计算机科学188：人工智能导论
2024年春季 笔记18

作者（其他所有笔记）：尼基尔·夏尔马

作者（贝叶斯网络笔记）：乔希·胡格和杰基·梁，由王瑞吉编辑

作者（逻辑笔记）：亨利·朱，由考佩林编辑

致谢（机器学习与逻辑笔记）：部分章节改编自教材《人工智能：一种现代方法》。

最后更新时间：2023年8月26日

值迭代

既然我们有了一个用于测试马尔可夫决策过程（MDP）中状态值最优性的框架，接下来自然要问的问题是如何实际计算这些最优值。为了回答这个问题，我们需要有时间限制的值（这是强制设定有限时间范围的自然结果）。对于一个时间限制为 k 个时间步的状态 s，其时间限制值记为 $U_k(s)$，表示在考虑的马尔可夫决策过程在 k 个时间步终止的情况下，从 s 可获得的最大期望效用。等效地，这是在MDP的搜索树上进行深度为 k 的期望最大化运行所返回的值。

值迭代是一种动态规划算法，它使用迭代更长的时间限制来计算有时间限制的值，直到收敛（也就是说，直到每个状态的 U 值与上一次迭代中的值相同：$\forall s, U_{k+1}(s) = U_k(s)$）。其操作如下：

1. $\forall s \in S$，初始化 $U_0(s) = 0$。这应该是直观的，因为设置时间步长为0意味着在终止之前不能采取任何行动，因此也无法获得任何奖励。
2. 重复以下更新规则，直到收敛：

$$\forall s \in S,\ U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U_k(s')]$$

在值迭代的第 k 次迭代中，我们使用每个状态的限时为 k 的限时值来生成限时为 $(k+1)$ 的限时值。本质上，我们使用子问题（所有 $U_k(s)$）的计算解来迭代构建更大子问题（所有 $U_{k+1}(s)$）的解；这就是值迭代成为动态规划算法的原因。

请注意，虽然贝尔曼方程在结构上看起来与上述更新规则基本相同，但它们并不一样。贝尔曼方程给出了最优性条件，而更新规则给出了一种迭代更新值直到收敛的方法。当达到收敛时，贝尔曼方程将对每个状态都成立：$\forall s \in S, U_k(s) = U_{k+1}(s) = U^*(s)$。

为简洁起见，我们经常用简写形式 $U_{k+1} \leftarrow BU_k$ 来表示
$U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma U_k(s')\right]$，其中 B 被称为贝尔曼算子。贝尔曼算子是一个关于 γ 的压缩映射。为证明这一点，我们需要以下一般不等式：

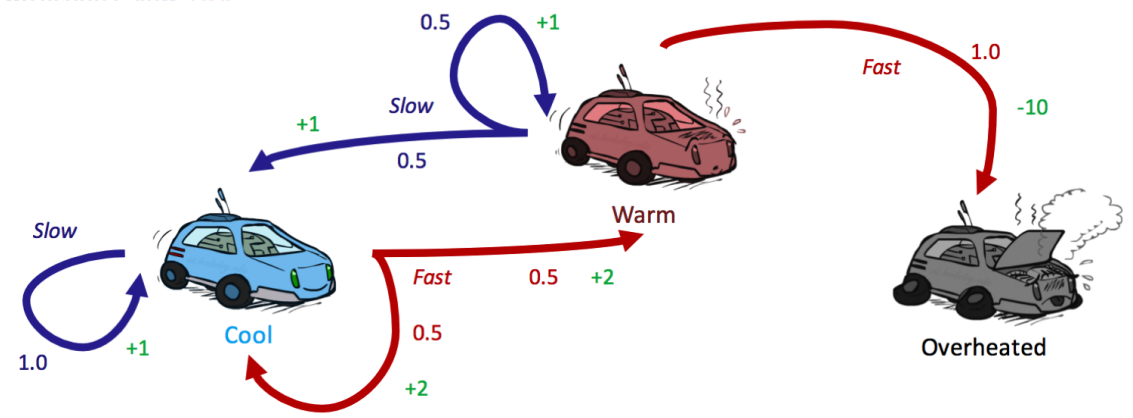
$$|max_z f(z) - max_z h(z)| \leq max_z |f(z) - h(z)|.$$

现在考虑在相同状态 $U(s)$ 和 $U'(s)$ 下评估的两个值函数。我们证明，关于最大范数，贝尔曼更新 B 是一个由 $\gamma \in (0,1)$ 构成的压缩映射，如下所示

$$\begin{aligned}
&|BU(s) - BU'(s)| \\
&= \left| \left(\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U(s')] \right) - \left(\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U'(s')] \right) \right| \\
&\leq \max_a \left| \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U(s')] - \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U'(s')] \right| \\
&= \max_a \left| \gamma \sum_{s'} T(s,a,s') U(s') - \gamma \sum_{s'} T(s,a,s') U'(s') \right| \\
&= \gamma \max_a \left| \sum_{s'} T(s,a,s')(U(s') - U'(s')) \right| \\
&\leq \gamma \max_a \left| \sum_{s'} T(s,a,s') \max_{s'} |U(s') - U'(s')| \right| \\
&= \gamma \max_{s'} |U(s') - U'(s')| \\
&= \gamma ||U(s') - U'(s')||_\infty,
\end{aligned}$$

其中第一个不等式由上述引入的一般不等式得出，第二个不等式通过取 U 和 U' 之间差值的最大值得出，最后在倒数第二步中，我们利用无论 a 如何选择概率总和都为1这一事实。最后一步使用向量 $x = (x_1, \ldots, x_n)$ 的最大范数定义，即 $\| x\|_\infty = \max(|x_1|, \ldots, |x_n|)$。

因为我们刚刚证明了通过贝尔曼更新进行的值迭代在 γ 中是一种收缩，所以我们知道值迭代会收敛，并且当我们达到满足 $U^* = BU^*$ 的不动点时就会发生收敛。
让我们通过回顾之前的赛车马尔可夫决策过程来实际看一些值迭代的更新情况，引入一个 $\gamma = 0.5$ 的折扣因子：



我们通过初始化所有 $U_0(s) = 0$ 来开始值迭代：

	cool	warm	overheated
U_0	0	0	0

在第一轮更新中，我们可以如下计算 $\forall s \in S, U_1(s)$：

$$\begin{aligned}
U_1(cool) &= \max\{1\cdot[1+0.5\cdot 0],\ 0.5\cdot[2+0.5\cdot 0]+0.5\cdot[2+0.5\cdot 0]\} \\
&= \max\{1,2\} \\
&= \boxed{2} \\
U_1(warm) &= \max\{0.5\cdot[1+0.5\cdot 0]+0.5\cdot[1+0.5\cdot 0],\ 1\cdot[-10+0.5\cdot 0]\} \\
&= \max\{1,-10\} \\
&= \boxed{1} \\
U_1(overheated) &= \max\{\} \\
&= \boxed{0}
\end{aligned}$$

	cool	warm	overheated
U_0	0	0	0
U_1	2	1	0

类似地，我们可以重复该过程，使用新得到的 $U_1(s)$ 值来计算第二轮更新，以得到 $U_2(s)$。

$$\begin{aligned}
U_2(cool) &= \max\{1\cdot[1+0.5\cdot 2],\ 0.5\cdot[2+0.5\cdot 2]+0.5\cdot[2+0.5\cdot 1]\} \\
&= \max\{2,2.75\} \\
&= \boxed{2.75} \\
U_2(warm) &= \max\{0.5\cdot[1+0.5\cdot 2]+0.5\cdot[1+0.5\cdot 1],\ 1\cdot[-10+0.5\cdot 0]\} \\
&= \max\{1.75,-10\} \\
&= \boxed{1.75} \\
U_2(overheated) &= \max\{\} \\
&= \boxed{0}
\end{aligned}$$

	cool	warm	overheated
U_0	0	0	0
U_1	2	1	0
U_2	2.75	1.75	0

值得注意的是，任何终端状态的 $U^*(s)$ 必定为 0，因为从任何终端状态都无法采取行动来获得任何奖励。

策略提取

回想一下，我们解决马尔可夫决策过程的最终目标是确定最优策略。这可以在使用一种称为策略提取的方法确定所有状态的最优值之后完成。策略提取背后的直觉非常简单：如果你处于状态 s，你应该采取能产生最大期望效用的行动 a。

不出所料， a 是能让我们达到具有最大Q值的Q状态的动作，由此可对最优策略进行形式化定义：

$$\forall s \in S,\ \pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^*(s')]$$

出于性能考虑，记住这一点很有用：对于策略提取而言，拥有状态的最优Q值会更好，在这种情况下，只需一次求最大值操作就能从一个状态确定最优动作。仅存储每个 $U^*(s)$ 意味着在应用求最大值操作之前，我们必须使用贝尔曼方程重新计算所有必要的Q值，这等同于执行深度为1的期望最大化。

Q值迭代

在使用值迭代求解最优策略时，我们首先找到所有最优值，然后使用策略提取来提取策略。然而，你可能已经注意到，我们还处理另一种编码有关最优策略信息的值：Q值。

Q值迭代是一种计算有时间限制的Q值的动态规划算法。它由以下等式描述：

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$$

请注意，此更新只是对值迭代更新规则的轻微修改。实际上，唯一真正的区别在于，由于当我们处于某个状态时在转移之前选择动作，而当我们处于Q状态时在选择新动作之前进行转移，所以动作上的max运算符的位置发生了变化。一旦我们得到了每个状态和动作的最优Q值，那么我们就可以通过简单地选择具有最高Q值的动作来找到某个状态的策略。

策略迭代

值迭代可能会相当缓慢。在每次迭代中，我们必须更新所有 $|S|$ 状态的值（其中 $|n|$ 指基数运算符），计算每个动作的Q值时，每个状态都需要遍历所有 $|A|$ 动作。反过来，计算这些Q值中的每一个又需要再次遍历每个 $|S|$ 状态，导致运行时间较差为 $O\left(|S|^2\,|A|\right)$ 。此外，当我们只想确定MDP的最优策略时，值迭代往往会进行大量的过度计算，因为通过策略提取计算出的策略通常比值本身收敛得快得多。解决这些缺陷的方法是使用策略迭代作为替代方案，该算法在保持值迭代最优性的同时提供显著的性能提升。策略迭代的操作如下：

1. 定义一个初始策略。这可以是任意的，但初始策略越接近最终的最优策略，策略迭代收敛得就越快。
2. 重复以下步骤直至收敛：

    • 使用策略评估来评估当前策略。对于策略 π ，策略评估意味着为所有状态 s 计算 $U^\pi(s)$ ，其中 $U^\pi(s)$ 是在遵循 π 时从状态 s 开始的期望效用：

$$U^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma U^\pi(s')]$$

将策略迭代在迭代 i 时的策略定义为 π_i。由于我们为每个状态固定了一个单一动作，我们不再需要最大值运算符，这实际上使我们得到了一个由上述规则生成的 $|S|$ 方程组。然后，每个 $U^{\pi_i}(s)$ 都可以通过简单地求解这个方程组来计算。或者，我们也可以像在值迭代中一样，使用以下更新规则来计算 $U^{\pi_i}(s)$，直到收敛：

$$U_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma U_k^{\pi_i}(s')]$$

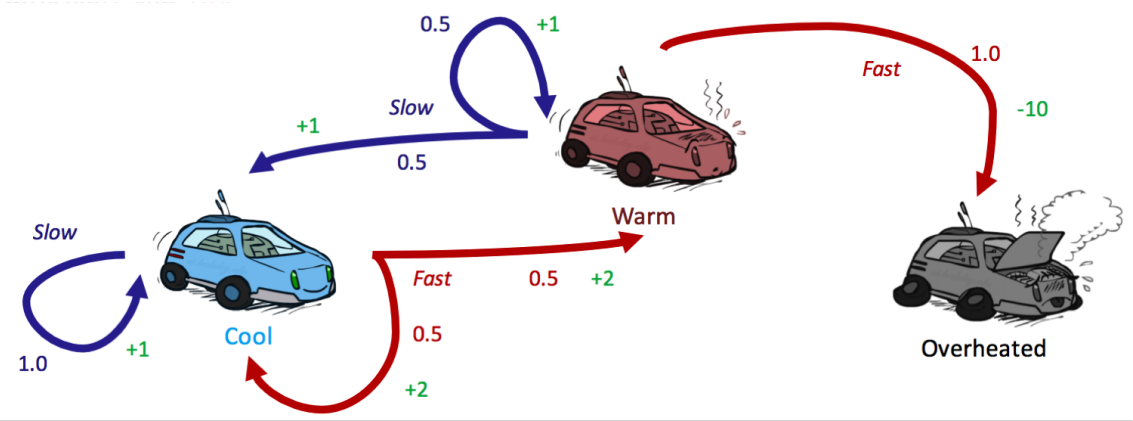
然而，在实践中，第二种方法通常较慢。

- 一旦我们评估了当前策略，使用策略改进来生成更好的策略。策略改进对由策略评估生成的状态值进行策略提取，以生成这个新的改进策略：

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma U^{\pi_i}(s')]$$

如果 $\pi_{i+1} = \pi_i$，则算法已经收敛，我们可以得出结论 $\pi_{i+1} = \pi_i = \pi^*$。

让我们最后再看一次我们的赛车示例（是不是已经看腻了？），看看使用策略迭代是否能得到与值迭代相同的策略。回想一下，我们使用的折扣因子是 $\gamma = 0.5$。



我们从一个初始策略开始：总是慢速行驶

	cool	warm	overheated
π_0	*slow*	*slow*	–

由于终止状态没有外向动作，没有策略可以为其赋值。因此，像我们所做的那样，将过热状态从考虑中忽略是合理的，并且只需为任何终止状态 s 赋值 $\forall i, U^{\pi_i}(s) = 0$。下一步是对 π_0 进行一轮策略评估：

$$\begin{aligned} U^{\pi_0}(cool) &= 1 \cdot [1 + 0.5 \cdot U^{\pi_0}(cool)] \\ U^{\pi_0}(warm) &= 0.5 \cdot [1 + 0.5 \cdot U^{\pi_0}(cool)] + 0.5 \cdot [1 + 0.5 \cdot U^{\pi_0}(warm)] \end{aligned}$$

求解关于 $U^{\pi_0}(\text{cool})$ 和 $U^{\pi_0}(\text{warm})$ 的这个方程组，得到：

	cool	warm	overheated
U^{π_0}	2	2	0

现在我们可以使用这些值来运行策略提取：

$$\begin{aligned}
\pi_1(cool) &= \text{argmax}\{slow : 1 \cdot [1 + 0.5 \cdot 2],\ fast : 0.5 \cdot [2 + 0.5 \cdot 2] + 0.5 \cdot [2 + 0.5 \cdot 2]\} \\
&= \text{argmax}\{slow : 2,\ fast : 3\} \\
&= \boxed{fast} \\
\pi_1(warm) &= \text{argmax}\{slow : 0.5 \cdot [1 + 0.5 \cdot 2] + 0.5 \cdot [1 + 0.5 \cdot 2],\ fast : 1 \cdot [-10 + 0.5 \cdot 0]\} \\
&= \text{argmax}\{slow : 3,\ fast : -10\} \\
&= \boxed{slow}
\end{aligned}$$

第二轮运行策略迭代会产生 $\pi_2(\text{cool}) =$ 快速和 $\pi_2(\text{warm}) =$ 慢速。由于这与 π_1 是相同的策略，我们可以得出 $\pi_1 = \pi_2 = \pi^*$。实际验证一下！

	cool	warm
π_0	*slow*	*slow*
π_1	*fast*	*slow*
π_2	*fast*	*slow*

这个例子展示了策略迭代的真正威力：仅经过两次迭代，我们就已经得到了我们的赛车马尔可夫决策过程的最优策略！这比我们在同一个马尔可夫决策过程上运行值迭代时的情况要好，在我们执行的两次更新之后，值迭代距离收敛仍需要几次迭代。

总结

上述内容很容易让人混淆。我们介绍了值迭代、策略迭代、策略提取和策略评估，所有这些看起来都很相似，都是使用带有细微变化的贝尔曼方程。以下是每种算法的目的总结：

- 值迭代：用于通过迭代更新直到收敛来计算状态的最优值。
- 策略评估：用于计算特定策略下状态的值。
- 策略提取：用于在给定某些状态值函数的情况下确定一个策略。如果状态值是最优的，那么这个策略将是最优的。此方法在运行值迭代后使用，以便从最优状态值计算最优策略；或者作为策略迭代中的一个子例程，用于为当前估计的状态值计算最佳策略。
- 策略迭代：一种封装了策略评估和策略提取的技术，用于迭代收敛到最优策略。由于策略通常比状态值收敛得快得多，所以它往往比值迭代表现更好。