强化学习基础篇（七）动态规划之价值迭代

1、最优化原理(Principle of optimality)

我们可以将一个最优策略分解为两个部分：

- 一、在当前状态下采用了最优的动作A_*
- 二、在后续状态S'开始沿着最优策略继续进行。

强化学习中的最优性定理可以表示为：

一个策略$\pi(a|s)$ 能够使得状态s获得最优价值$v_\pi(s) = v_*(s)$，当且仅当：

- 对于所有的状态s可到达任何状态s'。
- π从状态s'中可以获得最优价值，即$v_\pi(s') = v_*(s')$。

2、确定性价值迭代（Deterministic Value Iteration）

如果我们可以知道子问题$v_*(s')$的解，即已经获得下一状态s'的最优价值函数。那么我们就可以很容易得到当前状态s的最优价值函数：

$$v_*(s) \leftarrow \max_{a \in A}[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')]$$

在最优性定理的基础上，如果我们清楚地知道我们期望的最终状态的位置以及明确的状态间关系，那么可以认为是一个确定性的价值迭代。此时，我们可以把问题分解成一些列的子问题，从最终目标状态开始分析，逐渐往回推，直至推至所有状态。

3、最短路径问题（Shortest Path）

问题定义

在一个4×4的方格世界中，找到任意一个方格到最左上角方格的最短路径。其中在终点(g)处的奖励为0，其余状态的奖励为-1。

g	A	B	C
D	E	F	G
H	I	J	K
L	M	N	O

思路：

终点是左上角写着g的灰色格，所以，在迭代的过程中，距离g最近的那两个格子（即A和D）的$V(s)$最先可以算出最大值。对于A来说，只要选择向左，就可以到达终点获得reward，而向右则没有奖赏。所以根据公式中的max，A就会选择左走，它的V值最大，所以第一次迭代它就已经可以获得最大值了，然后接下来就是对于B来说，第一次迭代向左和向右都一样，但是在第二次迭代中就会发现两次向左就可以达到最大值。这就是值迭代的过程。

求解过程：

一、初始化每个状态的V为0。

二、对于第一次迭代$k=1$：

$$V_2(s)=\max_{a\in A}[R_s^a+\gamma\sum_{s'\in S}P_{ss'}^aV_1(s')]$$

由于对于$\forall s\in S:V_1(s)=0$，所以就可以得出$\forall s\in S:V_2(s)=-1$。

三、对于第二次迭代$k=2$：

$$V_3(s)=\max_{a\in A}[R_s^a+\gamma\sum_{s'\in S}P_{ss'}^aV_2(s')]$$

对于状态A:

$$\begin{aligned}V_3(A)&=\max_{a\in(N,S,W,E)}\{(R_A^N+\gamma P_{AA'}^NV_2(A)),(R_A^S+\gamma P_{AA'}^SV_2(E)),(R_A^W+\gamma P_{AA'}^WV_2(g)),(R_A^E+\gamma P_{AA'}^EV_2(B))\}\\&=\max_{a\in(N,S,W,E)}\{(-1+1*(-1)),(-1+1*(-1)),(-1+1*(0)),(-1+1*(-1))\}\\&=-1\end{aligned}$$

对于状态B:

$$\begin{aligned}V_3(B)&=\max_{a\in(N,S,W,E)}\{(R_B^N+\gamma P_{BB'}^NV_2(B)),(R_B^S+\gamma P_{BF'}^SV_2(F)),(R_B^W+\gamma P_{BA'}^WV_2(A)),(R_B^E+\gamma P_{BC'}^EV_2(C))\}\\&=\max_{a\in(N,S,W,E)}\{(-1+1*(-1)),(-1+1*(-1)),(-1+1*(-1)),(-1+1*(-1))\}\\&=-2\end{aligned}$$

以此类推，直到每个状态的累积奖赏都不变。

结果：

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

4、价值迭代（Value Iteration）

问题：

寻找最优策略π。

解决方案：

从初始状态价值开始同步迭代计算，最终收敛，整个过程中没有遵循任何策略。

与策略迭代一样，价值迭代最终也同样收敛到最优值函数，不过，与策略迭代不一样的是，价值迭代没有显式的策略，它通过贝尔曼最优方程，隐式地实现了策略改进这一步。值得注意的是，中间过程中的价值函数可能并不对应于任何策略

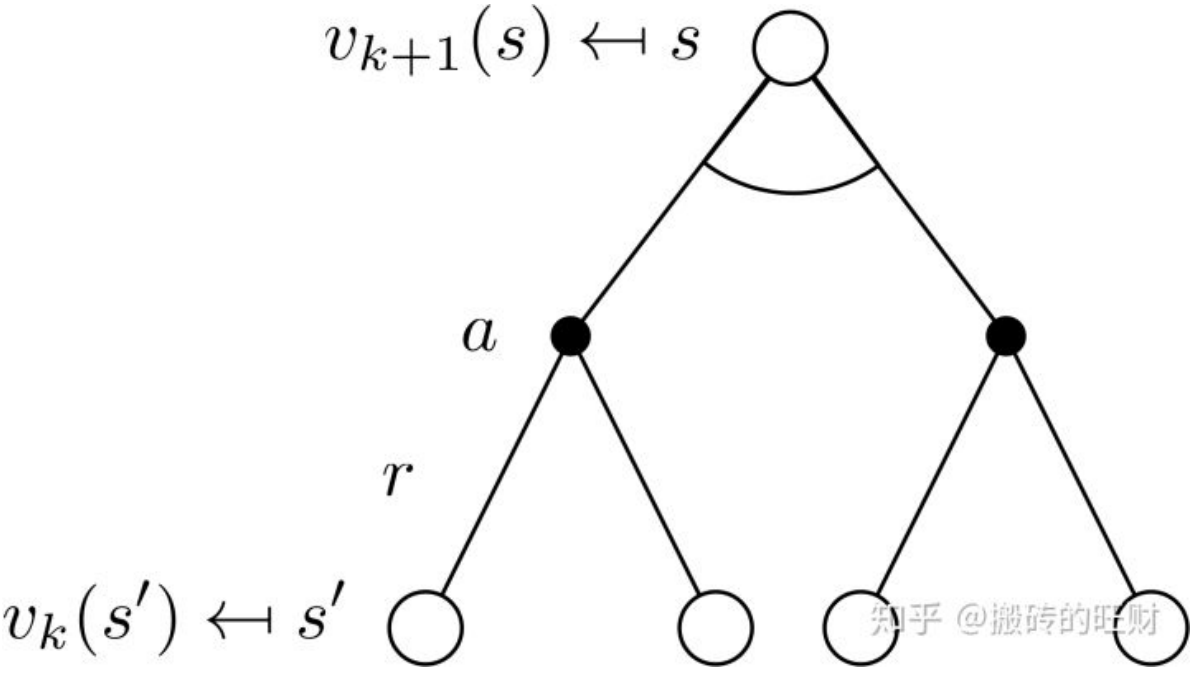
价值迭代虽然不需要策略参与，但仍然需要知道状态之间的转移概率，也就是需要知道模型。

算法描述

算法 3 同步备份下的值迭代算法

1: for $k = 1, 2, \cdots$ do
2: for 所有的状态 $s \in \mathcal{S}$ do
3: 通过 $v_k(s')$ 更新 $v_{k+1}(s)$
4: end for
5: end for

我们通过回溯（backup）图来对价值迭代进行理解：



其更新过程为贝尔曼最优方程：

一般形式为：

$$v_{k+1}(s) = \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$$

矩阵形式为：

$$v_{k+1} = \max_{a \in A}(R^a + \gamma P^a v_k)$$

与贝尔曼期望方程相比，贝尔曼最优方程将期望操作变为最大操作，也即隐式地实现了策略改进这一步，不过是固定为greedy的，而不是其他策略改进方法，我们通过greedy得到的策略通常为确定性策略。

5、策略评估（Policy Evaluation）、策略迭代(Policy Iteration)以及值迭代(Value Iteration)总结

这三个算法通过一张表可以总结如下：

问题（Problem）	**应用的贝尔曼方程 （Bellman Equation）**	**使用到的算法（Algorithm）**
预测(Prediction)	贝尔曼期望方程（Bellman Expectation Equation）	迭代式策略评估（Iterative Policy Evaluation）
控制（Control）	贝尔曼期望方程（Bellman Expectation Equation）+ 贪婪策略改进（Greedy Policy Improvement）	策略迭代(Policy Iteration)
控制（Control）	贝尔曼最优方程（Bellman Optimality Equation）	值迭代(Value Iteration)

- 策略评估（Policy Evaluation）、策略迭代(Policy Iteration)以及值迭代(Value Iteration)这三个算法都是基于状态值函数$v_\pi(s)$或者$v_*(s)$
- 如果动作数量为m，状态数量为n，那么每次迭代的计算复杂度为$O(mn^2)$。
- 这三个算法也同样可以应用到动作值函数$q_\pi(s,a)$或者$q_*(s,a)$，同时每次迭代的计算复杂度为$O(m^2n^2)$。