增强学习（二）----- 马尔可夫决策过程MDP - 金淑林

1. 马尔可夫模型的几类子模型

大家应该还记得马尔科夫链(Markov Chain)，了解机器学习的也都知道隐马尔可夫模型(Hidden Markov Model，HMM)。它们具有的一个共同性质就是马尔可夫性(无后效性)，也就是指系统的下个状态只与当前状态信息有关，而与更早之前的状态无关。

马尔可夫决策过程(Markov Decision Process, MDP)也具有马尔可夫性，与上面不同的是MDP考虑了动作，即系统下个状态不仅和当前的状态有关，也和当前采取的动作有关。还是举下棋的例子，当我们在某个局面（状态s）走了一步(动作a)，这时对手的选择（导致下个状态s’）我们是不能确定的，但是他的选择只和s和a有关，而不用考虑更早之前的状态和动作，即s’是根据s和a随机生成的。

我们用一个二维表格表示一下，各种马尔可夫子模型的关系就很清楚了：

	不考虑动作	考虑动作
状态完全可见	马尔科夫链(MC)	马尔可夫决策过程(MDP)
状态不完全可见	隐马尔可夫模型(HMM)	不完全可观察马尔可夫决策过程(POMDP)

2. 马尔可夫决策过程

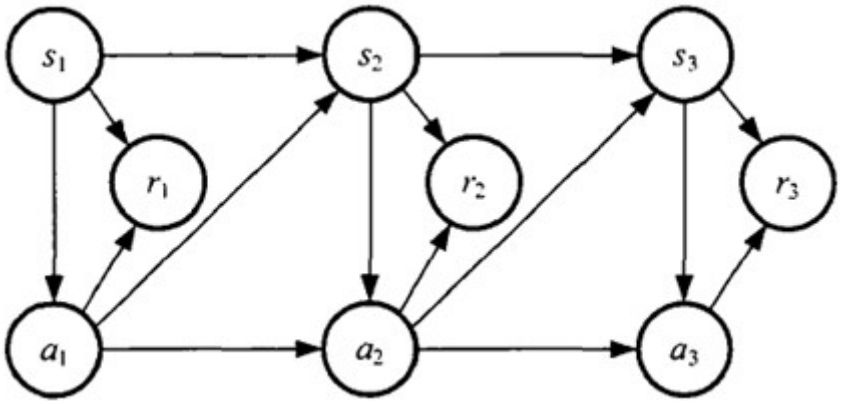
一个马尔可夫决策过程由一个四元组构成M = (S, A, P_{sa}, R) [注1]

- S: 表示状态集(states)，有s∈S，s_i表示第i步的状态。
- A:表示一组动作(actions)，有a∈A，a_i表示第i步的动作。
- P_{sa}: 表示状态转移概率。P_{sa} 表示的是在当前s ∈ S状态下，经过a ∈ A作用后，会转移到的其他状态的概率分布情况。比如，在状态s下执行动作a，转移到s'的概率可以表示为p(s'|s,a)。
- R: S×A↦ℝ，R是回报函数(reward function)。有些回报函数状态S的函数，可以简化为R: S↦ℝ。如果一组(s,a)转移到了下个状态s'，那么回报函数可记为r(s'|s, a)。如果(s,a)对应的下个状态s'是唯一的，那么回报函数也可以记为r(s,a)。

MDP 的动态过程如下：某个智能体(agent)的初始状态为s_0，然后从 A 中挑选一个动作a_0执行，执行后，agent 按P_{sa}概率随机转移到了下一个s_1状态，s_1∈ P_{s0a0}。然后再执行一个动作a_1，就转移到了s_2，接下来再执行a_2…，我们可以用下面的图表示状态转移的过程。

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

如果回报r是根据状态s和动作a得到的，则MDP还可以表示成下图：



3. 值函数(value function)

上篇我们提到增强学习学到的是一个从环境状态到动作的映射（即行为策略），记为策略π: S→A。而增强学习往往又具有延迟回报的特点: 如果在第n步输掉了棋，那么只有状态s_n和动作a_n获得了立即回报$r(s_n,a_n)=-1$，前面的所有状态立即回报均为0。所以对于之前的任意状态s和动作a，立即回报函数r(s,a)无法说明策略的好坏。因而需要定义值函数(value function，又叫效用函数)来表明当前状态下策略π的长期影响。

用$V^{\pi}(s)$表示策略π下，状态s的值函数。r_i表示未来第i步的立即回报，常见的值函数有以下三种：

a)

$$V^{\pi}(s)=E_{\pi}\left[\sum_{i=0}^{h} r_i \middle| s_0=s\right]$$

b)

$$V^{\pi}(s)=\lim_{h\to\infty} E_{\pi}\left[\frac{1}{h}\sum_{i=0}^{h} r_i \middle| s_0=s\right]$$

c)

$$V^{\pi}(s)=E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_i \middle| s_0=s\right]$$

其中：

a)是采用策略π的情况下未来有限h步的期望立即回报总和；

b)是采用策略π的情况下期望的平均回报；

c)是值函数最常见的形式，式中γ∈[0,1]称为折合因子，表明了未来的回报相对于当前回报的重要程度。特别的，γ=0时，相当于只考虑立即不考虑长期回报，γ=1时，将长期回报和立即回报看得同等重要。接下来我们只讨论第三种形式，

现在将值函数的第三种形式展开，其中r_i表示未来第i步回报，s'表示下一步状态，则有：

$$V^{\pi}(s) = E_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \middle| s_0 = s \right]$$
$$= E_{\pi}\left[r_0 + \gamma E\left[\gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \right] \middle| s_0 = s \right]$$
$$= E_{\pi}\left[r(s' \mid s, a) + \gamma V^{\pi}(s') \middle| s_0 = s \right]$$

给定策略π和初始状态s，则动作a=π(s)，下个时刻将以概率p(s'|s,a)转向下个状态s'，那么上式的期望可以拆开，可以重写为：

$$V^{\pi}(s) = \sum_{s' \in S} p(s' \mid s, a)\left[r(s' \mid s, a) + \gamma V^{\pi}(s') \right]$$

上面提到的值函数称为状态值函数(state value function)，需要注意的是，在$V^{\pi}(s)$中，π和初始状态s是我们给定的，而初始动作a是由策略π和状态s决定的，即a=π(s)。

定义动作值函数(action value functionQ函数)如下：

$$Q^{\pi}(s, a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i \middle| s_0 = s, a_0 = a \right]$$

给定当前状态s和当前动作a，在未来遵循策略π，那么系统将以概率p(s'|s,a)转向下个状态s'，上式可以重写为：

$$Q^{\pi}(s, a) = \sum_{s' \in S} p(s' \mid s, a)\left[r(s' \mid s, a) + \gamma V^{\pi}(s') \right]$$

在$Q^{\pi}(s,a)$中，不仅策略π和初始状态s是我们给定的，当前的动作a也是我们给定的，这是$Q^{\pi}(s,a)$和$V^{\pi}(a)$的主要区别。

知道值函数的概念后，一个MDP的最优策略可以由下式表示：

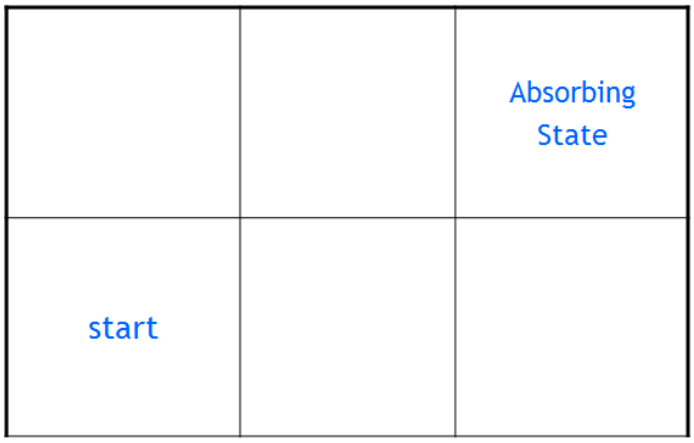
$$\pi^* = \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

即我们寻找的是在任意初始条件s下，能够最大化值函数的策略π*。

4. 值函数与Q函数计算的例子

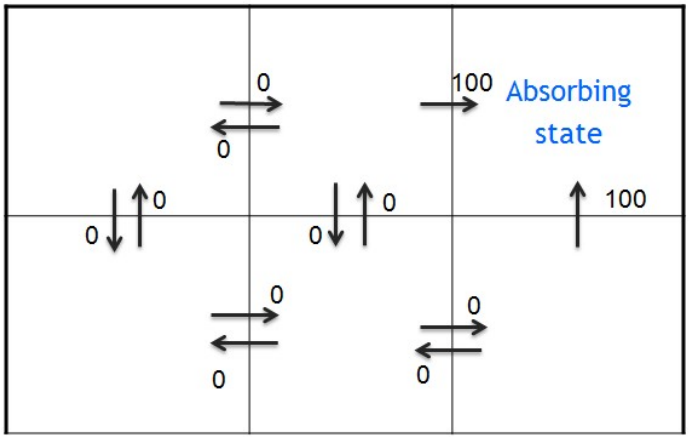
上面的概念可能描述得不够清晰，接下来我们实际计算一下，如图所示是一个格子世界，我们假设agent从左下角的start点出发，右上角为目标位置，称为吸收状态(Absorbing state)，对于进入吸收态的动作，我们给予立即回报100，对其他动作则给予0回报，折合因子γ的值我们选择0.9。

为了方便描述，记第i行，第j列的状态为s_{ij}, 在每个状态，有四种上下左右四种可选的动作，分别记为a_u, a_d, a_l, a_r。（up，down，left，right首字母），并认为状态按动作a选择的方向转移的概率为1。

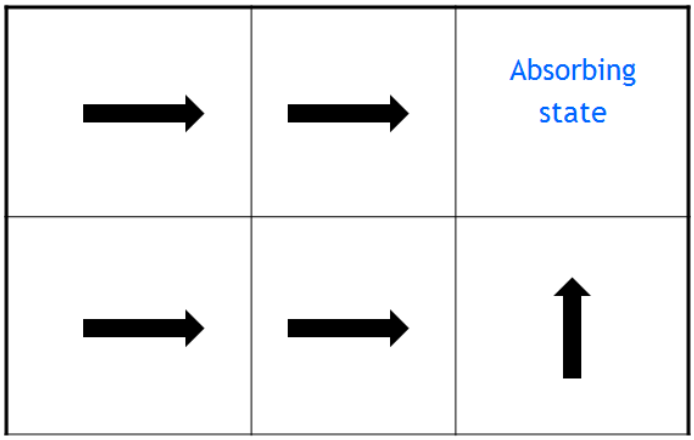


1.由于状态转移概率是1，每组(s,a)对应了唯一的s'。回报函数r(s'|s,a)可以简记为r(s,a)

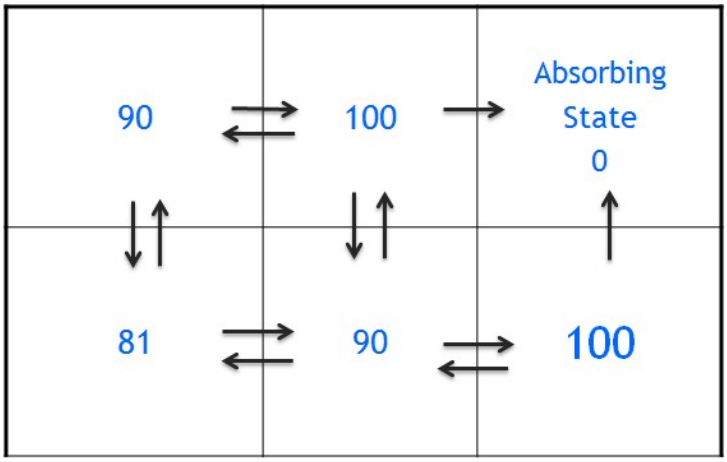
如下所示，每个格子代表一个状态s，箭头则代表动作a，旁边的数字代表立即回报，可以看到只有进入目标位置的动作获得了回报100，其他动作都获得了0回报。 即$r(s_{12},a_r) = r(s_{23},a_u) =100$。



2. 一个策略π如图所示：



3. 值函数$V^{\pi}(s)$如下所示



根据V^{π}的表达式，立即回报，和策略π，有

$V^{\pi}(s_{12}) = r(s_{12},a_r) = r(s_{13}|s_{12},a_r) = 100$

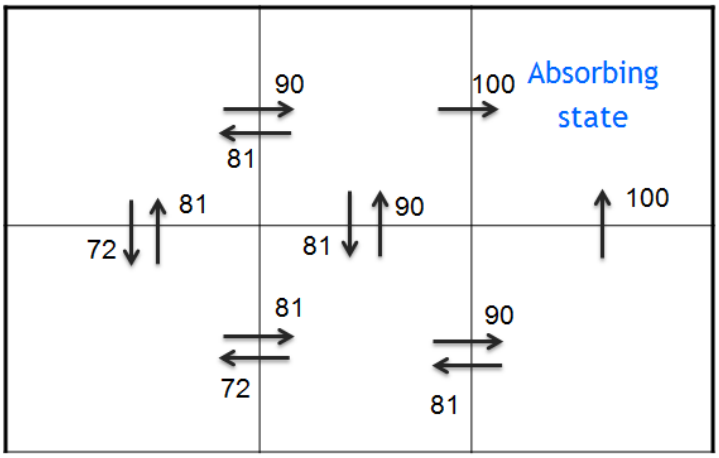
$V^{\pi}(s_{11}) = r(s_{11},a_r)+\gamma*V^{\pi}(s_{12}) = 0+0.9*100 = 90$

$V^{\pi}(s_{23}) = r(s_{23},a_u) = 100$

$V^{\pi}(s_{22}) = r(s_{22},a_r)+\gamma*V^{\pi}(s_{23}) = 90$

$V^{\pi}(s_{21}) = r(s_{21},a_r)+\gamma*V^{\pi}(s_{22}) = 81$

4. Q(s,a)值如下所示



有了策略π和立即回报函数r(s,a), $Q^{\pi}(s,a)$如何得到的呢?

对s_{11}计算Q函数（用到了上面V^{π}的结果）如下：

$Q^{\pi}(s_{11},a_r)=r(s_{11},a_r)+ \gamma *V^{\pi}(s_{12}) =0+0.9*100 = 90$

$Q^{\pi}(s_{11},a_d)=r(s_{11},a_d)+ \gamma *V^{\pi}(s_{21}) = 72$

至此我们了解了马尔可夫决策过程的基本概念，知道了增强学习的目标（获得任意初始条件下，使V^{π}值最大的策略π*），下一篇开始介绍求解最优策略的方法。

PS:发现写东西还是蛮辛苦的，希望对大家有帮助。另外自己也比较菜，没写对的地方欢迎指出~~

[注]采用折合因子作为值函数的MDP也可以定义为五元组M=(S, A, P, γ, R)。也有的书上把值函数作为一个因子定义五元组。还有定义为三元组的，不过MDP的基本组成元素是不变的。

参考资料：

[1] R.Sutton et al. Reinforcement learning: An introduction , 1998

[2] T.Mitchell. 《机器学习》，2003

[3] 金卓军，逆向增强学习和示教学习算法研究及其在智能机器人中的应用[D]，2011

[4] Oliver Sigaud et al，Markov Decision Process in Artificial Intelligence[M], 2010

上一篇我们已经说到了，增强学习的目的就是求解马尔可夫决策过程(MDP)的最优策略，使其在任意初始状态下，都能获得最大的V^{π}值。(本文不考虑非马尔可夫环境和不完全可观测马尔可夫决策过程(POMDP)中的增强学习)。

那么如何求解最优策略呢？基本的解法有三种：

动态规划法(dynamic programming methods)

蒙特卡罗方法(Monte Carlo methods)

时间差分法(temporal difference)。

动态规划法是其中最基本的算法，也是理解后续算法的基础，因此本文先介绍动态规划法求解MDP。本文假设拥有MDP模型M=(S, A, P_{sa}, R)的完整知识。

1. 贝尔曼方程（Bellman Equation）

上一篇我们得到了V^{π}和Q^{π}的表达式，并且写成了如下的形式

$$V^{\pi}(s)=\sum_{s'\in S}p(s'|s,\pi(s))\left[r(s'|s,\pi(s))+\gamma V^{\pi}(s')\right]=E_{\pi}\left[r(s'|s,a)+\gamma V^{\pi}(s')\middle|s_0=s\right]$$

$$Q^{\pi}(s,a)=\sum_{s'\in S}p(s'|s,a)\left[r(s'|s,a)+\gamma V^{\pi}(s')\right]=E_{\pi}\left[r(s'|s,a)+\gamma V^{\pi}(s')\middle|s_0=s,a_0=a\right]$$

在动态规划中，上面两个式子称为贝尔曼方程，它表明了当前状态的值函数与下个状态的值函数的关系。

优化目标π*可以表示为：

$$\pi^*(s)=\arg\max_{\pi}V^{\pi}(s)$$

分别记最优策略π*对应的状态值函数和行为值函数为V*(s)和Q*(s, a)，由它们的定义容易知道，V*(s)和Q*(s, a)存在如下关系:

$$V^*(s)=\max_{a}Q^*(s,a)$$

状态值函数和行为值函数分别满足如下贝尔曼最优性方程(Bellman optimality equation)：

$$V^*(s) = \max_a E\left[r(s'|s,a) + \gamma V^*(s') \middle| s_0 = s\right]$$
$$= \max_{a \in A(s)} \sum p(s'|s,\pi(s))\left[r(s'|s,\pi(s)) + \gamma V^{\pi}(s')\right]$$

$$Q^*(s) = E\left[r(s'|s,a) + \gamma \max_{a'} Q^*(s',a') \middle| s_0 = s, a_0 = a\right]$$
$$= \sum p(s'|s,\pi(s))\left[r(s'|s,\pi(s)) + \gamma \max_{a \in A(s)} Q^*(s',a')\right]$$

有了贝尔曼方程和贝尔曼最优性方程后，我们就可以用动态规划来求解MDP了。

2. 策略估计(Policy Evaluation)

首先，对于任意的策略π，我们如何计算其状态值函数$V^{\pi}(s)$？这个问题被称作策略估计，

前面讲到对于确定性策略，值函数

$$V^{\pi}(s) = \sum_{s' \in S} p(s'|s,\pi(s))\left[r(s'|s,\pi(s)) + \gamma V^{\pi}(s')\right]$$

现在扩展到更一般的情况，如果在某策略π下，π(s)对应的动作a有多种可能，每种可能记为π(a|s)，则状态值函数定义如下：

$$V^{\pi}(s) = \sum_a \pi(a|s) \sum_{s' \in S} p(s'|s,\pi(s))\left[r(s'|s,\pi(s)) + \gamma V^{\pi}(s')\right]$$

一般采用迭代的方法更新状态值函数，首先将所有$V^{\pi}(s)$的初值赋为0（其他状态也可以赋为任意值，不过吸收态必须赋0值），然后采用如下式子更新所有状态s的值函数（第k+1次迭代）：

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s' \in S} p(s'|s,a)\left[r(s'|s,a) + \gamma V_k(s')\right]$$

对于$V^{\pi}(s)$，有两种更新方法，

第一种：将第k次迭代的各状态值函数$[V^k(s_1), V^k(s_2), V^k(s_3)..]$保存在一个数组中，第k+1次的$V^{\pi}(s)$采用第k次的$V^{\pi}(s')$来计算，并将结果保存在第二个数组中。

第二种：即仅用一个数组保存各状态值函数，每当得到一个新值，就将旧的值覆盖,形如$[V^{k+1}(s_1), V^{k+1}(s_2), V^k(s_3)..]$，第k+1次迭代的$V^{\pi}(s)$可能用到第k+1次迭代得到的$V^{\pi}(s')$。

通常情况下，我们采用第二种方法更新数据，因为它及时利用了新值，能更快的收敛。整个策略估计算法如下图所示：

Input π, the policy to be evaluated
Initialize an array $v(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $v \approx v_\pi$

3. 策略改进(Policy Improvement)

上一节中进行策略估计的目的，是为了寻找更好的策略，这个过程叫做策略改进(Policy Improvement)。

假设我们有一个策略π，并且确定了它的所有状态的值函数$V^{\pi}(s)$。对于某状态s，有动作$a_0=\pi(s)$。 那么如果我们在状态s下不采用动作a_0，而采用其他动作$a\neq\pi(s)$是否会更好呢？要判断好坏就需要我们计算行为值函数$Q^{\pi}(s,a)$，公式我们前面已经说过：

$$Q^{\pi}(s,a)=\sum_{s'\in S} p(s'|s,a)\left[r(s'|s,a)+\gamma V^{\pi}(s')\right]$$

评判标准是：$Q^{\pi}(s,a)$是否大于$V^{\pi}(s)$。如果$Q^{\pi}(s,a)> V^{\pi}(s)$，那么至少说明新策略【仅在状态s下采用动作a，其他状态下遵循策略π】比旧策略【所有状态下都遵循策略π】整体上要更好。

策略改进定理(policy improvement theorem)：π和π'是两个确定的策略，如果对所有状态$s\in S$有$Q^{\pi}(s,\pi'(s))\geq V^{\pi}(s)$，那么策略π'必然比策略π更好，或者至少一样好。其中的不等式等价于$V^{\pi'}(s)\geq V^{\pi}(s)$。

有了在某状态s上改进策略的方法和策略改进定理，我们可以遍历所有状态和所有可能的动作a，并采用贪心策略来获得新策略π'。即对所有的$s\in S$, 采用下式更新策略：

$$\begin{aligned}\pi'(s)&=\arg\max_a Q^{\pi}(s,a)\\&=\arg\max_a E_{\pi}\left[r(s'|s,a)+\gamma V^{\pi}(s')\middle|s_0=s,a_0=a\right]\\&=\arg\max_a \sum_{s'\in S} p(s'|s,a)\left[r(s'|s,a)+\gamma V^{\pi}(s')\right]\end{aligned}$$

这种采用关于值函数的贪心策略获得新策略，改进旧策略的过程，称为策略改进(Policy Improvement)

最后大家可能会疑惑，贪心策略能否收敛到最优策略，这里我们假设策略改进过程已经收敛，即对所有的s，$V^{\pi'}(s)$等于$V^{\pi}(s)$。那么根据上面的策略更新的式子，可以知道对于所有的$s\in S$下式成立：

$$\begin{aligned}V^{\pi'}(s)&=\max_a E\left[r(s'|s,a)+\gamma V^{\pi'}(s')\middle|s_0=s\right]\\&=\max_a \sum_{s'\in S} p(s'|s,a)\left[r(s'|s,a)+\gamma V^{\pi'}(s')\right]\end{aligned}$$

可是这个式子正好就是我们在1中所说的Bellman optimality equation，所以π和π'都必然是最优策略！神奇吧！

4. 策略迭代(Policy Iteration)

策略迭代算法就是上面两节内容的组合。假设我们有一个策略π，那么我们可以用policy evaluation获得它的值函数$V^{\pi}(s)$，然后根据policy improvement得到更好的策略π'，接着再计算$V^{\pi'}(s)$,再获得更好的策略π''，整个过程顺序进行如下图所示：

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

完整的算法如下图所示：

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \left[r(s, \pi(s), s') + \gamma v(s') \right]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a) \left[r(s, a, s') + \gamma v(s') \right]$
      If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

5. 值迭代(Value Iteration)

从上面我们可以看到，策略迭代算法包含了一个策略估计的过程，而策略估计则需要扫描(sweep)所有的状态若干次，其中巨大的计算量直接影响了策略迭代算法的效率。我们必须要获得精确的V^{π}值吗？事实上不必，有几种方法可以在保证算法收敛的情况下，缩短策略估计的过程。

值迭代（Value Iteration）就是其中非常重要的一种。它的每次迭代只扫描(sweep)了每个状态一次。值迭代的每次迭代对所有的s∈S按照下列公式更新：

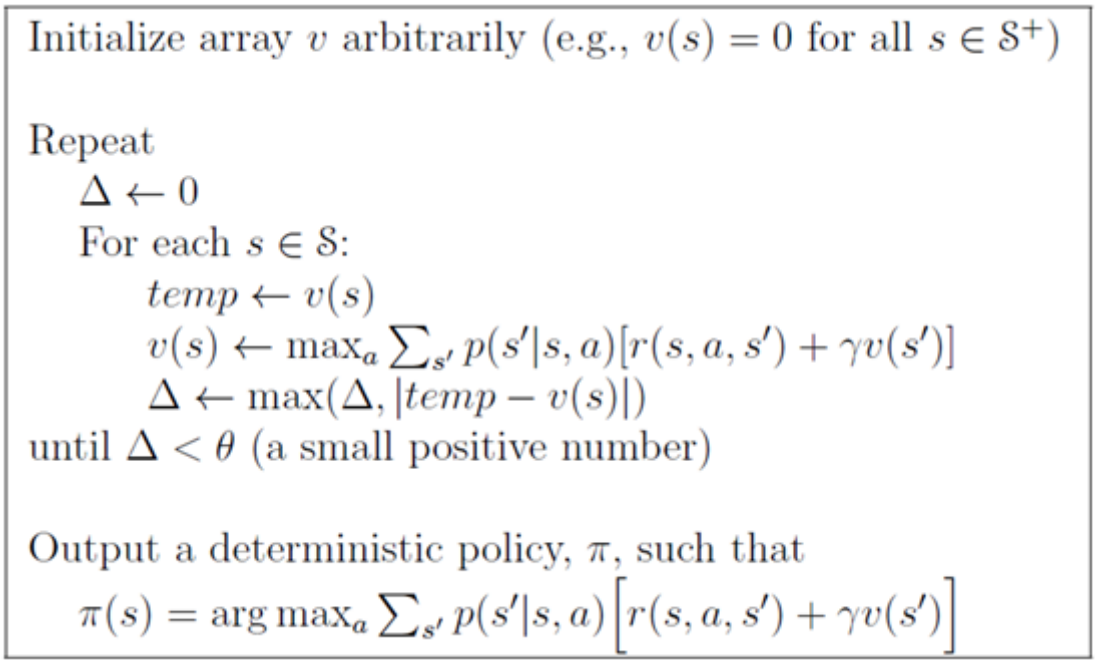
$$V_{k+1}(s) = \max_a E\left[r(s' \mid s, a) + \gamma V_k(s') \middle| s_0 = s \right]$$
$$= \max_a \sum p(s' \mid s, \pi(s)) \left[r(s' \mid s, \pi(s)) + \gamma V_k(s') \right]$$

即在值迭代的第k+1次迭代时，直接将能获得的最大的$V^{\pi}(s)$值赋给V_{k+1}。值迭代算法直接用可能转到的下一步s'的V(s')来更新当前的V(s)，算法甚至都不需要存储策略π。而实际上这种更新方式同时却改变了策略π_k和V(s)

的估值$V_k(s)$。 直到算法结束后，我们再通过V值来获得最优的π。

此外，值迭代还可以理解成是采用迭代的方式逼近1中所示的贝尔曼最优方程。

值迭代完整的算法如图所示：



由上面的算法可知，值迭代的最后一步，我们才根据V*(s)，获得最优策略π*。

一般来说值迭代和策略迭代都需要经过无数轮迭代才能精确的收敛到V*和π*， 而实践中，我们往往设定一个阈值来作为中止条件，即当$V^{\pi}(s)$值改变很小时，我们就近似的认为获得了最优策略。在折扣回报的有限MDP(discounted finite MDPs)中，进过有限次迭代，两种算法都能收敛到最优策略π*。

至此我们了解了马尔可夫决策过程的动态规划解法，动态规划的优点在于它有很好的数学上的解释，但是动态要求一个完全已知的环境模型，这在现实中是很难做到的。另外，当状态数量较大的时候，动态规划法的效率也将是一个问题。下一篇介绍蒙特卡罗方法，它的优点在于不需要完整的环境模型。

PS: 如果什么没讲清楚的地方，欢迎提出，我会补充说明...

参考资料：

[1] R.Sutton et al. Reinforcement learning: An introduction , 1998

[2] 徐昕，增强学习及其在移动机器人导航与控制中的应用研究[D],2002