**概念：**强化学习是与监督学习和无监督学习并列的一种机器学习模式。强化学习既不同于监督学习——对特定情境设置标签，通过学习对某一情境对应的标签进行预测；也不同于无监督学习——找出数据间的关联与隐性结构；是一种自监督学习方式。其过程简单来说就是：智能体(Agent)通过不断地尝试并更新策略(Policy)，找出最大化的回报值(return)。

**研究问题：**如何找出环境的状态(state)与动作的映射，如何采取动作以获取最大化的奖励（策略）。即如何找出策略，如何使用策略使得回报值最大。

**特点：**闭环（采取的动作会影响后来的输入）；不直接采取行动（没有直接的指导性信息，需要通过尝试找出能获取最大化奖励的动作）；动作产生影响的时间延续性（每一步的动作不仅仅影响当前的奖励，还会影响之后每一步的奖励）。

强化学习更注重交互作用中的目标导向性行为（感觉、动作、目标）。强化学习采用一种交互性目标寻求智能体，它有明确的目标，能对环境的各个方面进行感知，能选择行动来对环境造成影响。尽管环境具有不确定性，但智能体仍需进行具体操作，使得每个重要的子问题都能被独立地进行研究。

强化学习最接近于人类与其他动物的学习方式，许多强化学习的核心算法都是受到生物学习系统的启发。

强化学习是人工智能回归简单通用原则的重要组成，致力于找寻人工智能领域的一般方法。

**强化学习要素：**策略(policy)、奖励信号(reword signal)、值函数(value function)、环境模型(model of environment)。

策略定义为智能体在给定时间内的行为准则，表现为环境中可感知的状态到应采取动作的映射。

奖励信号定义为强化学习的目标，奖励信号是智能体行为好与坏的判断标准，表现为一种即时的奖励，也是更改策略的准则。

值函数定义为一种更长期的奖励，使智能体不单单关注即时奖励，而是更多地关注长期的奖励，以达到奖励最大化的目标。

环境模型定义为模拟环境行为的方法，对环境进行建模，在模型中进行学习，对于给定的状态和动作，预测下一时刻的状态和回报，并找出收益最大的动作。

**不采用进化算法的原因：**强化学习问题的解决方法不仅仅是围绕价值估算函数的，也可以使用进化算法直接在策略空间搜索最优策略，对于策略空间较小或者易于求出策略空间的情况下，进化算法是有效的。另外当环境状态无法准确感知的情况下，进化算法具有优势。但进化算法不能实现在智能体与环境交互的过程中进行学习，即忽略了智能体在一个生命周期内经历了哪些状态或选择了哪些行为，所以进化算法在某些情况下获取的信息可能是具有误导性的。但会采用一种策略梯度方法。

**强化学习与优化问题的联系：**智能体的学习目标是得到最大化奖励值，但这并不意味着智能体一定能获得最大化的奖励。智能体只是在试图增加它所能获得的奖励，许多因素都会阻止它达到最大化的奖励。智能体可能具有创造力，使得它有时会找到一种“新的方法”，以从环境中获取更多地奖励，也可以理解为一种通向成功的新途径。但需要注意的是，这种新的“解决方案”可能带来意想不到的不良后果，即局部最优问题。这种问题被称为优化的不可预测性风险，这也是强化学习中必须注意的问题。

**井字游戏：**在使用×的情况下，所有三个×能连在一起的情况胜率设置为1，所有三个○能连在一起或者井字被填满的情况胜率设置为0，其余情况初始值设置为0.5。当游戏进行了许多回合，游戏中每一个可行动的状态，我们在表中查找每个行动后的状态，可以选择贪婪的选择使胜率最大的行动，也可以尝试其他行动方式来作为一种探索。不论作何选择，选择前后需备份两个状态的价值，如下表达式，表示更新前的状态，表示更新后的状态。更新后的估计价值为，实际价值为，则更新后的估计值可以表示为：

其中是步长参数，表示学习效率。

**强化学习历史：**分为三条线——动态规划与最优控制问题（贝尔曼方程、马尔可夫决策过程）；试错学习（多臂老虎机）；时序差分学习（Q Learning）

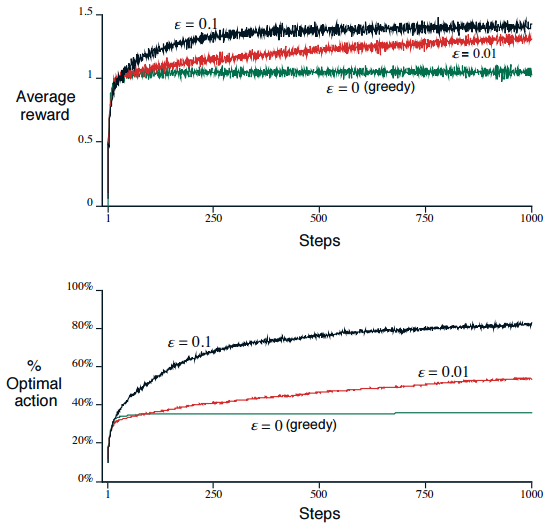
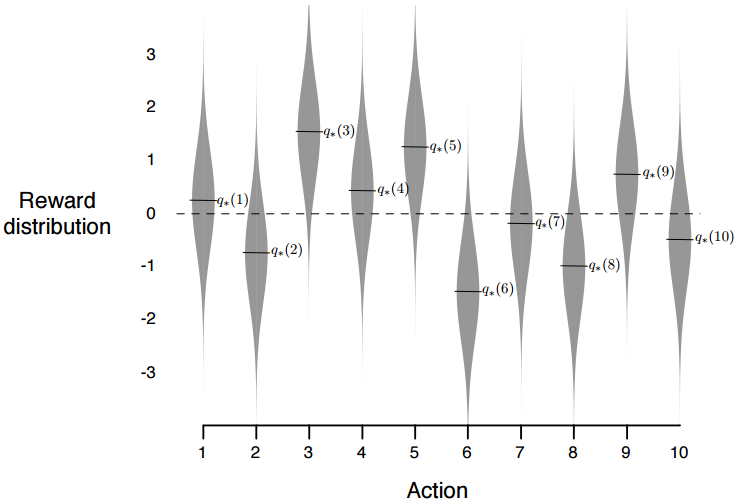
**多臂老虎机问题：**

不同于监督学习的指导性反馈，强化学习采用的是一种评价性反馈。这种反馈完全依赖于所采取的行动，即通过反馈来评估所采取的行动，而不是从反馈中直接通过正确的行动给所采取行动做出对或错的指导。多臂老虎机问题是指在一台老虎机上，有很多个杠杆，每个杠杆拉下后对应的奖励都不同，但它们都对应有一个固定的奖励概率分布，通过不断地拉杠杆，就可以找到能获得最大奖励的那根杠杆。下式为动作真实价值的表达式，其中为期望，为t时刻（第t次）的动作，为t时刻的奖励（实际价值）：

如果每次动作的价值已知，你就能一直选择价值最大的动作，那么可以得到动作的估计价值。在任意时刻，都有一个动作的估计价值是最大的，这样的动作称之为贪婪行为，选择一个贪婪行为的操作叫做Exploitation，而选择一个非贪婪行为的操作叫做探索。探索是指以相同的概率去选择所有动作中的一个。探索虽然短期回报不及Exploitation，但长期回报更高，所以如何权衡与选择（平衡）两种操作方式是该问题的关键。

**平衡方式：**

1.：以的概率选择探索，1-的概率选择利用。其中的选择十分关键，约小表示探索的越少，当，智能体不进行探索，那么当他尝试过每个动作，就会得到该动作的反馈（本次的真实价值），那么智能体会选择奖励最大的动作继续做下去。然而，动作的真实价值往往不是固定的，可能会有一个浮动的范围，所以探索是必要的。下图展示了对一个10臂的老虎机，其每个臂的奖励遵循方差为1的正态分布，分别令，，所产生的平均奖励值及最优臂选择率的曲线：



不难看出，此时使用贪婪行为的结果是最差的，而对比和的情况可以发现，探索越多，越能更早地找到使奖励最大化的臂。但必须注意的是，即使找到了这个臂，仍有的概率不会选择它，这也是方法的弊端所在。并且，方法适用于每个动作奖励具有方差的情况，当方差为0，直接使用贪婪行为是更好地做法。

**平均价值计算：**对于单独一个动作，其平均价值即每次选择该动作的回报之和除以选择的次数，如下式，其中是第次选择该动作的奖励：

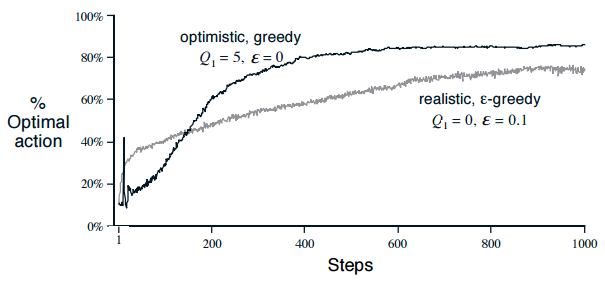
那么对于下一次选择该动作，其平均价值的更新表达式即为：

其中为步长，称之为第n步的误差值。

**非平稳问题：**对于一个实际问题，一个“老虎机”往往是非平稳的（没有固定的价值分布概率），处理这样的问题通常的做法是使用一个恒定的步长系数，在0-1之间取值，则之前的表达式可以转化为这样：

由于，所以上式被称为加权平均。不难发现，随着i的增加，所占的权重会越来越大，也就是说平均价值会更关注更接近n的奖励。

**乐观初始值：**之前的方法在一定程度上依赖一个初始值，对于这个初始值的给定，往往会选择一个较为乐观的态度，当，那么学习会简单而正常地进行，但如果选择一个较大的初始值，如，那么在一开始的学习中，得到的很难超过5，这并不是坏事，它激励了智能体多去尝试探索，当然导致的结果是，前期由于探索较多，得到的奖励会更少，但智能体会更早的找到最优动作，以至于能更早地最大化奖励。如下图：



**UCB算法：**对于方法，其探索过程只是单纯地在每个非贪婪行为中做无差别的选择，而UCB算法给出了一种动作选择方式。如下式，表示a动作在t时刻的平均价值，后一项称之为UCB值，其中表示t时刻选择a动作的次数，c是>0的系数。那么根据函数性质，一个动作如果从来没被选择过，那么他的优先级是最高的，并且动作被选择的次数越少，他累计的UCB值就越高，从而增加了他被选择的概率。UCB算法在多臂老虎机问题中效果比算法要好，但在强化学习的其他问题中几乎不会被用到。

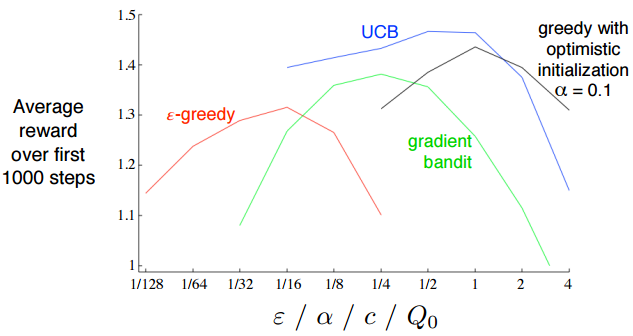
**梯度老虎机算法：**该算法提供了一种新的动作选择思路，它为提供了一个动作的偏好作为选择依据，偏好越大，动作选择的概率就越大。该算法公式如下：

偏好的更新公式如下：

其中是步长参数，是t时刻的平均价值，这相当于一条用于比较奖励的基准线，当奖励高于标准线，则偏好值提升，低于标准线则降低。由上式可得，在提高一个动作的选择概率的同时，降低了其他动作的选择概率，即偏好的定义。

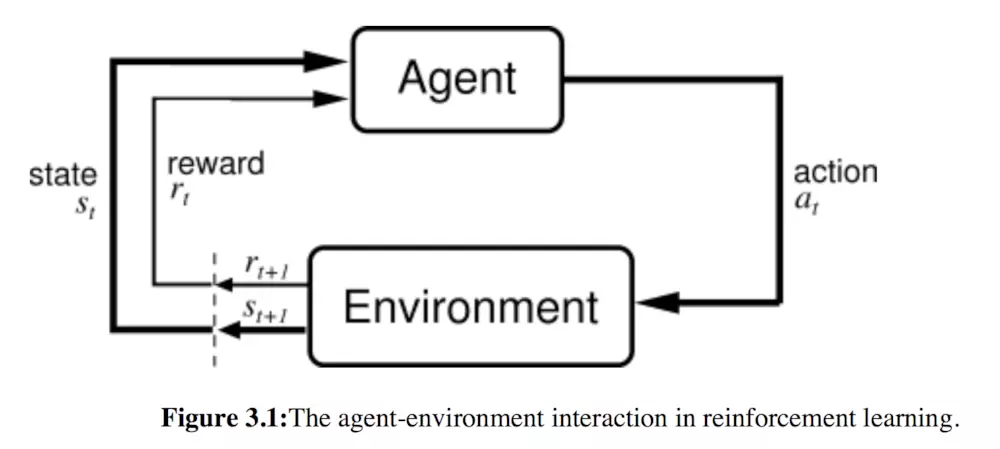
**关联检索：**到目前为止，我们考虑的情况都是基于环境不发生改变的前提下的，对单一场景做出的最优的动作选择。而在实际中，环境是无时无刻不在变化的，那么当前的理论就难以满足这样的需求了。所以，我们要让智能体学会在不同的环境中选择不同的动作。状态的引入是必要的，不仅仅当前时刻的状态会影响当前时刻动作的选择，每一时刻的动作也会对下一时刻的状态造成影响。只有将智能体的动作与状态关联在一起，才是一个完整的强化学习问题。

对于多臂老虎机问题，以上方法的性能比较如图，可以说UCB方法最好。



**有限马尔可夫决策过程：**

强化学习问题描述了一个从交互中找寻目标的直接框架，其中学习者与决策者称为智能体，不包含智能体本身，与之交互的其他东西称之为环境。在智能体与环境交互的过程中，智能体通过不断地选择动作对环境造成影响，而环境在下一时刻会反馈给智能体两个量：状态和奖励。状态一定会有，也可能和上一时刻状态相同，但奖励未必会有，因为奖励不是每一步都会产生的，直到一个特定的状态产生，才会有奖励或惩罚。强化学习的目标就是通过智能体选择动作，找出最大化的总奖励值对应的策略。



具体来说，智能体和环境在每个时刻进行交互，t=1,2，…… 对每个时刻t，智能体会接收到环境的当前状态，并在该状态的基础上选择动作。每一个动作之后，智能体会收到一个反馈奖励值，并且进入一个新的状态。在每一时刻，智能体将实现一个从状态到每个可能动作的概率映射，这个映射称之为决策，表示为。强化学习方法就是让智能体从每次动作中累计经验，从而学会一种能使整个过程中累积的总奖励值最大化的决策方式。

动作和状态是抽象化的概念，并不单单指具体的动作和状态。总的来说，动作可以是我们想让智能体学习并做出的任何决定，而状态可以是任何可能对决策造成影响的东西。

**智能体与环境的界限：**智能体和环境的界限在于，任何不能被智能体随意改变的部分都属于环境，所以即使奖励和状态都能轻易地被智能体感知，但由于他们不受控于智能体，所以也属于环境的范畴。智能体本身也不一定是指代某个物体本身，如一个复杂的机器人，可能有多个智能体在同时进行决策，而每个智能体都有自己的边界，如A负责决定如何移动，B负责决定几点充电。在实践中一旦选择了特定的状态、行为和奖励，那么智能体的决策任务和界限也就随之确定了。

**目标和奖励：**为智能体的动作设置奖励是为了让其更好地实现目标，奖励信号的设置原则应该是让智能体实现最终的目标，而不是通过奖励给智能体设置子任务。如围棋的最终目标是比对手的活棋数更多，即赢得棋局，但如果设置子任务为吃掉对面的棋子，智能体就可能不惜输掉棋局来吃掉更多对面的棋子。

**回报：**智能体的目标是找到一个最大的累计奖励，这个奖励的总和定义为回报，其表达式为：

其中是最后一个时刻。这个定义对于那些能够自然分成一个个序列的任务很有意义，比如玩一局游戏或者走一个迷宫等等，每一次都是一个序列。那些有一个中止状态的任务被称为周期性任务(episodic tasks)。另外有一些没法分开成一个个序列的任务，比如一个机器人整个生命过程，也就是可以被认为是是无穷大的任务，叫做持续性任务(continuing tasks)。对于这种任务直接把所有的奖励相加会得到一个无穷大值，也就没法实现回报最大化，因此期望值被定义为非持续的形式，在t+1以后每个奖励都多加一个[0,1]之间的系数，即：

称之为衰减率，当，智能体只关注当前的即时回报，而不关注长期的回报，当趋向于1，那么智能体就会更有远见。

对于一个周期性任务，我们也可以人为地把它设置成持续性任务，只需要将时刻后的奖励值设置为0即可。这样，就能将周期性任务和持续性任务的表达式统一为：

其中可以为无穷大，也可以等于1，用来分别表示持续性任务和周期性任务。

**马尔可夫属性：**无后效性。一个动作的结果不依赖于以前的任何动作和状态，仅仅取决于当前状态。两方面：需要求解的内容与之前的状态无关；仅仅用现有信息就能求出。如位置和速度就满足，且二者缺一不可，多一也不可。将这个性质进行拓展，可以得到下式：

即下一时刻的状态与回报只与当前时刻的状态和动作有关。对于一些不满足马尔可夫性质的状态信息，比如机器人在行动时的损耗，可以将其忽略，近似成马尔可夫性质，这样做最大的好处是可以加快智能体的学习速度。

**马尔可夫决策过程：**符合马尔可夫性的强化学习任务叫做马尔科夫决策过程(MDP)。一个马尔可夫决策过程由状态集，动作集，状态转移概率和奖励函数构成。

状态集𝑆包含了所有可能出现的状态。

动作集𝐴包含了所有可能出现的动作。

转移概率𝑃是指在状态𝑠下选择动作𝑎，下一时刻状态为𝑠′的概率。

奖励函数𝑅是指选择动作𝑎的情况下，从状态𝑠到状态𝑠′得到的奖励的期望。

在状态下执行动作，得到的奖励函数为：

下一时刻的可能状态的状态转移概率为：

下一时刻奖励函数为：

**策略：**MDP的关键问题在于寻找一个策略：在状态𝑠下执行动作𝑎，组成一个动作函数𝜋(𝑠,𝑎)。我们的目的在于选择一个策略𝜋可以最大化累积奖励𝐺𝑡。

**值函数：**值函数的定义是为了体现智能体在状态下实施策略的长期影响，即对最终总回报值的影响。其定义式如下：

称这个值函数为状态值函数，表示在采取策略时，状态的回报的期望。举例说明就是在下棋的时候，一副棋面在采取策略时，赢棋的可能性大小。

在状态选择动作的情况下，实施策略的值函数定义式为：

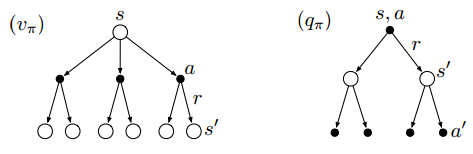
称这个值函数为动作值函数，表示在采取策略时，在状态下选择动作的回报的期望。举例说明就是在下棋的时候，一副棋面在采取一个动作的时候，赢棋的可能性大小。

值函数可以通过经验进行估计，智能体的每一个策略都有一个平均的真实反馈值，随着趋于无穷大，这个平均值也会逐渐收敛至或。所以我们可以用抽样的方法估计值函数，称做蒙特卡罗方法。

可以写成递归的形式，叫做贝尔曼方程(Bellman equation)。如下式：

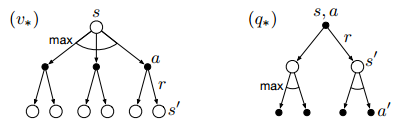
这个式子表明了状态价值与后继状态价值的关系。当我们知道了的值函数，就能推导出的值函数，也就提供了一种通过迭代计算来求解状态值函数的途径。

同理动作值函数的贝尔曼方程如上式，它们都可以通过下面的回溯图来解释。



**最优策略和最优值函数：**如果一个策略的所有状态的值函数都大于策略的所有状态的值函数，那么就说策略优于策略。这样就会有一个最优策略，记做。那么其最优状态值函数即，最优动作值函数即，其表达式为：

写成贝尔曼方程的形式：



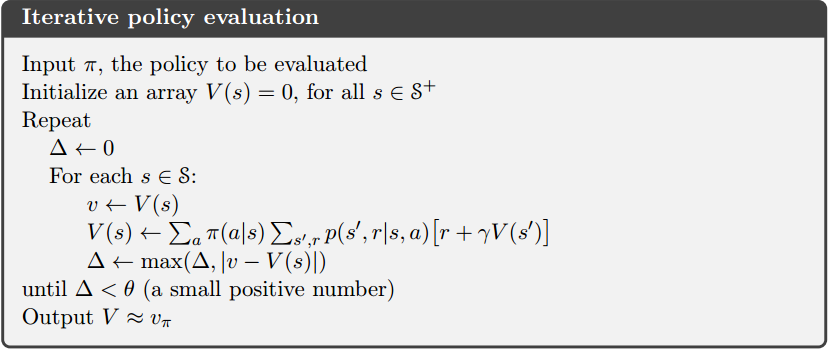
**最优问题和近似解：**虽然我们已经定义出了最优值函数和最优策略，而且理论上也可以直接计算出来。但是通常情况下我们没法得到这么多的计算资源。与此同时内存溢出也是一个很大的问题，因为很多问题的状态数量太多超过存储范围。对于这些情况我们就不能够使用直接存储每个状态的值函数，而是必须使用一种更精简的参数型函数表示的方法。强化学习的框架迫使我们进行近似求解，而且这个框架同时也很容易进行近似，比如对于很多小概率出现的状态，选择最优解和次优解区别不大。而且强化学习的在线学习的特性让其能更多关注出现次数多的状态，这是一个强化学习区分其它近似求解MDP的关键特点。

**动态规划：**

动态规划的核心思想是将一个复杂的问题分解成多个子问题，并通过求解子问题来解决原问题。对于一个强化学习问题，我们可以认为它是一个多阶段的决策问题，每个时间步长（状态）都可以视为一个阶段，我们要做的就是在每一阶段选择一个策略，使得整个过程达到一个最好的效果，即获取最大化回报。

在DP中，为了评价一个策略的好坏，对任意的策略计算其状态值函数的过程又称为策略评估。这个过程的表达式如状态值函数的贝尔曼方程：

为了求出，对于一个策略，我们需要知道之后每一个状态的值函数、奖励以及状态转移概率，这种方法被称为全备份。在迭代实现中，我们可以采用更新的方式，用新的状态值函数、奖励及状态转移概率去替换旧的来实现，把这个变量表示为，这个过程称之为迭代策略评估，如下图所示。



首先选择一个需要评价的策略，初始化并设置一个误差变量，对每一个状态进行迭代。每次迭代将上一次求出的存入中间变量，并求出此时的状态值函数存入，其差值存入误差变量。多次迭代后，当足够小，此时的即为该策略的状态值函数的近似值。

当我们已经求得了状态值函数，我们会思考对某些状态是否应该改变策略选择其他的动作，这是需要动作值函数对策略进行评估：

该式的重点在于是否大于，如果是，那么说明在状态选择动作得到的结果要比根据策略去选择动作的效果要好。这也被称为策略改进理论，对所有的状态，如果与满足下式：

那么就说要优于。同时可以推出：

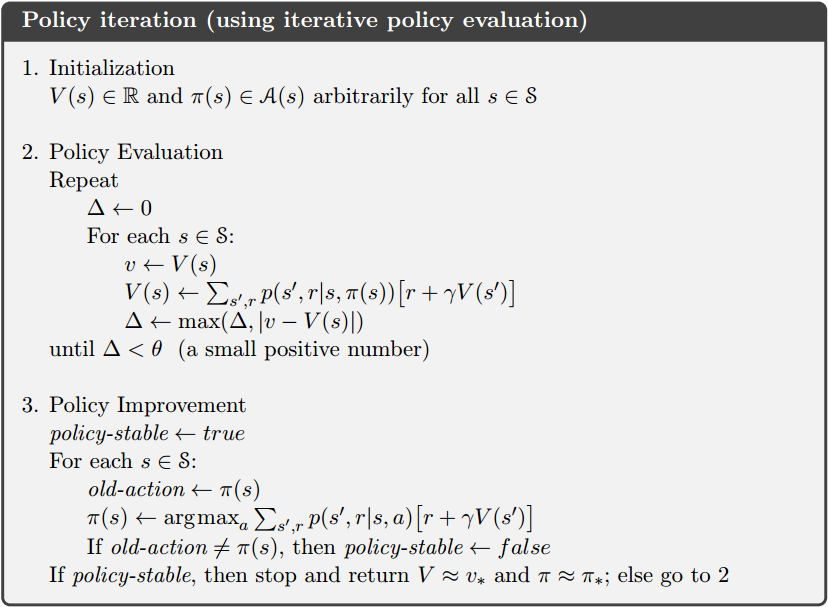
如通过贪婪策略进行策略改进即下式：

即最优贝尔曼方程，使用贪心得到最优策略的过程叫做策略改进。

**策略迭代：**一旦一个策略被使用改进得到了一个新的策略那么就可以计算出新的值函数。根据新的值函数又可以进行改进得到更好的策略，如此循环往复直到达到最优策略。

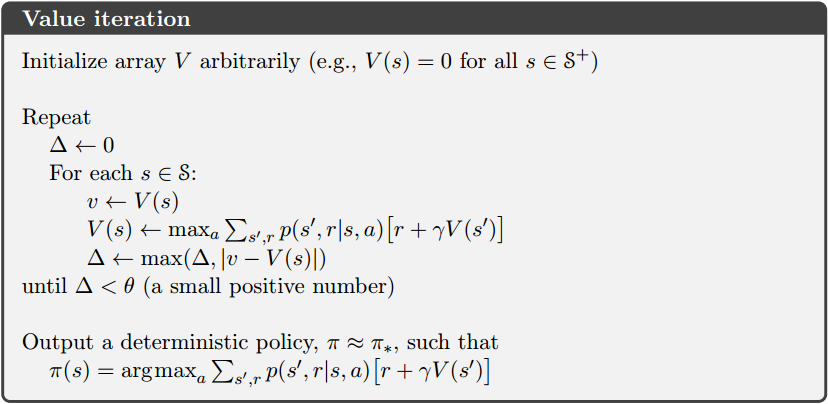


因为一个有限状态MDP只有有限数量的策略，因此这个过程一定会在有限步之内得到最优策略和最优值函数。这个方式叫做策略迭代。具体的算法如下表所示：



即在策略评估的基础上，加上策略改进的过程，对策略和状态值函数进行交替改进直至收敛，最终得到最优策略和最优值函数。

**值迭代：**由于策略迭代的缺点：每次策略评估都要进行多次迭代直至收敛后再进行策略改进，提出了值迭代算法，其过程如下：



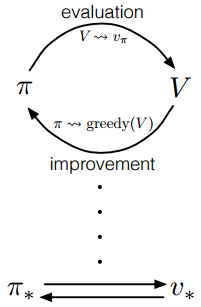
即在每次迭代产生的同时，进行策略改进，取最优动作作为策略开始下一次迭代，直至收敛。二者的区别在于，策略迭代中策略更新是将每个状态下各个行为的期望值总和替代上一次迭代中的，而值迭代是用某个行为的最大动作值函数去替代上一次迭代中的。

**异步动态规划：**目前为止讲的几个DP算法的一个主要的缺陷是这些算法都需要遍历所有的状态空间。如果状态空间很大，那么一次遍历中策略就可能在很长时间都得不到及时更新。异步DP算法主要是使用非系统性遍历所有状态空间的in-place DP算法。这些算法不关心每个状态值函数的更新顺序，某些状态的值可能比其它状态的值更新快好几倍。但是异步算法依然需要更新所有状态的值函数之后才能够达成正确的收敛，也就是不能忽略某些状态值函数的计算。但是异步DP算法能够灵活选择需要进行更新的状态。

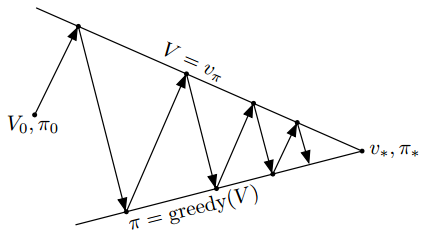
可以将一些不同的更新方式灵活的用在不需要进行遍历操作的DP算法中。但是避免了遍历并不意味着减少了计算量 ，它只是意味着我们不需要在策略有改进之前陷入长时间的状态遍历中，也就是选择需要更新的状态来加速算法的运算过程。因为某些状态可能不需要像其它状态那样更新那么多次。

异步算法同样能够使得实时交互与计算过程之间的混合更加容易。我们可以在一个智能体实时经历一个MDP过程的同事运行一个迭代DP算法，智能体的经历可以用来决定算法需要更新哪些状态的值函数。同时DP算法中产生的最新值函数和策略也能够指引智能体进行决策。这样能够重点关注哪些和智能体最相关的那部分状态，这种重点关注是强化学习中不断出现的主题。

**广义策略迭代：**每个算法都含有独特的策略以及值函数，策略可以一直利用值函数进行改进，而值函数也被更新得符合当前策略，如下图，如果两个过程都稳定了，那么值函数和策略一定都达到了最优。因为只要当它们都互相一致而且满足了贝尔曼最优等式之后才有可能稳定，也就是保证了最优。



GPI中评估和改进的过程可以被看成是一直合作与竞争。竞争是说它们两个过程互相拉向相反的方向，因为让当前策略基于当前值函数最优一般意味着让值函数与新策略不一致。而让值函数与策略一致一般又会导致策略在更新完的值函数上不是最优。但是长期来看这两个过程互相作用最终达到了最优策略和最优值函数。这两个过程的每一个都是向着达成最优的目标前进，尽管他们前进的路线可能都不是直线。

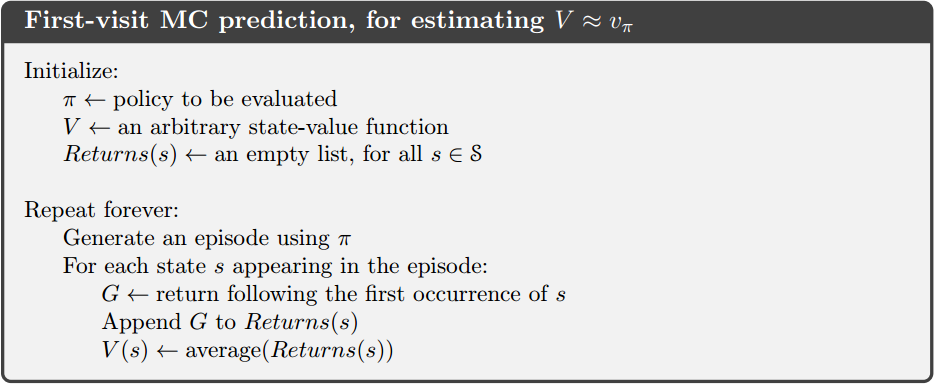


**蒙特卡罗方法：**

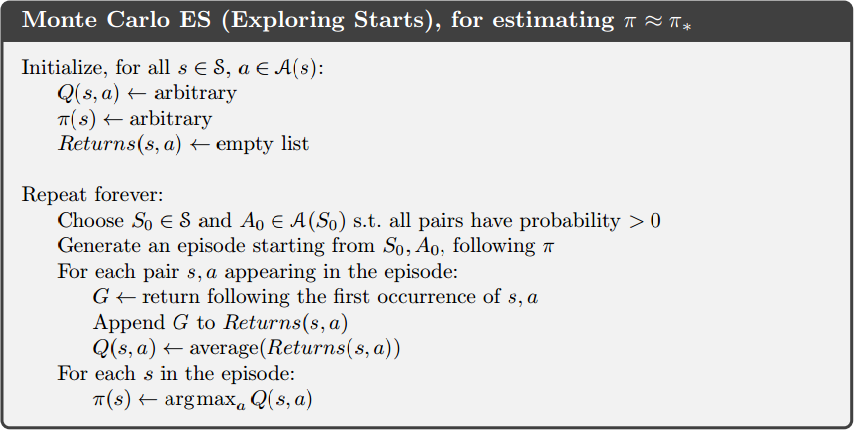
对于环境无法建模的情况，使用蒙特卡罗法。蒙特卡罗方法又叫统计模拟方法，它使用随机数（或伪随机数）来解决计算的问题，是一类重要的数值计算方法。该方法的名字来源于世界著名的赌城蒙特卡罗，而蒙特卡罗方法正是以概率为基础的方法。

对于强化学习问题，蒙特卡罗方法可以理解为：对一个阶段性任务，通过多次模拟，得到许多含有状态、策略和回报的样本。通过这些样本的平均值来估计在状态下，遵循策略得到的期望回报，即。

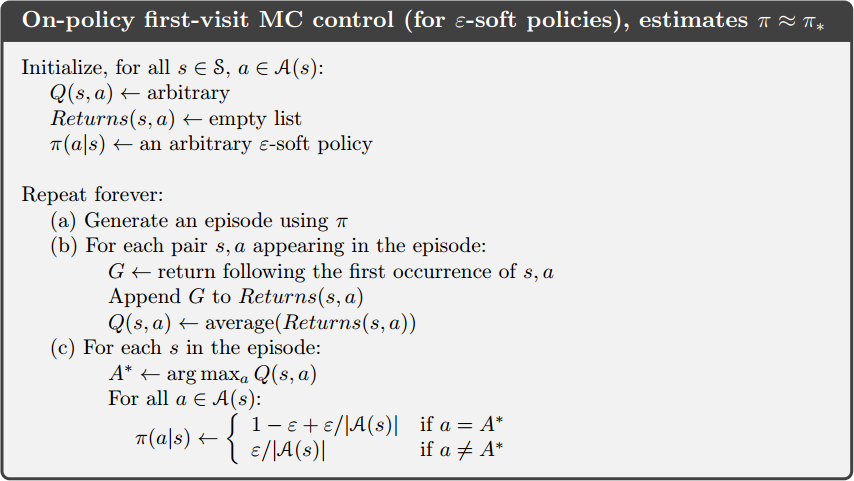
**首次访问MC算法：**对于每次迭代，当系统到达一个状态称之为一次访问。显然，状态在迭代中可能被多次访问。如果对每次迭代时第一次访问状态后得到的所有回报做平均，称之为First-visit MC算法。而对每次迭代时每一次访问状态后得到的所有回报做平均则称之为Every-visit MC算法。首次访问MC算法的伪代码如下：



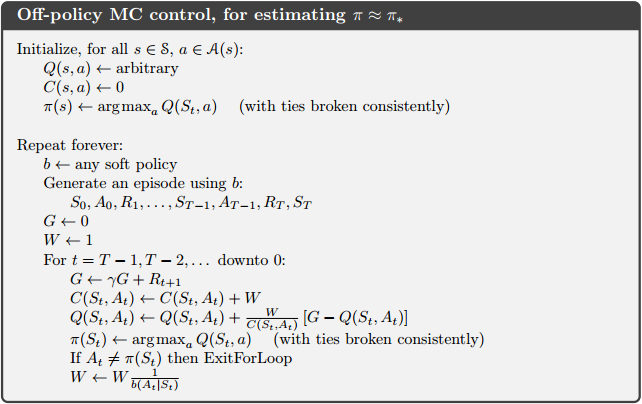
**MCES算法：**对模型未知的环境，仅仅使用状态值函数是不够的，因此引入动作值函数。并且，有了动作值函数的估计值，就可以做策略提升了。对于一些状态动作组有可能会永远用不到的情况，使用探索初始的方式，即每个状态对应的动作都有大于0的可能性被选中。并基于这种状态和动作，遵循策略进行迭代：将第一次遇到和后的回报存入回报集，那么其动作值函数即的均值，在对状态的每个episode结束时，使用最大的策略。如下伪代码：



**在线MC算法：**对于一些状态动作组有可能会永远用不到的情况，使用软决策：即所有的动作都可能被执行，如。那么每个非贪婪行为都有的概率被选中，而贪婪行为被选中的概率为。其更新过程如下图，首先随便使用一个策略，执行MCES的过程，但在最后更新时，将最大的动作记为，并对使用。其伪代码如下：



**离线MC算法：**在线策略的方法是使用相同的策略进行值函数的估计和动作的选择，离线方法则是将二者分开为行为策略和目标策略，这样可以把目标策略设置为固定的，如，动作策略则是任何可能的行为。



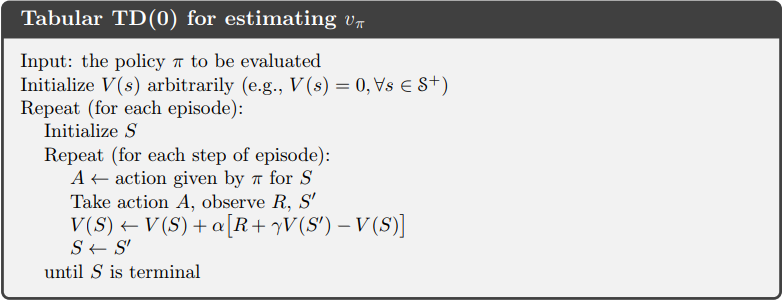
**时间差分方法：**

时间差分方法是蒙特卡罗思想和动态规划思想的结合，总结来说：

|  |  |
| --- | --- |
| 方法 | 特点 |
| 动态规划DP  蒙特卡罗MC  时间差分TD | 需要环境模型，即状态转移概率  状态值函数的估计是自举的(bootstrapping)，即当前状态值函数的更新依赖于已知的其他状态值函数  可以从经验中学习不需要环境模型  状态值函数的估计是相互独立的  只能用于episode tasks  结合了前两种的特点  不需要环境模型  不局限于episode task，可以用于连续的任务 |

最简单的TD值函数计算方式（TD(0)）如下：

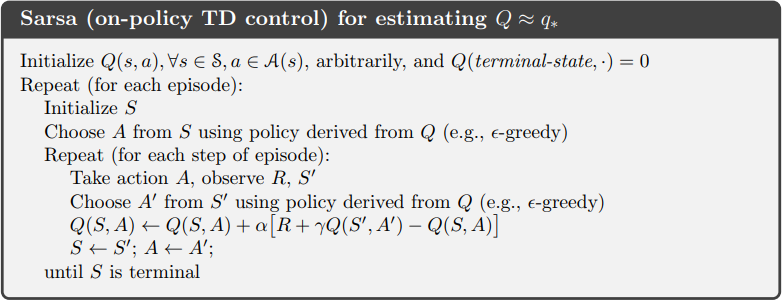
简单来说，TD是利用真实的立即回报和下一状态的状态值函数来更新的一种方法。TD(0)完整更新步骤的伪代码如下：



**TD error：**在每一时刻，因为估计产生的误差被称之为TD error。即：

**Sarsa：**Sarsa是一种On-Policy的TD方法，其动作值函数的更新表达式如下：

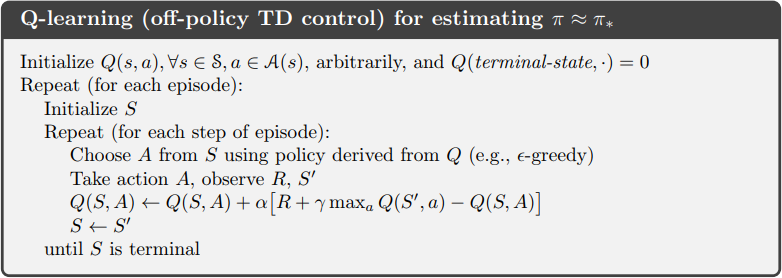
Sarsa表达的是(,,,,)这五组元素之间的关系，其更新步骤大致是：对每一个episode，对状态，用一个策略（如）选择动作，得到对应的和下个状态，然后对，使用相同策略选择动作，更新一直到一个episode结束。其伪代码如下：



**Q-Learning：Q-Learning**是一种Off-Policy的TD方法，其动作值函数的更新表达式如下：

不难看出，这里的不同在于对于下一状态，更新值时，直接使用了最大的，相当于采用了值最大的动作，这个策略与选择时的策略无关。

实际上这里使用了表格的方法，每个状态对应每个动作都有一个值，初始为0。把第一次选择动作得到的值填入当前的表，在选择下一个动作时，选择表中值最大的动作，然后用上面的更新表达式更新值，其更新的伪代码如下：



**强化学习分类：**

**环境划分：**

**Model-Free RL（不理解环境）：**Q Learning, Sarsa, Policy Gradients

直接对现实世界试验得到反馈进行学习，需要按部就班得到反馈。

**Model-Based RL（理解环境）：**Q Learning, Sarsa, Policy Gradients

先对现实世界建模，在模型中试验得到反馈进行学习，可以通过想象预判断会发生的情况，根据想象选择最好的情况，并根据情况采取相应策略。

**目标划分：**

**Policy-Based RL（基于概率）：**Policy Gradients

通过感官分析所处的环境，分析下一步采取各种行动的概率，通过概率选择，每一种动作都有可能被选中，只是概率不同。可分析连续动作。

**Value-Based RL（基于价值）：**Q Learning, Sarsa

通过感官分析所处的环境，分析下一步采取各种行动的价值，直接选择价值最高的行动。无法分析连续动作。

**Actor-Critic：**

结合概率与价值，通过概率做出动作，再对做出的动作得到价值。

**更新机制划分：**

**Monte-Carlo update（回合更新）：**基础版Policy Gradients, Monte-Carlo Learning

游戏开始 -> 等待游戏结束 -> 更新行为准则。

**Temporal-Difference update（单步更新）：**Q Learning, Sarsa, 升级版Policy Gradients

游戏中的每一步都可以更新，更有效率。

**在线与离线：**

**On-Policy（在线学习）：**Sarsa, Sarsa(λ)

必须本人在场，本人在边玩边学习。

**Off-Policy（离线学习）：**Q Learning, Deep Q Network

不仅本人玩，还能看着别人玩，并且边看边学习，或者先玩，玩完再做离线学习。