目前，机器学习算法根据模型的学习过程大致可以分为四类：**有监督学习(Supervised learning)**，**无监督学习(Unsupervised learning)**，**半监督学习(Semi-supervised learning)**和**强化学习(Reinforcement learning)**。监督式学习是从**标记好的训练数据**中进行模型的训练，常用来做分类和回归；无监督式学习是根据数据的特征直接对数据的结构和数值进行归纳，其特点是**仅对网络提供输入范例，而它会自动从这些范例中找出其潜在类别规则**，当学习完毕并经测试后可以将之应用到新的案例上，常用来做聚类；半监督式学习是根据利用**少量标注样本和大量未标注样本**进行模型的学习，常用来做回归和分类；强化学习是通过不断的试错、反馈进行学习，常用来做序列决策或者控制问题。

强化学习

强化学习是让计算机实现从一开始什么都不懂, 通过不断地尝试, 从错误中学习, 最后找到规律, 学会并达到目的的方法. 实际中的强化学习例子有很多. 比如最有名的 Alpha go, 机器头一次在围棋场上战胜人类高手, 让计算机自己学着玩经典游戏 Atari, 这些都是让计算机在不断的尝试中更新自己的行为准则, 从而一步步学会如何下好围棋, 如何操控游戏得到高分.

在强化学习过程中，计算机需要一位虚拟的老师, 但这个老师不会告诉你如何移动, 如何做决定, 他只会给你的行为打分, 那我们应该以什么形式学习这些现有的资源, 或者说怎么样只从分数中学习到如何做决定呢? 很简单, 我只需要记住那些高分, 低分对应的行为, 下次用同样的行为拿高分, 并避免低分的行为，这也是强化学习的核心思想。对比**监督学习**，监督学习是已经有了数据和数据对应的正确标签, 而强化学习还要更进一步, 一开始它并没有数据和标签，而是要通过一次次在环境中的尝试, 获取这些数据和标签, 然后再学习通过哪些数据能够对应哪些标签, 通过学习到的这些规律, 尽可能地选择带来高分的行为，即在强化学习中, 分数标签就是它的老师。可以看出，在强化学习中, 一种行为的分数是十分重要的，即强化学习具有分数导向性，这种分数导向性好比监督学习中的正确标签.

1、马尔科夫决策过程（MDP）

强化学习任务通常可以用马尔科夫决策过程来描述：Agent处于环境$\boldsymbol{E}$,状态空间$\boldsymbol{X}$,可采取的动作空间$\boldsymbol{A}$,潜在转移概率函数$\boldsymbol{P}$.考虑Agent初始状态为$\boldsymbol{x}$,在当前状态采取动作$\boldsymbol{a}$ 到达下一状态$\boldsymbol{x_1}$对应的转移概率为$\boldsymbol{P:x \to x_1}$ ； 此时考虑环境会根据潜在的“奖赏(Reward)”函数$\boldsymbol{R}$ 反馈给Agent一个奖赏值。则此时的四元组<$\boldsymbol{X}$,$\boldsymbol{A}$,$\boldsymbol{P}$,$\boldsymbol{R}$>则对应了强化学习任务的基本过程。

定义策略$\boldsymbol{\pi}$ 对应在当前状态$\boldsymbol{x}$ 下得到要执行的动作$\boldsymbol{a}$;

一般有两种表述方式：通常确定性策略常用$\boldsymbol{a=\pi(x)}$ 和随机性策略常用概率表述$\boldsymbol{\pi(x,a),\sum_a \pi(x,a)=1}$ 。

定义奖赏折扣$\boldsymbol{\gamma}$：$\boldsymbol{\gamma \in [0,1)}$ ，表示随着时间的推移回报的折扣。

考虑一段动态过程：Agent在状态$\boldsymbol{x_0}$,选择动作$\boldsymbol{a_0}$,根据转移概率$\boldsymbol{P_(x_0 \to x_1)}$到达状态$\boldsymbol{x_1}$,然后执行动作$\boldsymbol{a_1}$ .....即可得到：

$$\boldsymbol{x_0 \to a_0 \to x_1 \to a_1 \to x_2 \to a_2 \to x_3 \to \cdots}$$

经过上面的转移路径，我们可以得到相应的回报函数和如下：

$$\boldsymbol{R(x_0,a_0)+\gamma R(x_1,a_1)+\gamma^2 R(x_2,a_2)+\cdots}$$

如果回报函数$\boldsymbol{R}$只与$\boldsymbol{S}$有关，我们上式可重新写作:

$$\boldsymbol{R(s_0)+\gamma R(s_1)+\gamma^2 R(s_2)+\cdots}$$

我们的目标是选择一组最佳的动作，使得全部的回报加权和期望最大：

$$Reward = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

注意到$\gamma < 1$，则越靠后的状态对回报和影响越小。

1.1状态值函数

在模型已知的情况下，对任意策略π 能够估计出策略带来的期望累积奖赏。令函数$V^{\pi}(x)$ 表示从状态x 出发，使用策略π 所带来的累积奖赏；描述的是当前状态下 采取策略之后得到的奖赏，即在该状态下按照策略π 和对应的状态转移概率函数 P，对动作空间的所有可执行动作a 遍历一遍得到的期望奖赏值。

$$\begin{cases} V_T^{\pi}(x) = \mathbb{E}_{\pi}[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x], T\text{步累积奖赏}; \\ V_{\gamma}^{\pi}(x) = \mathbb{E}_{\pi}[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x], \gamma\text{折扣累积奖赏}. \end{cases}$$

注：1.状态转移概率函数指在给定状态x_0 下，执行一个给定动作，到达所有可能状态所对应的概率；

2.策略π 只是给出指定状态x_0 下，选择每一可选动作的相应概率。

1.2状态-动作值函数

函数$Q^{\pi}(x)$ 表示从状态x 出发，执行动作a 后再使用策略π 带来的累计奖赏。

$$\begin{cases} Q_T^{\pi}(x,a) = \mathbb{E}_{\pi}[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x, a_0 = a]; \\ Q_{\gamma}^{\pi}(x,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x, a_0 = a]. \end{cases}$$

令x_0 表示起始状态，a_0 表示起始状态上采取的第一个动作，对于T 步累积奖赏，用下标t 表示后续执行的步数。

Model-free 中, Agent只能按部就班, 一步一步等待环境的反馈,然后再根据反馈采取下一步行动. 而 **Model-based** 则是Agent能够通过想象来预判断接下来将要发生的所有情况,然后选择这些想象情况中最好的那种来采取下一步的策略, 这也就是围棋场上 Alpha Go 能够超越人类的原因.

2、有模型**(Model-based)**

2.1策略评估

模型已知情况下，对任意策略π 能估计出该策略带来的期望累计奖赏。

由于P, R 已知，可以对T 步累积奖赏进行动作—状态全概率展开，即从值函数的初始值V_0^{π} 出发，通过一次迭代就可以计算出每个状态的单步奖赏V_1^{π} ，进而可以从单步奖赏出发，迭代T 轮即可精确求出值函数。

动作--状态全概率展开

由于MDP具有马尔科夫性质，即系统下一时刻的状态仅由当前时刻的状态决定，不依赖于以往任何性质，于是值函数就有如下递推形式即**Bellman**等式：

$$\begin{aligned} V_T^{\pi}(x) &= \mathbb{E}_{\pi}[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x] \\ &= \mathbb{E}_{\pi}[\frac{1}{T} r_1 + \frac{T-1}{T}\frac{1}{T-1}\sum_{t=2}^{T} r_t | x_0 = x] \\ &= \sum_{a \in A}\pi(x,a)\sum_{x' \in X} P_{x \to x'}^{a}(\frac{1}{T}R_{x \to x'}^{a} + \frac{T-1}{T}\mathbb{E}_{\pi}[\frac{1}{T-1}\sum_{t=1}^{T-1} r_t | x_0 = x']) \\ &= \sum_{a \in A}\pi(x,a)\sum_{x' \in X} P_{x \to x'}^{a}(\frac{1}{T}R_{x \to x'}^{a} + \frac{T-1}{T}V_{T-1}^{\pi}(x')). \end{aligned}$$

类似的，对于γ折扣累积奖赏有

$$V_{\gamma}^{\pi}(x)=\sum_{a\in A}\pi(x,a)\sum_{x\prime\in X}P_{x\to x\prime}^{a}(R_{x\to x\prime}^{a}+\gamma V_{\gamma}^{\pi}(x\prime)).$$

注意：以上能够进行展开的基础是P和R已知。

通过以上递推公式进行计算值函数，实际就是一种动态规划算法。

输入： MDP 四元组 $E=\langle X,A,P,R\rangle$;
被评估的策略 π;
累积奖赏参数 T.
过程：
1: $\forall x\in X:\ V(x)=0$;
2: **for** $t=1,2,\ldots$ **do**
3: $\quad\forall x\in X:\ V'(x)=\sum_{a\in A}\pi(x,a)\sum_{x'\in X}P_{x\to x'}^{a}\left(\frac{1}{t}R_{x\to x'}^{a}+\frac{t-1}{t}V(x')\right)$;
4: $\quad$**if** $t=T+1$ **then**
5: $\quad\quad$**break**
6: $\quad$**else**
7: $\quad\quad V=V'$
8: $\quad$**end if**
9: **end for**
输出： 状态值函数 V

基于T步累积奖赏的策略评估算法

对于V_{γ}^{π},由于γ^{t}在t很大的时候趋于0，所以可以将过程3换成$V_{\gamma}^{\pi}(x)$的迭代公式即可实现类似的算法；通常还需要设置一个阈值θ,使得迭代能够终止，即将4处的判断条件换为$max_{x\in X}|V(x)-V\prime(x)|<\theta$.

有了状态值函数，就能直接算出状态--动作值函数：

$$\begin{cases}Q_{T}^{\pi}(x,a)=\sum_{x\prime\in X}P_{x\to x\prime}^{a}(\frac{1}{T}R_{x\to x\prime}^{a}+\frac{T-1}{T}V_{T-1}^{\pi}(x\prime));\\ Q_{\gamma}^{\pi}(x,a)=\sum_{x\prime\in X}P_{x\to x\prime}^{a}(R_{x\to x\prime}^{a}+\gamma V_{\gamma}^{\pi}(x\prime)).\end{cases}$$

2.2策略改进

按照上面的策略评估方法，可以对某个策略奖赏进行评估，若发现并非最优策略，则可以进行策略的改进。

理想最优策略应该是能够最大化累积奖赏：

$$\pi^{*}=\arg\max_{\pi}\sum_{x\in X}V^{\pi}(x).$$

一个强化学习任务可以有多个最优策略，最优策略所对应的值函数V^{*}称为最优值函数，即：

$$\forall x\in X:V^{*}(x)=V^{\pi^{*}}(x).$$

注：当策略空间无约束时的V^{*}才是最优策略对应的值函数

由于最优值函数的累积奖赏已达到最大，因此对之前的Bellman等式进行调整，即可得到对动作取最优：

$$V_{T}^{*}(x)=\max_{a\in A}\sum_{x\prime\in X}P_{x\to x\prime}^{a}(\frac{1}{T}R_{x\to x\prime}^{a}+\frac{T-1}{T}V_{T-1}^{*}(x\prime));$$

$$V_\gamma^*(x)=\max_{a\in A}\sum_{x\prime\in X}P_{x\to x\prime}^a(R_{x\to x\prime}^a+\gamma V_\gamma^*(x\prime)).$$

即

$$V^*(x)=\max_{a\in A}Q^{\pi^*}(x,a)$$

进一步可得到最优状态--动作值函数，即最优**Bellman**等式：

$$Q_T^*(x,a)=\sum_{x\prime\in X}P_{x\to x\prime}^a(\frac{1}{T}R_{x\to x\prime}^a+\frac{T-1}{T}\max_{a\prime\in A}Q_{T-1}^*(x\prime,a\prime));$$

$$Q_\gamma^*(x,a)=\sum_{x\prime\in X}P_{x\to x\prime}^a(R_{x\to x\prime}^a+\gamma\max_{a\prime\in A}Q_\gamma^*(x\prime,a\prime)).$$

注意公式推导时的代入，由于值函数对于策略的改进是单调递增的，所以对于当前策略的改进可以放心地用选取当前最优的动作来表示。

2.3策略迭代与值迭代

策略迭代 就是从一个随机策略出发，县进行策略评估，然后改进策略，评估改进后的策略，再进一步改进策略...

输入： MDP 四元组 $E=\langle X,A,P,R\rangle$;
累积奖赏参数 T.
过程：
1: $\forall x\in X:\ V(x)=0,\ \pi(x,a)=\frac{1}{|A(x)|}$;
2: **loop**
3: **for** $t=1,2,\ldots$ **do**
4: $\forall x\in X:V'(x)=\sum_{a\in A}\pi(x,a)\sum_{x'\in X}P_{x\to x'}^a\left(\frac{1}{t}R_{x\to x'}^a+\frac{t-1}{t}V(x')\right)$;
5: **if** $t=T+1$ **then**
6: **break**
7: **else**
8: $V=V'$
9: **end if**
10: **end for**
11: $\forall x\in X:\pi'(x)=\arg\max_{a\in A}Q(x,a)$;
12: **if** $\forall x:\pi'(x)=\pi(x)$ **then**
13: **break**
14: **else**
15: $\pi=\pi'$
16: **end if**
17: **end loop**
输出： 最优策略 π

基于T 步累积奖赏的策略迭代算法

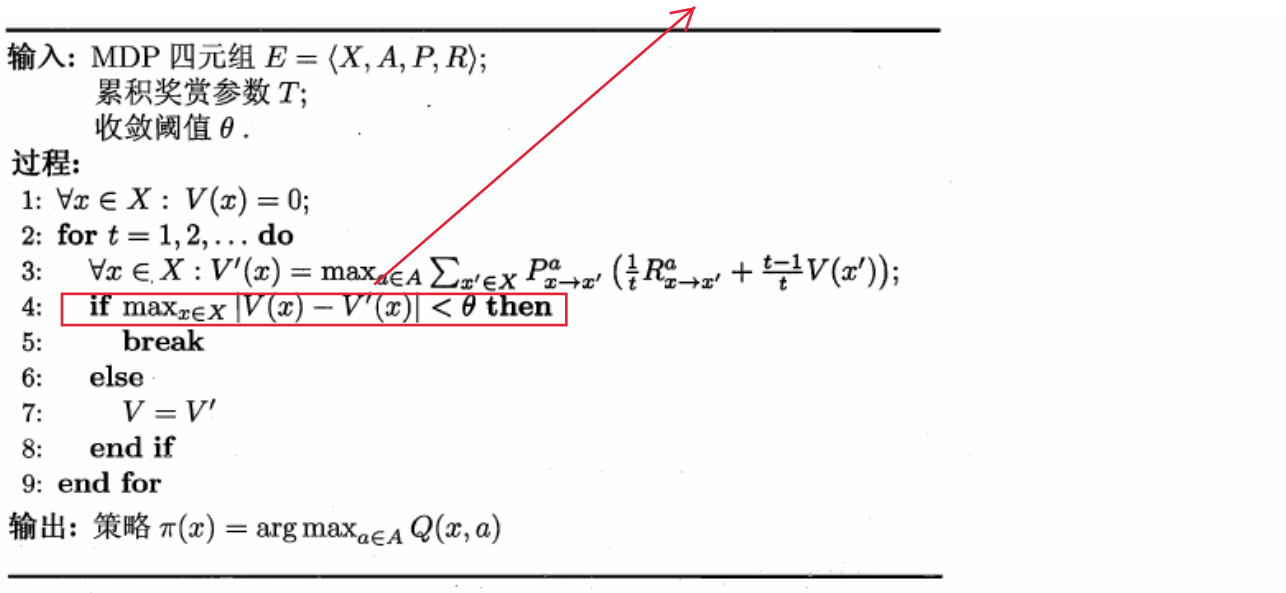
由上述流程可以看出，策略迭代算法在每次改进策略后都需要重新进行策略评估，导致耗时增加；

由公式推导可以得到：

$$V_T(x)=\max_{a\in A}\sum_{x\prime\in X}P_{x\to x\prime}^a(\frac{1}{T}R_{x\to x\prime}^a+\frac{T-1}{T}V_{T-1}(x\prime));$$

$$V_\gamma(x)=\max_{a\in A}\sum_{x\prime\in X}P^a_{x\to x\prime}(R^a_{x\to x\prime}+\gamma V_\gamma(x\prime)).$$

即策略改进和值函数的改进是一致的，因此可将策略改进视为值函数的改善，即可得到值迭代算法：



输入: MDP 四元组 $E=\langle X,A,P,R\rangle$;
累积奖赏参数 T;
收敛阈值 θ .
过程:
1: $\forall x\in X:\ V(x)=0$;
2: **for** $t=1,2,\ldots$ **do**
3: $\forall x\in X: V'(x)=\max_{a\in A}\sum_{x'\in X}P^a_{x\to x'}\left(\frac{1}{t}R^a_{x\to x'}+\frac{t-1}{t}V(x')\right)$;
4: **if** $\max_{x\in X}|V(x)-V'(x)|<\theta$ **then**
5: **break**
6: **else**
7: $V=V'$
8: **end if**
9: **end for**
输出: 策略 $\pi(x)=\arg\max_{a\in A}Q(x,a)$

基于T步累积奖赏的值迭代算法

综上可知： 在模型已知时强化学习任务能归结为基于动态规划的寻优问题，即为每个状态寻找一个最好的动作。

3、免模型(Model-free)

现实的强化学习任务中，环境的转移概率、奖赏函数甚至状态的总数往往很难得知，则学习算法无法依赖与环境建模，此时称为“免模型学习(model-free learning)”，难度要比有模型学习困难得多。

3.1蒙特卡罗强化学习

免模型情形下，策略迭代算法首先遇到的问题就是无法进行策略评估，因为模型未知情况下无法进行全概率展开；所以只能通过在环境中执行选择的动作来观察转移的状态和得到的奖赏。

另一方面，策略迭代算法估计的是状态值函数，最终策略是通过状态--动作值函数获得，当模型未知时V->Q转换会出现困难；而且，在模型未知的情况下，机器只能从任一起始状态开始探索环境，策略迭代算法是需要对每个状态进行估计，所以这种情况下就无法实现，只能在探索过程中逐渐发现各个状态并估计各状态--动作对的值函数。

一种直接的策略评估替代方法就是进行多次“采样”(采样次数为有限次，所以更适合进行T步累积奖赏的学习任务)，求取平均累计奖赏作为期望累积奖赏的近似，即**蒙特卡罗RL:** 从起始状态出发，使用某种策略进行采样，执行该策略T步并获得轨迹，对采样轨迹中的每一个状态-动作对，记录其后的奖赏之和，作为该状态-动作对的一次累积奖赏采样值，通过多次采样得到多条轨迹后，使用累积奖赏的平均作为状态-动作值函数的估计。

3.1.1同策略与异策略

输入： 环境 E;
动作空间 A;
起始状态 x_0;
策略执行步数 T.

过程：

$Q(x,a)=0,\ \mathrm{count}(x,a)=0,\ \pi(x,a)=\frac{1}{|A(x)|}$;

for $s=1,2,\ldots$ **do**

在 E 中执行策略 π 产生轨迹
$< x_0,a_0,r_1,x_1,a_1,r_2,\ldots,x_{T-1},a_{T-1},r_T,x_T >$;

for $t=0,1,\ldots,T-1$ **do**

$R=\frac{1}{T-t}\sum_{i=t+1}^{T} r_i$;

无折扣的T步累计奖赏

$Q(x_t,a_t)=\frac{Q(x_t,a_t)\times \mathrm{count}(x_t,a_t)+R}{\mathrm{count}(x_t,a_t)+1}$;

$\mathrm{count}(x_t,a_t)=\mathrm{count}(x_t,a_t)+1$

end for

对所有已见状态 x :

$$\pi(x,a)=\begin{cases}\arg\max_{a'} Q(x,a'), & \text{以概率 } 1-\epsilon;\\ \text{以均匀概率从 } A \text{ 中选取动作}, & \text{以概率 } \epsilon .\end{cases}$$

end for

输出： 策略 π

同策略蒙特卡罗强化学习算法

1.默认均匀概率选取动作，进行第一次采样产生轨迹；3.采样第s条轨迹；5.对每一个状态--动作对计算轨迹中的累积奖赏；6.更新每个 状态--动作对的平均奖赏；9.对所有已见状态进行策略改进，并用于下次数据采样产生轨迹。

同策略(on-policy)算法中用于产生一系列采样数据的策略和最终产生的策略都是$\epsilon-greedy$ 策略(除第一条采样轨迹产生的策略外)；然而策引入$\epsilon-greedy$ 策略只是为了便于评估(还可以避免陷入exploitation)并不是为了最终使用，实际希望改进的是原始策略，所以在策略评估的时候引入$\epsilon-greedy$，而在策略改进时改进原始策略的方法即是**异策略(off-policy)**.

输入： 环境 E;
动作空间 A;
起始状态 x_0;
策略执行步数 T.

过程：

1: $Q(x,a)=0,\ \mathrm{count}(x,a)=0,\ \pi(x,a)=\frac{1}{|A(x)|}$;
2: **for** $s=1,2,\ldots$ **do**
3: 在 E 中执行 π 的 ϵ-贪心策略产生轨迹
$<x_0,a_0,r_1,x_1,a_1,r_2,\ldots,x_{T-1},a_{T-1},r_T,x_T>$;
4: $p_i=\begin{cases}1-\epsilon+\epsilon/|A|, & a_i=\pi(x);\\ \epsilon/|A|, & a_i\neq\pi(x),\end{cases}$
5: **for** $t=0,1,\ldots,T-1$ **do**
6: $R=\frac{1}{T-t}\sum_{i=t+1}^{T}(r_i\times\prod_{j=i}^{T-1}\frac{1}{p_j})$;
7: $Q(x_t,a_t)=\frac{Q(x_t,a_t)\times\mathrm{count}(x_t,a_t)+R}{\mathrm{count}(x_t,a_t)+1}$;
8: $\mathrm{count}(x_t,a_t)=\mathrm{count}(x_t,a_t)+1$
9: **end for**
10: $\pi(x)=\arg\max_{a'}Q(x,a')$
11: **end for**

输出： 策略 π

注：以概率epsilon采取的策略为对备选动作库进行等概率处理，所以二者决策动作不同时，产生动作的概率为epsilon/|A|，当二者相同时即概率为二者概率之和

为什么是除以p_j而不是乘？蒙特卡洛重要性采样的概念，要除去重要性采样权重

异策略蒙特卡罗强化学习算法

注：4.采取$\epsilon-soft$ 策略；6.计算的R 是修正的累积奖赏（需要用到两种策略产生同一条采样轨迹的概率比值）

3.2时序差分学习

蒙特卡罗强化学习算法通过考虑采样轨迹(在完成一个采样轨迹之后再更新策略的值估计),克服模型未知给策略估计造成的困难，与基于动态规划的策略迭代和值迭代算法在每执行一步策略后就进行值函数更新的算法相比，效率低得多。**时序差分(TD)学习** 则结合了动态规划与蒙特卡罗方法的思想，得到更高效的免模型学习。

蒙特卡罗强化学习算法的本质，是通过多次尝试后求平均值来作为未来期望累计奖赏的近似，但是在求平均的时候采用的是"批处理式"，即在一次完整的采样轨迹完成后再对所有的状态--动作对进行更新，实际这个过程是可以增量更新的：

$$Q_{t+1}^{\pi}(x,a)=Q_t^{\pi}(x,a)+\frac{1}{t+1}(r_{t+1}-Q_t^{\pi}(x,a))$$

里面的减号是因为外面的Q_t{π}系数本来为t/t+1,而不是1

更一般的，将$\frac{1}{t+1}$ 替换成系数α_{t+1} ,实践中通常都是令α_t 为一个较小的正数值α ,这样做不会影响Q_t 是累积奖赏之和的性质.更新步长α 越大，则会使靠后的累积奖赏更重要。

以γ 折扣累积奖赏为例，利用动态规划方法且考虑模型未知时使用状态--动作值函数更为方便：

$$\begin{aligned}Q^{\pi}(x,a)&=\sum_{x\prime\in X}P_{x\to x\prime}^{a}(R_{x\to x\prime}^{a}+\gamma V^{\pi}(x\prime))\\&=\sum_{x\prime\in X}P_{x\to x\prime}^{a}(R_{x\to x\prime}^{a}+\gamma\sum_{a\prime\in A}\pi(x\prime,a\prime)Q^{\pi}(x\prime,a\prime))\end{aligned}$$

增量求和如下：

$$Q_{t+1}^{\pi}(x,a) = Q_t^{\pi}(x,a) + \alpha(R_{x\to x\prime}^{a} + \gamma Q_t^{\pi}(x\prime,a\prime) - Q_t^{\pi}(x,a)).$$

注：$x\prime$ 是前一次在状态x 执行动作a 后转移的状态，$a\prime$ 是策略π 在$x\prime$ 上选择的动作。

两个典型的算法是**Sarsa**和**Q-Learning** ,整个算法就是一直不断更新 Q table 里的值, 然后再根据更新之后的值来判断要在某个 state 采取怎样的 action.

3.2.1**Sarsa**

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  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)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

1.先在未更新Q值表前计算利用策略找出动作集下一状态，用于迭代更新；
2.参与更新的Q值是从当前Q值表中选出最大的状态-动作值。

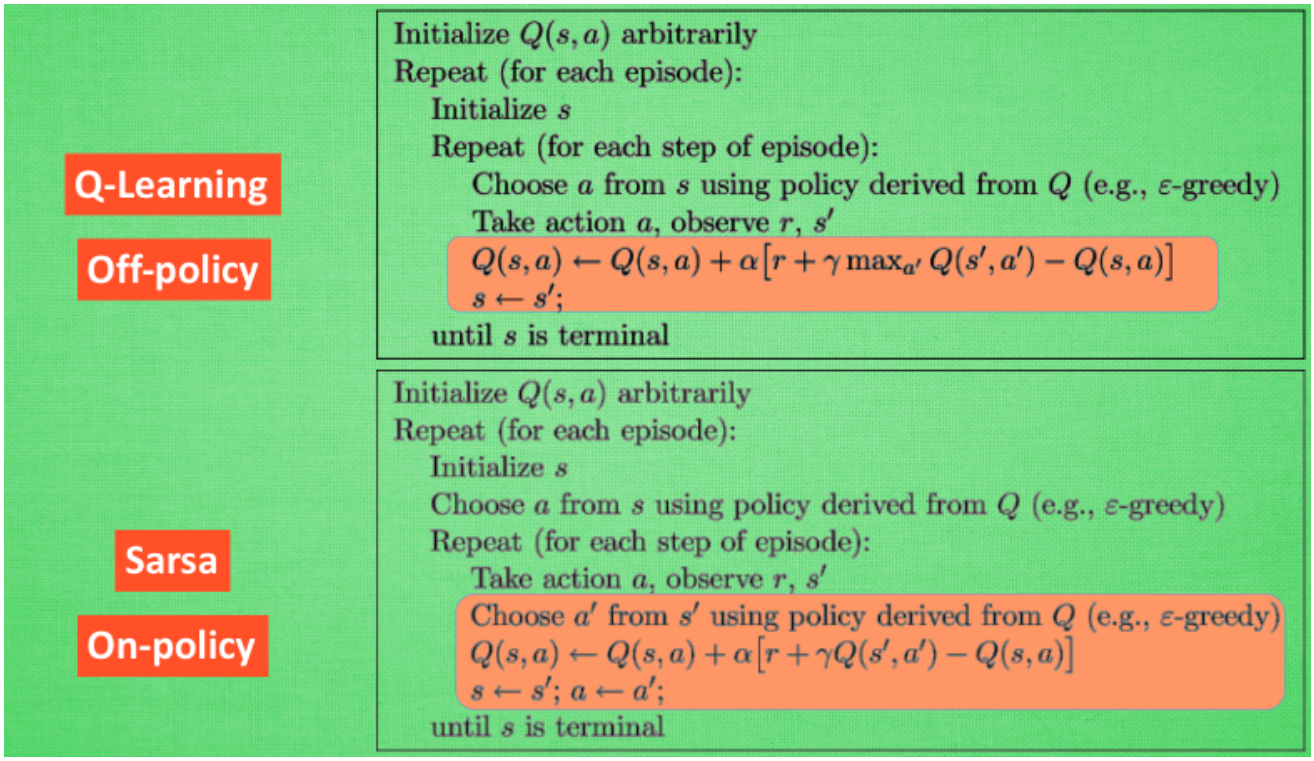
3.2.2**Q-learing**

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

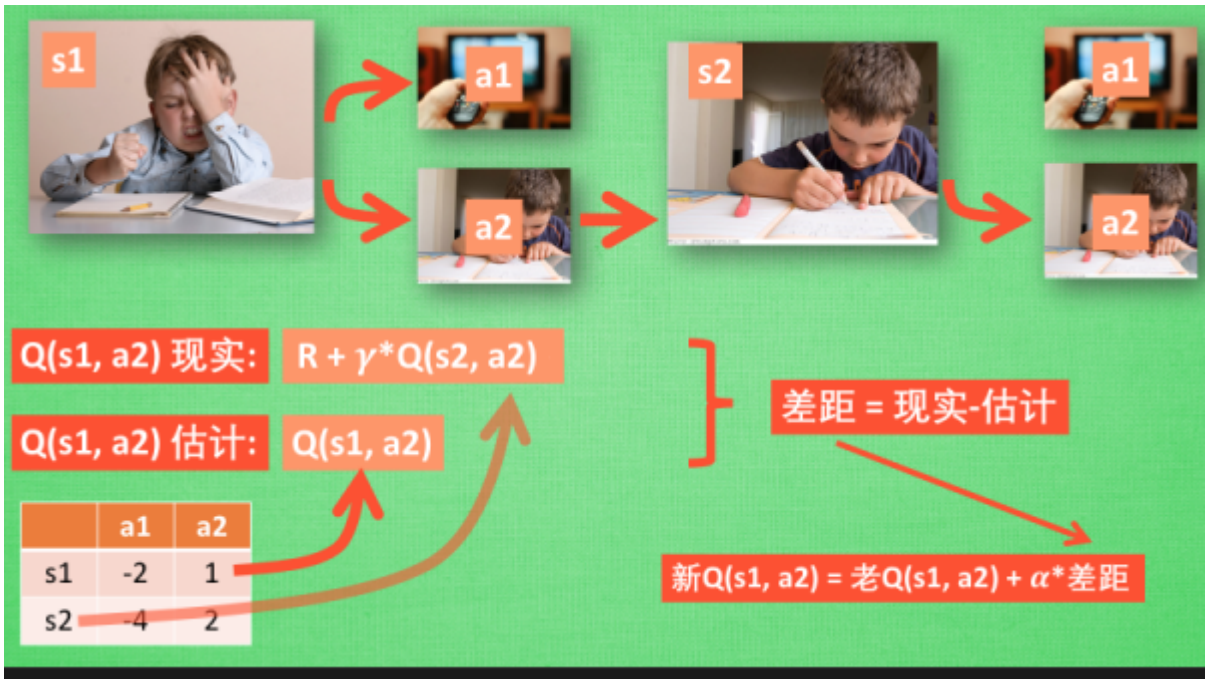
3.2.3**on-policy**和**off-policy**

Q-learning 是一个 off-policy 的算法, 因为里面的 `max` action 让 Q table 的更新可以不基于正在经历的经验(可以是现在学习着很久以前的经验,甚至是学习他人的经验).

Sarsa 的整个循环都将是在一个路径上, 也就是 on-policy, 下一个 state, 和下一个 action 将会变成他真正采取的 action 和 state. 和 Q-learning 的不同之处就在于此： Q-learning 的下个一个 state、 action 在算法更新的时候都还是不确定的 (off-policy)，而是选取使Q函数值最大的那个action. 而 Sarsa 的 state, *action* 在这次算法更新的时候已经确定好了 (on-policy).具体参见下述算法中二者对于$Q(s,a)$ 的迭代表达式。



Sarsa 相对于 Q-learning比较胆小，因为 Q-learning 永远都是想着 `maxQ` 最大化, 而不考虑其他非 `maxQ` 的结果. 我们可以理解成 Q-learning 是一种贪婪, 大胆的算法. 而 Sarsa 是一种保守的算法, 他在乎每一步决策, 对于错误比较敏感.

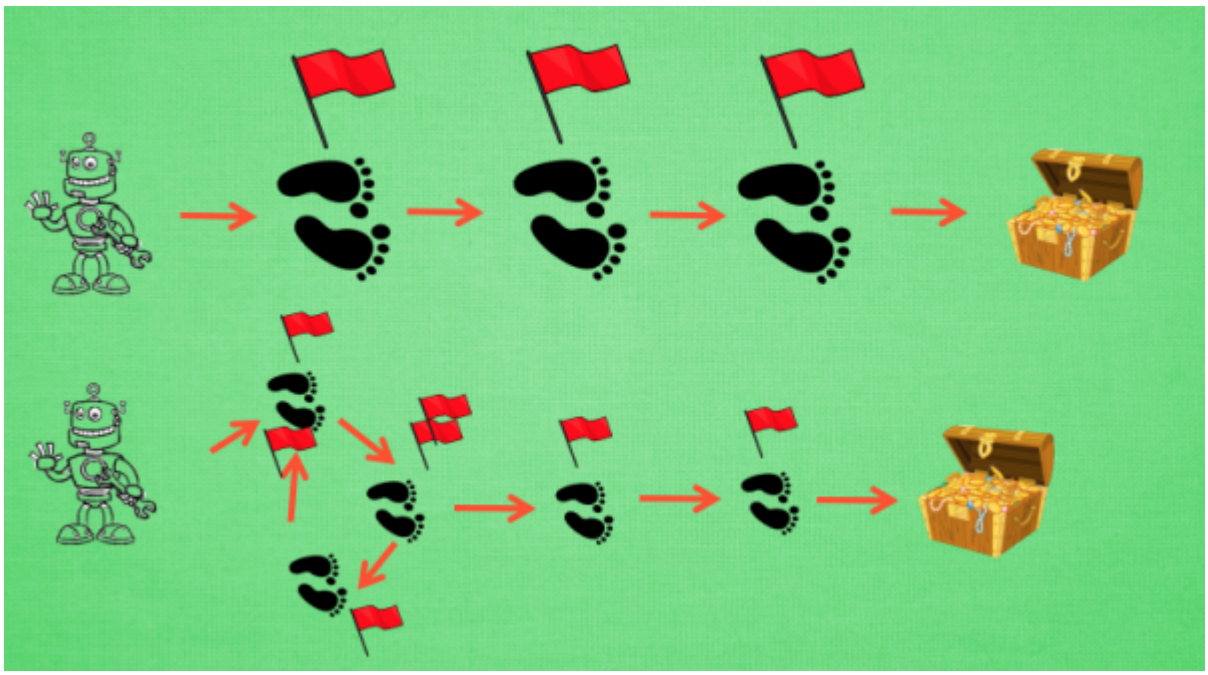


如上图，我们会经历正在写作业的状态 s1, 然后再挑选一个带来最大潜在奖励的动作 a2, 这样我们就到达了"继续写作业状态" s2, 而在这一步, 如果用的是 Q-learning, 就会观看一下在 s2 上选取哪一个动作会带来最大的奖励, **但是在真正要做决定时, 却不一定会选取到那个带来最大奖励的动作**, Q-learning 在这一步只是估计了一下接下来的动作值，而迭代动作的选取还是跟最初的策略相同. 而 Sarsa 是实践派, 他说到做到, 在 s2 这一步估算的动作也是接下来要做的动作，算法最后的更新是s和a共同的更新.两种算法都有他们的好处, 比如在实际中, 你比较在乎机器的损害, 用一种保守的算法, 在训练时就能减少损坏的次数.

3.2.4**Sarsa(lambda)**

Sarsa 是一种单步更新法, 在环境中每走一步, 更新一次自己的行为准则, 我们可以在这样的 Sarsa 后面打一个括号, 说他是 Sarsa(0), 因为他等走完这一步以后直接更新行为准则. 如果延续这种想法, 走完这步, 再走一步, 然后再更新, 我们可以叫他 Sarsa(1). 同理, 如果等待回合完毕我们一次性再更新呢, 比如这回合我们走了 n 步, 那我们就叫 Sarsa(n). 为了统一这样的流程, 我们就有了一个 lambda 值来代替我们想要选择的步数, 这也就是 Sarsa(lambda) 的由来.

1）**单步更新和回合更新：**虽然我们每一步都在更新, 但是在没有获取宝藏的时候, 我们现在站着的这一步也没有得到任何更新, 也就是直到获取宝藏时, 我们才为获取到宝藏的上一步更新为: 这一步很好, 和获取宝藏是有关联的, 而之前为了获取宝藏所走的所有步都被认为和获取宝藏没关系. 回合更新虽然我要等到这回合结束, 才开始对本回合所经历的所有步都添加更新, 但是这所有的步都是和宝藏有关系的, 都是为了得到宝藏需要学习的步, 所以每一个脚印在下回合被选则的几率又高了一些. 在这种角度来看, 回合更新似乎会有效率一些.



我们看看这种情况, 还是使用单步更新的方法在每一步都进行更新, 但是同时记下之前的寻宝之路. 你可以想像, 每走一步, 插上一个小旗子, 这样我们就能清楚的知道除了最近的一步, 找到宝物时还需要更新哪些步了. 不过, 有时候情况可能没有这么乐观. 开始的几次, 因为完全没有头绪, 我可能在原地打转了很久, 然后才找到宝藏, 那些重复的脚步真的对我拿到宝藏很有必要吗? 答案我们都知道. 所以Sarsa(lambda)就来拯救你啦.

2）**lambda含义：**其实 lambda 就是一个衰变值, 他可以让你知道离奖励越远的步可能并不是让你最快拿到奖励的步, 所以我们想象我们站在宝藏的位置, 回头看看我们走过的寻宝之路, 离宝藏越近的脚印越看得清, 远处的脚印太渺小, 我们都很难看清, 那我们就索性记下离宝藏越近的脚印越重要, 越需要被好好的更新. 和之前我们提到过的 奖励衰减值 gamma 一样, lambda 是脚步衰减值, 都是一个在 0 和 1 之间的数.

3）**lambda取值：**当 lambda 取0, 就变成了 Sarsa 的单步更新, 当 lambda 取 1, 就变成了回合更新, 对所有步更新的力度都是一样. 当 lambda 在 0 和 1 之间, 取值越大, 离宝藏越近的步更新力度越大. 这样我们就不用受限于单步更新的每次只能更新最近的一步, 我们可以更有效率的更新所有相关步了.

4、在线学习和离线学习

强化学习还可以分为在线学习和离线学习, 所谓在线学习, 就是Agent边玩边学习, 而离线学习是从过往的经验中学习, 但是这些过往的经历没必要是自己的经历, 任何人的经历都能被学习. 或者我也不必要边玩边学习, 我可以白天先存储下来玩耍时的记忆, 然后晚上通过离线学习来学习白天的记忆.

In offline learning, the whole training data must be available at the time of model training. Only when training is completed can the model be used for predicting.

在离线学习中，整个训练数据必须在模型训练时可用。 只有在训练完成后，模型才能用于预测

In contrast, online algorithms process data sequentially. They produce a model and put it in operation without having the complete training dataset available at the beginning. The model is continuously updated during operation as more training data arrives.

相反，在线算法按顺序处理数据。 他们产生一个模型，并在开始时没有提供完整的训练数据集的情况下投入运行。 随着更多训练数据的到来，模型在运行过程中不断更新。

最典型的在线学习就是 Sarsa 了, 还有一种优化 Sarsa 的算法, 叫做 Sarsa lambda, 最典型的离线学习就是 Q learning, 后来人也根据离线学习的属性, 开发了更强大的算法, 比如让计算机学会玩电动的 Deep-Q-Network.