动态规划算法

简介

动态规划就是将待求解问题**分解成若干个子问题**，先求解子问题然后从这些子问题的解得到目标问题的解。动态规划会**保存已解决的子问题的答案**，在求解目标问题的过程中，需要这些子问题的答案时就可以直接利用，避免重复计算

基于动态规划的强化学习算法主要有两种：**策略迭代**、**价值迭代**

策略迭代中的策略评估使用贝尔曼期望方程来得到一个策略的状态价值函数，这是一个动态规划的过程；而价值迭代直接使用贝尔曼最优方程来进行动态规划，得到最终的最优状态价值。

局限：

（1）基于动态规划的这两种强化学习算法要求事先知道环境的状态转移函数和奖励函数，也就是需要知道整个马尔可夫决策过程，但在实际现实中这样的环境往往是很少的。

（2）策略迭代和价值迭代通常只适用于有限马尔可夫决策过程，即状态空间和动作空间是离散且有限的。

下面详细介绍一下这两种主要的算法

策略迭代

对于一个动作空间和状态空间有限的MDP可以使用策略迭代

策略迭代由两部分组成，策略评估和策略提升。通过这两部分来介绍策略迭代

策略评估

策略评估是用来计算策略的状态价值函数，MDP的状态的价值函数可以通过贝尔曼期望方程得到

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}[R_t + \gamma V^{\pi}(S_{t+1})|S_t = s] \\
&= \sum_{a \in A} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi}(s') \right)
\end{aligned}
$$

如上所示，所以我们可以利用动态规划的算法思想通过贝尔曼期望方程来对当前状态的价值求解

动态规划的算法思想就是将当前待求解的问题划分成子问题，然后通过解决子问题来解决当前问题，并且在求解子问题后会将子问题的解保存起来，等需要利用的时候再次利用从而避免重复的计算

所以对于贝尔曼期望方程，我们将求当前所求策略π下的状态价值作为当前要解决的问题，在更新当前状态价值时需要这个状态的下一个状态的价值函数（即贝尔曼期望方程等式左边的**V(s')**），我们把下一个可能的状态的价值函数当作子问题，子问题求解后根据贝尔曼期望方程就能求出当前状态价值函数了（当然前提状态转移函数以及奖励函数已知）

而更一般的情况，对于所有的状态的，我们可以利用上一轮迭代完后的价值函数来求解当前轮的所有状态价值函数

而这就涉及同步迭代与异步迭代

同步迭代与异步迭代

同步迭代：

同步迭代是指每个状态的价值函数都有两份价值拷贝，当前轮状态价值函数（$V_{new}(s)$），和上一轮状态价值函数（$V_{old}(s)$）

我们在用贝尔曼期望方程更新时，当前需要更新的状态价值在更新时都使用上一轮的状态价值函数进行迭代，这就是同步迭代

$$V_{new}(s) \leftarrow \max_{a \in A}\left(R(s)+\gamma \sum_{s' \in S} P_{sa}(s') V_{old}(s')\right)$$

这就保证了当前状态的价值函数在更新时，使用的数值更新都是同步的

然后当前所有状态的价值更新完成后，将上一轮的$V_{old}(s)$整体换成当前轮的$V_{new}(s)$，在进行下一轮迭代

$$V_{old}(s) \leftarrow V_{new}(s)$$

直到状态价值函数进行收敛

缺点：对于每个状态的价值函数都需要两个备份，所以会需要两倍的内存进行存储

异步迭代：

异步迭代每个状态的价值函数在内存中只有一个备份，所以更新时直接用当前的价值函数进行更新

所以当前状态价值更新时有的状态的价值函数作为被更新值，而有的已经用于更新

这就使每个状态的价值函数不是更新的，好处是相比同步迭代使用了一半的内存，而局限是可能在更新过程中存在一定的紊乱

初始化

在进行迭代时可以对每个状态的价值选定任意初始值V^0，当迭代k次后（很多很多次）$V^k = V^\pi$ 即为我们所求，以上也是更新公式的**不动点**

（不动点：不动点，是一个函数术语，在数学中是指“被这个函数映射到其自身一个点”。在函数的有限次迭代之后，回到相同值的点叫做周期点；不动点是周期等于 1 的周期点）

可以证明，当$k \to \infty$时，序列$\{V^k\}$会收敛到V^π，所以对于每个状态价值函数初始化时给定任意数值不断进行迭代就能得到在该策略下的状态价值

（**书上4.7节的扩展阅读有收敛证明**）

所以迭代次数k可能是一个很大很大的数，贝尔曼期望方程的迭代会使策略评估耗费很大很大的计算代价，不过在实际过程中，当某一轮的

$$\max_{s \in \mathcal{S}} |V^{k+1}(s) - V^k(s)$$

非常小，就可以提取结束策略评估，这样做可以提升效率，并且得到的价值也非常接近真实的价值。

策略提升：

策略评估计算得到当前策略的状态价值函数之后，就可以根据状态价值函数来改进该策略。

已知在一个策略π下的某个状态s的价值函数$V^{\pi}(s)$，假设智能体在该状态s依旧遵循策略π采取动作a，得到的动作价值函数$Q^{\pi}(s,a)$大于当前策略评估得到的$V^{\pi}(s)$

说明在状态s下采取动作a会比原来的策略π(a|s)能得到更高的期望回报，所以存在一个**π'**在任意一个状态s下有

$$Q^{\pi}(s,\pi'(s)) \geq V^{\pi}(s)$$

所以对于任意状态有：

$$V^{\pi'}(s) \geq V^{\pi}(s)$$

这便是**策略提升**，于是可以贪心地在每一个状态选择动作价值最大的动作作为新的要采取的策略**π'**

$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a)$$

（argmax 是求函数值最大值对应的自变量，这里求的是动作a这个参数自变量）

这样构造的π' 不光满足策略提升定理的条件，它能够比策略π更好或者说至少一样好

这是通过贪心算法得到的策略提升，但如果得到提升策略与原策略一样时，说明策略迭代达到了收敛，此时在该策略π的基础上策略π'就是最优策略

价值迭代

价值迭代也是对于一个动作空间和状态空间有限的MDP使用，对于策略迭代中的策略评估，需要进行很多轮才能收敛得到某一策略的状态函数，然后再进行策略提升，当状态和动作空间比较大时，这需要很大的计算量。在有时这样是不可取的，那么能不能在一轮策略评估后就进行策略提升呢？

当然是可以的，这就是价值迭代，价值迭代概括的说就是策略评估只进行了一轮更新的策略迭代算法

价值迭代的过程：

- 对于每一个状态s，首先初始化价值函数V(s) = 0
- 然后对于每一个状态的V(s)，根据贝尔曼最优方程

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')V(s')$$

不断进行迭代，直到V(s)收敛，由于我们有一个折扣因子在0~1之间，所以会最终收敛到一个有限的数

然后利用V(s)去计算一个策略

在以上的计算中没有明确的策略去执行动作，而是以价值函数作为中心来进行不断迭代，迭代到价值函数最优来反过来确定策略，此时价值函数为最优价值函数，故有

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

最优状态价值是选择此时使最优动作价值最大的那一个动作时的状态价值

所以可以根据最优价值函数来确定策略

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$

$$\pi(s) = \arg\max_a \{r(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')\}$$