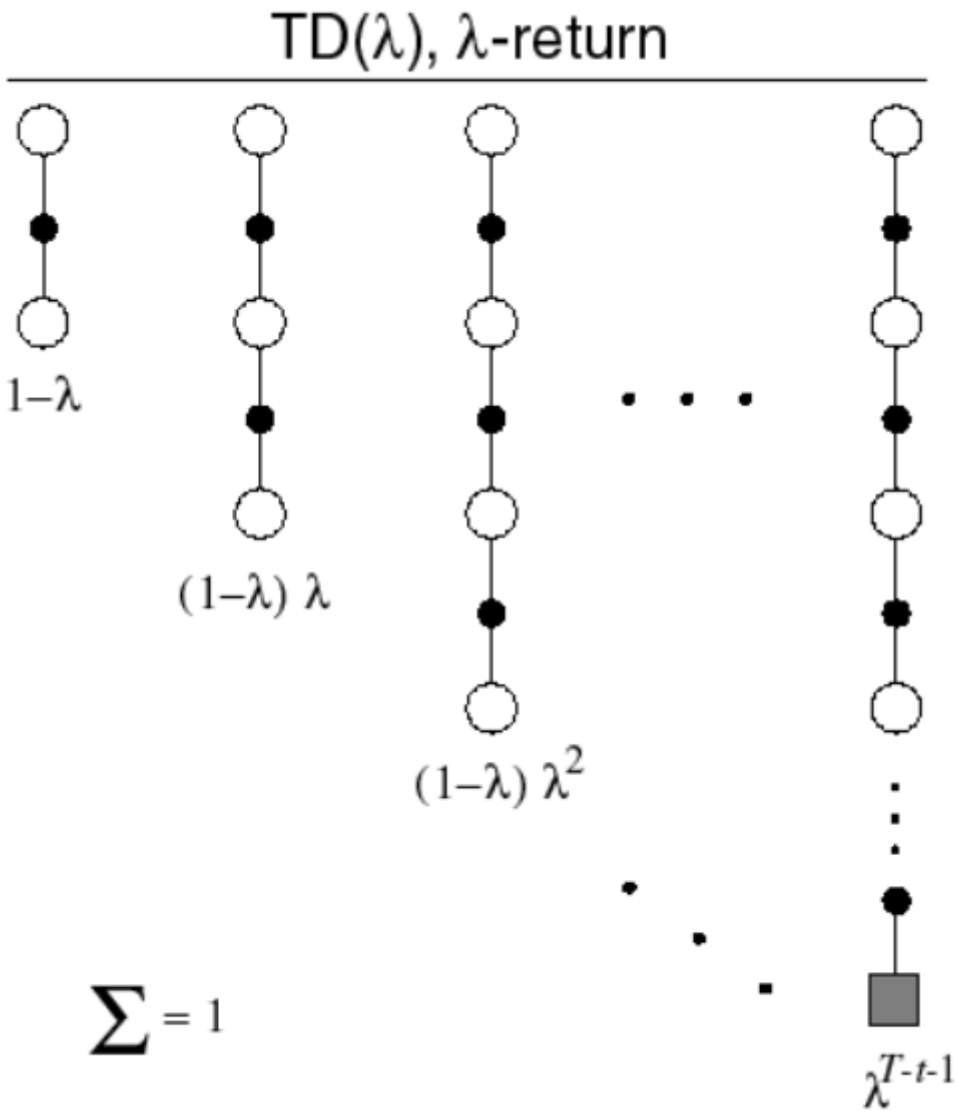
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

λ预测

对应的λ-预测写成TD(λ):

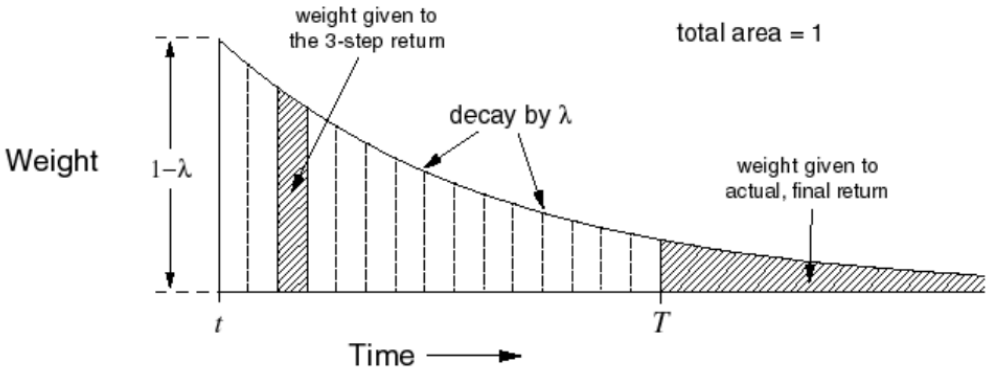
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^\lambda - V(S_t))$$

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。



TD(λ)对于权重分配的图解

TD(λ) Weighting Function



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

这张图还是比较好理解，例如对于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。

TD((λ)的设计使得Episode中，**后一个状态的状态价值与之前所有状态的状态价值有关，同时也可以说成是一个状态价值参与决定了后续所有状态的状态价值。但是每个状态的价值对于后续状态价值的影响权重是不同的。**我们可以从两个方向来理解TD(λ)：

前向认识TD(λ)

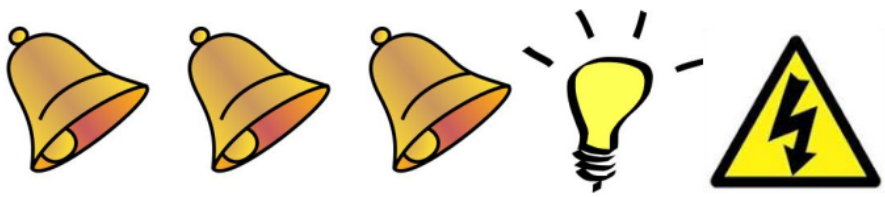
引入了λ之后，会发现要更新一个状态的状态价值，必须要走完整个Episode获得每一个状态的即时奖励以及最终状态获得的即时奖励。这和MC算法的要求一样，因此TD(λ)算法有着和MC方法一样的劣势。λ取值区间为[0,1]，**当λ=1时对应的就是MC算法。**这个实际计算带来了不便。

反向认识TD(λ)

TD(λ)从另一方面提供了一个单步更新的机制，通过下面的示例来说明。

示例——被电击的原因

这是之前见过的一个例子，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?



两个概念：

频率启发 (Frequency heuristic)：将原因归因于出现频率最高的状态

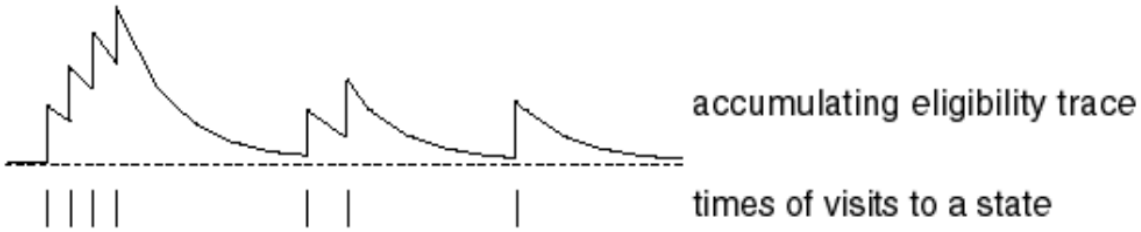
就近启发 (Recency heuristic)：将原因归因于较近的几次状态

给每一个状态引入一个数值：**效用追踪**（Eligibility Traces, ES，**也有翻译成“资质追踪”，这是同一个概念从两个不同的角度理解得到的不同翻译**），可以结合上述两个启发。定义：

$$E_0(s) = 0$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s)$$

其中 $1(S_t = s)$ 是一个条件判断表达式。

下图给出了$E_t(s)$ 关于t的一个可能的曲线图：



该图横坐标是时间t，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。

特别的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**，有一个瞬间最高上限：

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

把刚才的描述体现在公式里**更新状态价值**，得到如下式子：

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

> 注：每一个状态都有一个 E值， E值随时间而变化。

当λ=0时，只有当前状态得到更新，等同于TD(0)算法；

当λ=1时，TD(1)粗略看与每次访问的MC算法等同；

- 在线更新时，状态价值差每一步都会有积累；
- 离线更新时，TD(1)等同于MC算法。

> 注：ET是一个非常符合神经科学相关理论的、非常精巧的设计。把它看成是神经元的一个参数，它反映了神经元对某一刺激的敏感性和适应性。神经元在接受刺激时会有反馈，在持续刺激时反馈一般也比较强，当间歇一段时间不刺激时，神经元又逐渐趋于静息状态；同时不论如何增加刺激的频率，神经元有一个最大饱和反馈。

小结

下表给出了λ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)
	$\Vert$	$\Vert$	$\Vert$
Forward view	TD(0)	Forward TD(λ)	MC
Online updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\Vert$	$\nVdash$	$\nVdash$
Forward view	TD(0)	Forward TD(λ)	MC
	$\Vert$	$\Vert$	$\Vert$
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，在下一讲讲解控制相关的算法时会详细介绍TD算法的实现。

参考文献

https://zhuanlan.zhihu.com/p/28107168

https://www.cnblogs.com/pinard/p/9492980.html

http://www.incompleteideas.net/book/the-book.html

https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf

https://baike.baidu.com/item/21 点/5481683?fr=aladdin