# 第一章：强化学习是什么

1. 强化学习的主要目的是研究并解决机器人智能体**贯序决策**问题。把“贯序决策”翻译成“白话”就是：强化学习希望机器人或者智能体在一个环境中，随着“时间的流逝”，不断地自我学习，并最终在这个环境中学到一套最为合理的行为策略

2. 强化学习设定的研究对象：

（1）Agent:：我们放到环境中去探索和学习用的机器人主体；

（2）Environment：机器人所处的环境（如游戏程序等其他复杂的环境）；

（3）Observation：观测，是指Agent能够观测（感知）到的环境信息。智能体能够感知什么，就把什么当成Observation即可；

（4）Action：指由Agent发出的行为和动作，以及Agent与Environment之间发生的动作交互。Action的内容定义很丰富，可以描述成是否做某个动作（用类似1、0等来表示），也可以描述成一个连续的向量（例如 [0.35,0.21,0.7]），关键看机器人是什么类型的；

（5）Reward：奖励值/回报值，在强化学习中，Reward是指Agent在做了一个动作后（或者在某一时刻）得到的用来描述其行为好坏的值，它是一个“超参数”：什么是超参数呢？就是无法通过训练自动学会的参数。超参数必须在整个训练开始之前确定，因为在训练过程中没有办法通过学习来确定这个值。

3. 机器人在一个环境中不断地试探，就会不断地得到这样一个短序列：St→At→Rt→St+1.

从Agent的角度看，做一个动作At，得到一个观测值Ot，以及一个标量奖励值Rt。观测值Ot可以通过另一个概念来几乎等同地理解，这个概念就是状态（State），写作或St。中e是指特定的Environment，t是指某一时刻，意思就是：在某一时刻所观测到的状态描述信息。

# 第二章：强化学习的脉络

1.策略（Policy）的描述：

（1），a指动作，s指状态，函数π（跟圆周率没有半点关系）就是用来描述策略的。从形式上看，策略就像一个函数，只要输入一个状态s，就输出一个动作a。

（2），这个θ仅仅为了说明在策略函数π中有参数θ。Θ不会太简单，不是用一两个实数就能描述清楚，在强化学习中是通过训练得到的。

（3），这个公式表示的是输入状态s后，策略输出的a不是一个确定的值，而是一个概率分布的描述。而概率的描述，就在后面这个部分P[At=a|St=s]。也就是说，当输入一个状态St=s的时候，应该输出的动作At=a的概率为多少。

2. 如何得到一个好的策略：评分函数

（1）**直接法**：通过优化策略函数a=π(s|θ) 的待定系数的方法来获得高回报值的数学期望值。思路如下：

策略函数：，在输入状态s确定的情况下，输出的动作a其实只和后面的参数θ有关；在环境中得分（奖励值）：，在一个确定的环境中，通常在一个确定的状态s下输入一个确定的动作a，就会有得分产生。

两者结合：，即在策略π确定的情况下，只要取一个参数θ的值，然后输入一个状态s，就会得到一个得分大小的描述。

一个策略究竟能得多少分？由于每个状态出现的概率不是均等的，所以可以求 其数学期望（加权平均值），而非均值：

对每个状态值si，用它们各自的概率P(si)，乘以策略输出的动作a的概率P(a|si)，再乘以在R(a|si)状态下做这个动作的评分值，最后计算它们的算术平均值，就能得到对一个策略的相对客观、公正的评分了。既然评分函数有了，就要想办法（最优化等方法）通过让θ取某个值，让整个函数取得最大值。

（2）**间接法**： 哪个状态s更好，哪个动作a就更好。思路如下：

策略实际上在做：当机器人所处的状态为某个状态的时候，选择输出一个动作a，然后迁移到下一个状态s。所以就有了非常自然的思路：哪个状态s更好，哪个动作a就更好。

对陌生的场景来说，只有当前状态st、动作a、下一个状态st+1、获得的奖励值r这些可以瞬时观察到的值。但是瞬时奖励值的大小，其实不能真实地反映这个动作及其所对应转移到的状态的价值。因为在这个状态的后面，可能隐藏着时间更为久远、尺度更为巨大的奖励或惩罚，而这些是无法通过当时的奖励值看出来的。

因此，我们通常会借助另一个函数——**值函数（Value Function）**来描述并期望尽可能精确地描述这个状态的价值。值函数的表达式可以这样写：

v就是价值（Value）的意思。这个表达式的含义也比较清楚：vπ(s)表示，在策略π确定的情况下，状态s的值为多少（价值为多少的概率；价值越高，状态就越好）；在等号的右边是计算方法，Eπ[Rt+1+γRt+2+γ2Rt+3+...|St=s]指的是将下个时间片的奖励值、下下个时间片的奖励值、下下下个时间片的奖励值……相加，然后得出一个用于评估这个状态的价值的数学期望值。

3. 马尔可夫决策过程：

“马尔可夫”，通常意味着：在一系列事件中，某给定事件发生的概率，只取决于前面发生的事件。（更久以前发生的事件不在研究范围内，只关注前面发生的事件，只针对前面发生的事件和现在发生的事件的关系来做研究。）

马尔可夫决策过程中的环境，需要满足条件

这个等式的含义是，前一个状态St到后一个状态St+1的转移概率，不会随着更久以前的状态的加入而改变。如果一个环境中的状态变化满足这样的条件，就可以称状态St是马尔可夫的（或者是满足马尔可夫的）

4.策略与评价

（1）基于策略的某个状态的价值表达式：

在策略π驱使下的状态s的价值，可以通过等号右边的部分计算出来；等号右边的部分，表示把所有的动作以策略所规定的输出概率π(a|s)取一遍，然后分别乘以各自的估值qπ(s,a)。qπ(s,a)表示，在策略π的驱使下，以状态s做动作a的估值（这个估值就是价值估值）。在强化学习中，q的含义是没有二义性的，就是指动作的估值（与vπ(s)的视角不同，vπ(s)是状态的估值）。**即：是动作的估值，是状态的估值。**

（2）的计算方式：

是通过加权平均的方法算出来的价值（状态乘以该状态奖励值），即当下获得的价值（现世报）；加号后部分表示一个打了折扣（代表折扣）的远期估值：到达每个状态s′的概率分别乘以其状态估值的这样一个求和的数学期望值。

从1.和2.中，vπ里面套着qπ,qπ里面套着vπ，没完没了。其实，可以用其中一个替换另一个，能通过传统的解方程的方法——只剩一个未知数，解起来似乎容易一些。

用qπ来代换vπ：

让人沮丧的是，这个方程很难通过传统解方程的方法解出来。一般来说，一个好的策略π，需要通过值迭代、策略迭代等方法找到。

（3）策略迭代

以某种策略开始在环境中做动作，会得到相应的vπ(s)，当然，也可以得到相应的qπ(s,a)。—>在一个确定的策略下，每一次都挑选qπ(s,a)估值最大的动作作为新的策略，然后再来一次。我们将新策略命名为π′。—>使用π′在环境里做动作，会得到新的v′π(s)和q′π(s,a)，而且比前一个策略π要好（因为每次都会选择比原来的估值大的动作前往估值大的状态）。一个迭代结束后，还会产生π′′、π′′′这样的策略。只要将这个过程持续下去，总会收敛到最优策略上。

5. Model-Based & Model-Free

（1）Model-Based 基于模型

这里的模型是指，在一个环境中各个状态之间转换的概率分布描述。（表中每个小格子的值是在一个状态下做某个动作后转移到另一个状态的概率）

可以做什么？不管是否满足马尔可夫特性，只要有这么一个模型存在，就必然能够在初始状态确定的情况下找出一条满足特定要求的路径。只要模型是确定的，转移概率就是确定的。只要转移概率是确定的，我们就能知道对应于S→S′ 的转移概率，以及在状态S下做什么样的动作A会有最高的回报值。这样，一个状态的估值vπ(s)，以及一个状态下的动作估值qπ(s,a)，就比较容易被准确地估计出来了。

如果有这样一个模型，我们能清楚地知道在这个过程中获得的奖励值的数学期望值，那么，这个问题就是一个规划问题。那就把它当成规划问题来解决（遍历树、解方程、解不等式等等），不需要使用强化学习算法。

（2）Model-Free 无模型

有人帮我们做好模型的情况很少出现。在绝大多数场景中，模型是未知的，至少是未曾精确量化的。怎么办？其实，这才是强化学习希望得到的更具普适性的方式和方法。

例如，Q-Learning是通过不断求解一个状态下的动作估值函数Q(s,a)来进行策略学习的，它并没有采用先根据统计结果做出一个模型再做规划的方法，而是直接以类似查表的方法，估算Q(s,a)中每个“小格子”的值，从而进行建模和求解的。

# 第三章：动态规划（DP）

用于准确估算一个状态的价值vπ(s)或者在一个状态下做某个动作的价值qπ(s,a)的思维方法一：动态规划（DP），也称为值迭代。

动态规划是一种建模和解题的思路。在这种思路的指导下，原始的问题（大的、复杂的问题）会被分解成多个可解的且结果可保存的子问题。一旦所有的子问题获解，原始的问题就获解了。

我们用vπ来代换qπ得到：

这样一来，求解vπ(s)就依赖于 π(a|s)、、 和v(s′)的准确值了。

：按照策略π，输入s后输出动作a的概率

：做动作a可以立刻得到的奖励值

：做动作a转移到状态s′的概率

：转移后的状态s′的估值

由此可以明确两件事：1）通过类似于递归求解的方式，逐层估算，如果每一层的估算都是准确的，就能迭代向上传递，把上面各层的各个状态的值估算准确。2）这个方法严重依赖模型，**要求研究对象满足MDP。**

动态规划的优化步骤：

1）以一种策略π开始在环境中做动作，这个时候，就会得到相应的vπ(s)和qπ(s,a)。这个步骤与策略迭代的第一步没什么区别。

2）因为在时序上属于树遍历问题，所以，要想准确估算树上部的状态节点的值，就要先估算树下部的状态节点的值。从下向上，逐层估算vπ(s)，用类似递归的方式向上传递。表达式写出来是这样的：

的意思是，在众多不同的a里，只取那个能保证后面的外层括号里的表达式的值最大的a。也就是说，如果知道有一个动作比其他动作都要好，为什么还要做其他动作？

表示，一个由动作a引发的“现世报”的奖励值加上它达到各个状态的概率和各个状态的估值。

3）持续以这个逻辑更新，直到vπ(s)不再发生变化。此时的策略，就是每次都选择能够导致状态迁移到最有价值状态的动作a。

# 第四章：蒙特卡罗法

蒙特卡罗法（MC方法），也叫作统计模拟方法，是一种以概率统计理论为指导的非常重要的数值计算方法。蒙特卡罗法是指使用随机数（或者更常见的伪随机数）来解决很多计算问题的方法，通过反复实验进行统计，用这个统计的近似值来代替模型中相应的真实值。

对于不能提供Model的环境，蒙特卡罗就可以派上用场。

1）机器人以策略π（即使这个策略非常不靠谱都没关系）做动作，它在起始状态s1下，一定能够走出一条路：s1→a1→r1→s2→a2→r2→s3→. . . →sT.

最后一个状态sT，对一个Episode（片段）来说，就是终止状态了（不管是通关了，还是“死翘翘”了）。

2）在这样一个序列产生以后，应该可以试着对状态s进行估值：，对每个序列中的每个值进行价值评估，要尽可能多的计算。每个得到的状态，都有到达的次数和相应的价值估值，因此可以对它们求加权平均值。

我们知道，一个状态的价值需要通过类似加权平均的概念才能获得。在蒙特卡罗法中，要想计算一个估值，也应该采用类似的方法。应该这样写：

这个表达式表示，对状态s的估值，需要在策略π的驱动下去求数学期望值。根据大数定律，随着的计算次数增多，所得到的加权平均值会向数学期望值逐步靠近。

1. 两种估值方法

（1）First-Visit Monte-Carlo Policy Evaluation（首次访问蒙特卡罗策略估值）

对任意状态s来说，如果在一个完整的Episode中第一次碰到它，就对计数器N(s)进行加1的操作。在对N(s)做加1操作的同时，需要对另一个用于记录总体奖励值的函数S(s)进行加Gt的操作：

只要这么持续下去，根据大数定律，最终V(s)肯定无限趋近于它的数学期望值。

（2）Every-Visit Monte-Carlo Policy Evaluation（每次访问蒙特卡罗策略估值）

对每一次的访问策略，采用“一视同仁”的方式，即，第1次碰到状态s1要做评估，第2次碰到状态s1也要做评估，即便是第100次碰到状态s1，该怎么评估，就怎么评估。

同样按照计算的方式分别对多次遇到的s1做估值计算即可。然后，依然按照之前的原则对s1的估值进行更新计算。

（3）增量平均

首次访问和每次访问，都可以通过在程序里设置计数器的方式解决求和及求平均值的问题。而在程序中，为了使得计算方便、表达简洁，通常会把这个更新公式写成它的增量平均（Incremental Mean）形式。可以用这样一个公式进行更新：

公式中凭空出现的a，经过推导发现就是

1. 优点与缺陷

蒙特卡罗法的过程，是一个通过不断采样让样本丰富起来，进而比较准确地估算出由策略π驱使的某个状态的估值vπ(s)的过程。通过多次实验进行统计，计算状态估值，一旦状态估值v(s)能够估算准确，就能评价一个动作的估值q(s,a)。

蒙特卡罗法的优势在于，它不是以模型为基础的，而是通过多次试探求平均值的方法让估值逐步变得准确的。

蒙特卡罗法的缺陷在于必须等待一个Episode结束才能开始进行估算（这样的估算效率不是很高）。能不能更快、更直接地得到这些状态的估值，从而高频度地进行策略优化呢？这是一个新的问题，答案也是有的，就是第5章中将要介绍的时间差分。

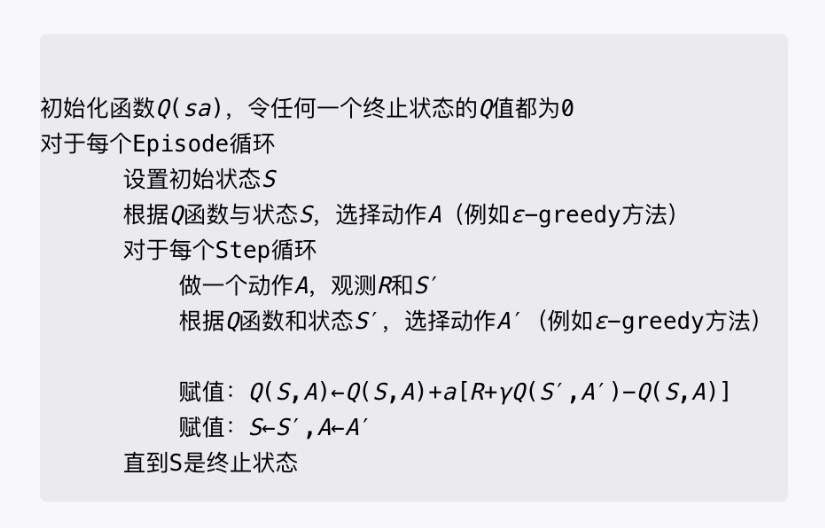
# 第五章：时间差分（TD）

时间差分法的思路与蒙特卡罗法不同，它不希望机器人通过一个完整的Episode试探之后才对一个状态进行估算，它可以走一步就估算一次。时间差分法中两个最为经典的算法，一个是SARSA 算法，另一个是Q-Learning算法。

1. SARSA算法

SARSA算法的计算逻辑：在状态S下，要想把动作A的估值Q(S,A)计算准确，需要研究：S→A→R→S′→A′这个序列，并对这个序列中的估值进行调整，从而达到使贝尔曼方程收敛的目的。更新方程是这样的：

意思是：SARSA算法的表现形式是Q(S,A)，即对某个状态下的一个动作做估值计算。在状态S下，做一个动作A，它的估值Q(S,A)需要的迭代计算逻辑是当前的Q(S,A)与一个a倍的叠加值相加。a在第四章见过，即1/N(s)（a的值越小，说明“记忆”的周期越长，也可以说，对于估值，“平均”的周期越长。）叠加值（R+γQ(S′,A′)−Q(S,A)）是用瞬时回报R和γ（折扣系数，表示对于长期回报的好恶）倍的下一个状态下的动作估值Q(S′,A′)相减得到的。【SARSA算法伪代码】



SARSA算法的具体流程如下：

1）随机初始化所有的状态和动作对应的价值函数q，对于**终止状态的q值初始化为0**。

2）开始迭代，初始化S为当前状态序列的第一个状态，设置A为 ϵ - greedy在当前状态 S 选择的动作。

【ε-greedy方法的要点如下。】

· 令ε取一个0和1之间的数字。

· 以1− ε的概率，让机器人做的动作，也就是Q(s,a)中能够产生最高估值的动作a。

· 以ε的概率进行试探，随机做一个动作，以兼顾试探和最优动作选择的方式不断提高机器人输出动作的质量。该要点和上一个涉及同一个步骤，只不过，在一次行动中只能二选一。

3）在状态S下执行当前动作A，得到新状态S′和奖励R。

4）用ϵ - greedy在状态 S′选择新的动作A′。

5）更新价值函数q(S,A)：q(S,A)=q(S,A)+αR+γq(S′,A′)−q(S,A)

6）一直迭代直至所有的q(S,A)收敛，在这里步长 α 一般需要随着迭代的进行逐渐变小，这样才能保证动作价值函数收敛。

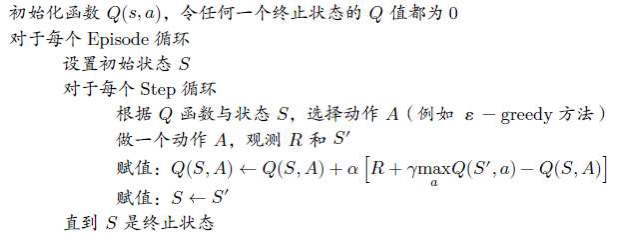
1. Q-Learning算法

Q-Learning算法：

SARSA算法：

Q-Learning算法对Q(S,A)的估值，同样参考了其后所能转移到的所有状态S′可能做的所有动作a′，并认为其中的**最大估值才是当前Q(S,A)的估值**。

【Q-Learning算法伪代码】



与SARSA算法的主要差别就在于对估值函数进行更新的表达式上。

【算法缺陷】：对一个状态，要想估算它的价值，尚且需要使用数学期望值这样的概念，通过多次测量取平均值，因此，单纯以最大值来做评价是有失偏颇的。获得最大值的动作可能是一个小概率事件，这涉及到一个概率问题，所以Q-Learning算法有暂时性的**“过估计”**的问题。但这个问题在目前的Q-Learning算法中不是永久存在的，而且在这个问题被发现之后，深度强化学习的DQN算法族也采用了很多方法来应对。

1. On-Policy和Off-Policy

在强化学习的范畴内：

“On-Policy”：在训练中用来计算估值并训练的策略和它所采用的所有Transition的策略是同一套策略。（如SARSA算法）

“Off-Policy”：在训练中用来计算估值并训练的策略和它所采用的所有Transition的策略是不同的。（如Q-Learning算法）

1. On-line学习和Off-line学习

On-line学习：一个机器人程序只能一边玩游戏，一边忘记以前的，一边存储最新的，一边更新贝尔曼方程。

Off-line学习：有一个无穷大的或者非常丰富的Transition样本库，一个机器人程序在一个海量存储环境中大批量地做贝尔曼方程的更新，并且只做这件事。

# 莫烦.RL

1. 强化学习方法分类汇总：
2. 不理解环境（Model-Free RL）：不尝试理解环境，环境给了什么就是什么；机器人只能按部就班一步一步等待真实世界的反馈，再根据反馈采取下一步的行动

理解环境（Model-Based RL）：学会了用一种模型来模拟环境；能够通过想象来预判断接下来要发生的所有情况，然后根据这些想象中的情况选择最好的那种，并根据这种情况来采取下一步的策略

1. 基于概率（Policy-Based RL）：通过感官分析所处的环境，直接输出下一步采取的各种行动的概率，然后根据概率采取行动，所以每种动作都有可能被选中，只是可能性不同；用一个概率分布在连续动作中选择特定的动作

基于价值（Value-Based RL）：通过感官分析所处的环境，直接输出所有动作的价值，我们会选择价值最高的那个动作；对于连续的动作无能为力

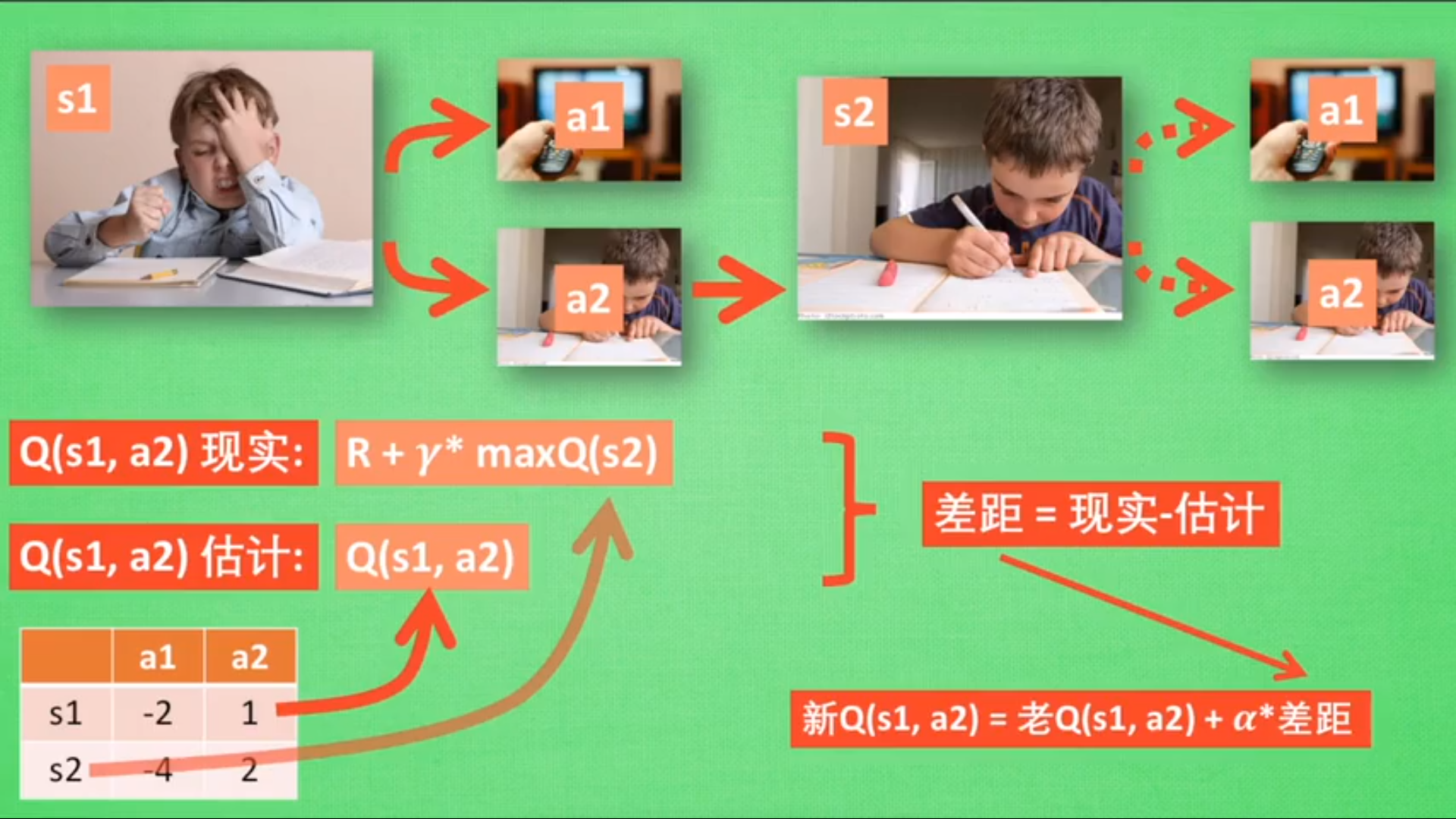
1. 回合更新（Monte-Carlo update）：假设强化学习是一个玩游戏的过程。游戏开始后需要等待游戏结束，然后再总结，再更新我们的行为准则

单步更新（Temporal-Difference update）：在游戏进行中的每一步都在更新，不用等待游戏的结束，这样就能边玩边学习了

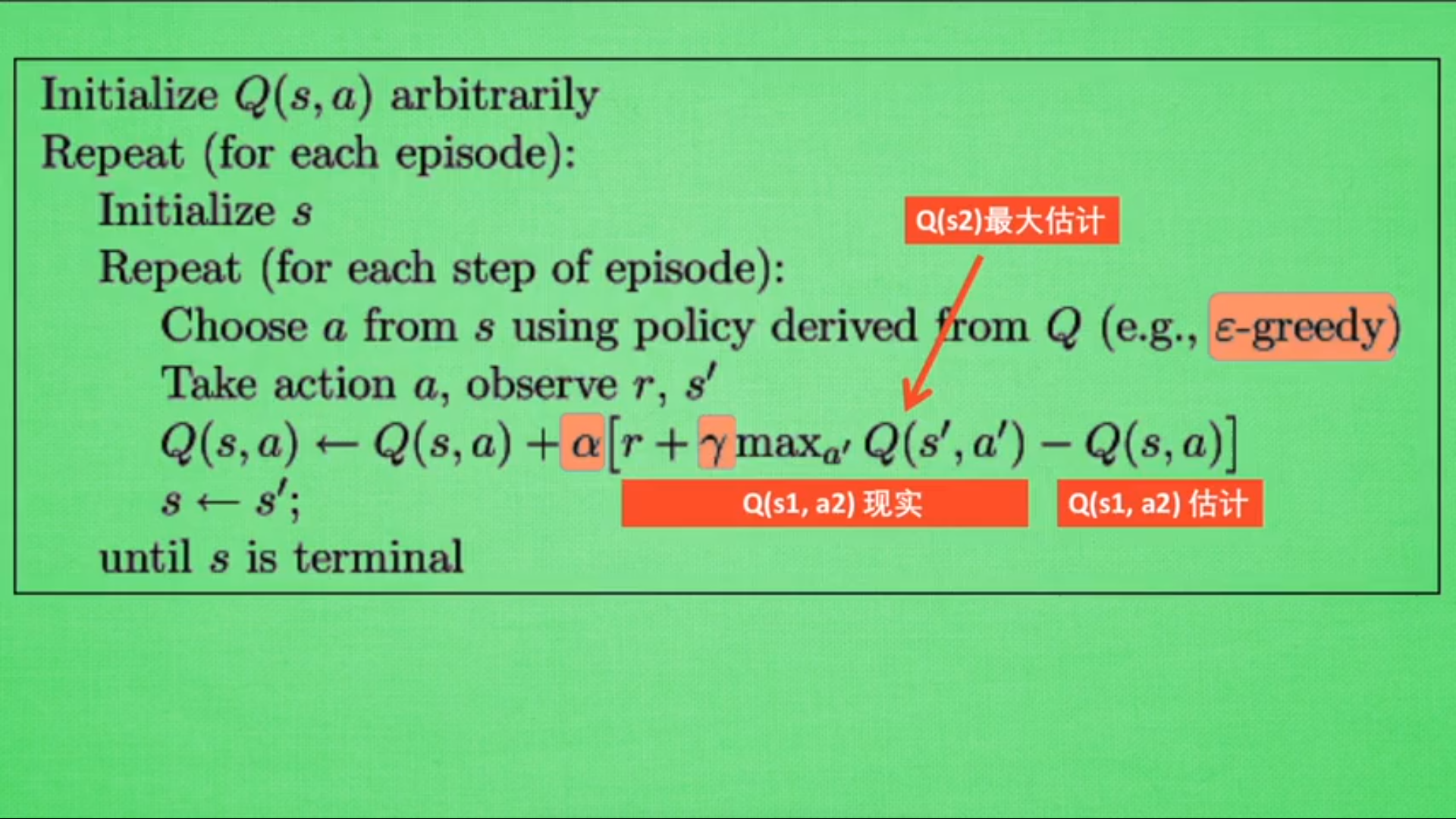
1. 在线学习（On-Policy）：本人在场，而且必须是本人边玩边学习

离线学习（Off-Policy）：可以选择自己玩，也可以选择看着别人玩，通过看着别人玩来学习别人的行为准则，同样是从过往经历中学习，但这些经历没必要是自己的

1. Q-Learning (Off-line)



**注意！**虽然用了maxQ(s2)来估计下一个s2状态，但还没有在状态s2作出任何的行为，s2的决策部分要等到更新完了以后再重新另外执行这一过程



ϵ - greedy是用在决策上的一种策略，如ϵ=0.9时，说明90%的情况按Q表的最优值来选择行为，10%的时间使用随机选择行为；

α是学习效率，来决定这一次误差有多少要被学习，α<1

γ是对未来奖励的衰减值