强化学习经典入门书的读书笔记系列--第三篇（下）

原创 2017-05-24 小吕 神经网络与强化学习

今天是第三篇的（下），接着上次说的马尔科夫过程来继续讨论，并藉由一个详细的例子，阐述其细节。

3.6 Markov Decision Processes

如果一个强化学习任务满足马尔科夫性质，那么就可以把这个任务叫做马尔科夫过程。如果状态空间和动作空间是有限的，那么就叫做有限马尔科夫过程，即finite MDP。finite MDP对强化学习理论尤其重要。因此如果你能很好的理解finite MDP，那么你也就能理解90%强化学习的内容了。

一个典型的finite MDP 由状态集、动作集和一步内的环境动态性来定义。给定状态s和动作a，下一个可能状态$s^{'}$的可能性由下式给出：

$$p(s^{'}|s,a) = Pr(S_{t+1} = s^{'}|S_t = s, A_t = a)$$

这个等式叫做转移概率。相似的，给定任何当前的状态和动作，s和a，以及下一个状态$s^{'}$,下一个reward的期望价值由下式给出：

$$r(s,a,s^{'}) = E[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']$$

这些等式完整描述了finite MDP的动态性质。我们本书中的大多理论的前提假设就是环境是finite MDP。

下面通过一个具体的例子，来详细描述finite MDP。

Example 3.7:Recycling Robot MDP

回收机器人可以通过是适当的简化且提供一些细节，来近似转化成一个有限马尔科夫过程的例子。回忆一下agent是通过外部事件来做出决定。在每个这样的时刻，机器人决定是否应该（1）主动搜集易拉罐，（2）原地等待，直到某人捡来易拉罐，（3）回到基地充电。假设环境的工作特点如下：发现易拉罐的最佳方式是主动搜寻，但是这样会消耗电池的电量，然而在原地待命就不会。机器人只要在搜索，那么它的电量就有耗尽的可能性。在这种情况下，机器人就只能原地不动，等待被救援了（产生很低的reward）。

机器人的agent只依据它的当前电量来做出决定（1）、（2）还是（3）。它可以区别电量的两种状态：high和low。因此状态空间就是S={high,low}。我们把agent的动作，看作是wait、search、recharge。当电量状态是high，回去充电明显是愚蠢的，因此当电量为high，action set里面没有recharge这一项。agent的action set如下:

$$A(high) = [search, wait]$$
$$A(low) = [search, wait, recharge]$$

如果电量程度是high，那么在一段时间内主动搜寻过程总是可以没有风险地完成（没有电量耗尽的风险）。经过一段时间的搜寻，电量的状态变化可以是high-high，也可以是high-low（假设开始电量状态为high），前者的概率为α,后者为$1-\alpha$。另一方面，当搜寻开始时的电量状态为low时，经过一段时间的搜寻，电量状态的变化可以是low-low,也可以是low-{电量耗尽，被动拿回去充电，而非主动充电，电量变为high}，前者的概率为β,后者的概率为$1-\beta$。每当一个易拉罐被收集了，就给出一个单位奖励；而当机器人的电量耗尽被救援时，给出-3的奖励。假设r_{search}和r_{wait}分别代表机器人在主动搜索和原地等待时可以得到的期望奖励，且$r_{search} > r_{wait}$。最后为了使问题较为简单，假设当机器人回去充电的时候，不会收集易拉罐，并且当电池耗尽时，不再收集易拉罐。这个系统就是finite MDP，转移概率和期望奖励如下表所示：

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	r_{search}
high	low	search	$1-\alpha$	r_{search}
low	high	search	$1-\beta$	-3
low	low	search	β	r_{search}
high	high	wait	1	r_{wait}
high	low	wait	0	r_{wait}
low	high	wait	0	r_{wait}
low	low	wait	1	r_{wait}
low	high	recharge	1	0
low	low	recharge	0	$0.$

有时转移图也能很直观的表示finite MDP的转移过程，如下：

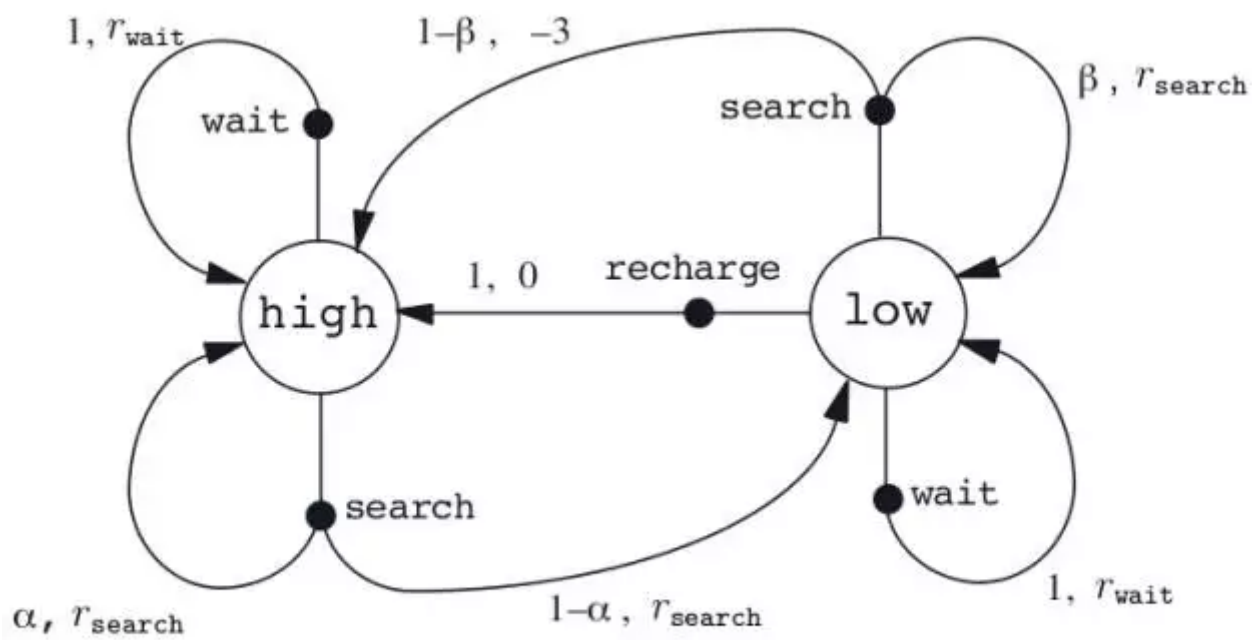

Figure 3.3: Transition graph for the recycling robot example.

3.7 Value Functions

几乎所有的强化学习算法都涉及到估计value function，这个function自变量是状态，用来判断在某个给定状态下做出某个动作的好坏程度。好坏程度用什么来定义呢？这里我们使用“未来的期望奖励”来定义，也可以说期望返回值。显然agent在未来期望得到的奖励取决于它做出什么决策。因此，value function在定义的时候和policy有关。

回忆一下policy的概念，是状态空间到动作空间的映射图，它定义了给定状态下做出给定动作的概率。因此，在某个policy π的前提下，某个状态s的value function $v_\pi(s)$,定义为从当前状态s开始，按照policy π的规则，一直走下去的期望返回值。公式如下：

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

我们把上式的$v_\pi(s)$叫做policy π下的状态-价值函数。

同样的，我们也可以类似的定义一个在policy π规则下，在状态为s时动作为a的value function，表示为$q_\pi(s,a)$。公式如下：

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

我们把$q_\pi(s,a)$叫做policy π下的动作-价值函数。

这两个函数可以通过经验来估计。比如agent为每个遇到的状态s保留其后所有真实返回奖励的average，那么随着实验次数的增加，遇到这个状态s的次数会趋近无穷，那么最终的average return也会趋近于其$v_\pi(s)$。同样的，如果我们为每个状态下的所有动作都保存其后所有真实返回奖励的average，那么我们也会得到精确的$q_\pi(s,a)$。我们把这种方法叫做蒙特卡洛法，因为它依赖于大量随机实验样本的均值。我们会在第五章介绍这个方法。然而当我们的状态空间很大，这么做有点不现实。我们也可以把$v_\pi(s)$和$q_\pi(s,a)$看成是带有参数的方程，通过调节参数，使得其结果和观测到的返回值更接近。这种方法也可以带来精确的估计，尽管很大程度上（精确性）依赖于如何选择近似方式。

强化学习中的value function的一个重要性质是它满足一种特别的递归关系。对于任何policy π和任何状态s，当前状态的value和其后的可能状态的value之间的关系如下式：

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v_\pi(s')\Big], \qquad (3.10)
\end{aligned}$$

注意$r(s,a,s')$表示期望奖励，当奖励是确定性的情况下，也就等于即时奖励（immediate reward）。该式叫做policy π的贝尔曼方程。贝尔曼方程对从当前状态起之后的所有可能性进行了平均化，并按照发生的概率给予不同的权重。

价值函数v_π是贝尔曼方程的唯一解。我们会在后几节讨论贝尔曼方程在value function的计算、近似和学习过程中的重要基础作用。我们在下面给出“反向图”的例子，这种图可以形象的给出value 是如何从后一个状态转移到前一个状态的。需要注意的是反向图中的状态并不一定是不同的，有可能某个状态即是当前状态，又是其后的状态。

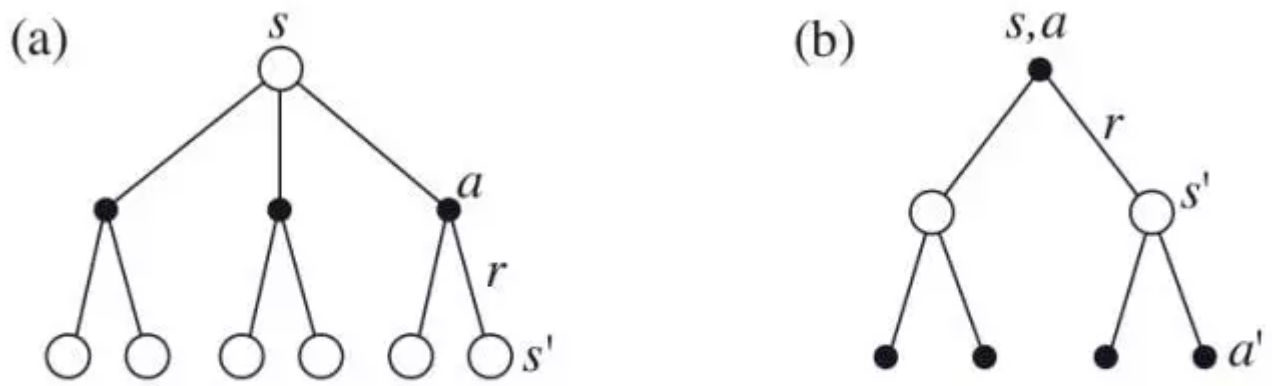

Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

3.8 Optimal value function

粗略地说，解决一个强化学习问题，也就是意味着找到一个policy，在这个policy下可以获得最好的长期期望奖励。对于finite MDP来说，我们可以用下面的方式精确的定义一个最优policy。如果我们说一个policy π 比另外一个policy π'更好，或者一样好，那么其实就是说，对于状态空间中的所有的状态s，按照前一个policy π 得到的$v_\pi(s)$都比后一个policy π' 得到的$v_{\pi'}(s)$要大（或者相等）。至少存在一个policy，使得它可以好于或者等于其他所有的policies，这个policy就叫做最优policy（optimal policy）。尽管有可能最优policy有多个，我们用π_*来表示。最优policies对应的value function是相同的，用v_*表示，定义如下：

$$v_*(s) = max_\pi v_\pi(s); for all s \in S$$

同样的，最优policies的最优action-value function也是相同的，用q_*表示，定义如下：

$$q_*(s,a) = max_\pi q_\pi(s,a);\ for all s \in S, a \in A(s)$$

另外，q_*和v_*之间的关系式如下：

$$q_*(s,a) = E[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]$$

因为v_*仍然是价值函数，因此，它也有对应的贝尔曼方程写法，叫做贝尔曼最优方程。贝尔曼最优方程的含义就是，在最优policy下某状态的value，一定等于当前状态下最优动作的期望返回值（in long run）。公式如下：

$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\ &= \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_{a \in A(s)} \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v_*(s')] \end{aligned}$$

注意后两行代表了两种写法。同样对于action-value function，贝尔曼最优方程如下：

$$\begin{aligned} q_*(s,a) &= E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a) | S_t = s, A_t = a] \\ &= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma \max_{a'} q_*(s', a')] \end{aligned}$$

对于贝尔曼最优方程来说，它的唯一解是独立于policy的。事实上，该方程是一系列等式，每一个对应一个状态，因此如果我们的状态空间有n个状态，那么就有n个等式，包含n个未知数。如果环境的动态性质由$r(s,a,s')$和$p(s'|s,a)$给出，那么理论上我们可以用任何解非线性方程的办法得到精确解。

一旦我们解出了v_*，那么我们就能很容易确定一个最优policy。对于每一个状态s,会有一个或者多个action可以使得贝尔曼方程中的最大值被取到。于是，任何把这些动作的概率赋值为正的policy，都可以是最优policy。如果你有了v_*，根据递归特性，那么任何使得在一步之后表现最好的action就是最优action。换句话说，任何“贪婪”的policy都是最优policy。“贪婪”这个词在计算机领域，通常指的是寻找局部最优解，而不考虑当前的选择是否在长期过程中是最优的。而v_*的优美之处恰恰就在此，看起来我们只是考虑了short-term（只是一步而已），然而贝尔曼最优方程却表明，当前的短期贪婪解，其实也就是长期最优解，因为v_*已经包含了未来reward的所有的可能性。通过v_*，我们把求长期的最优期望返回值问题转变成了求局部最优解的问题！问题被极大地简化了。

另外，如果我们解出了q_*，也就是action-value optimal function，那么问题就更简单了。我们甚至不需要做one-step search，也就是那一步之内的搜索都可以省了，我们直接就知道当前状态下哪个action使得$q_*(s,a)$。因此，使用这种表示方式，我们就可以在不知道任何后续状态或者环境动态特性的情况下，得到最优policy（当然，计算得到q_*的过程还是需要知道环境动态性质的，这里说的是一旦求解得到q_*之后，就不用管任何环境状态了）。

好了，说了那么多优点，我们要来说点缺点了。尽管显式地求解贝尔曼最优方程可以帮助我们求得最优policies，然而，这个方法直接用处并不太大。因为它涉及到穷尽所有的可能性。这个解法依赖于三个在实际中很难满足的假设：（1）我们精确的知道环境的动态性质；（2）我们有足够的计算资源去完成所有运算；（3）环境满足马尔科夫性质。这三个假设在我们感兴趣的问题中通常很难全部满足。比如对于西洋双陆棋来说，第一个和第三个假设可以满足，然而第二个假设很难满足，因为它有10^{20}个状态。因此，实际应用中，我们必须才去近似的方式。

有很多不同的决策方法可以看成是贝尔曼最优方程的近似。比如启发式搜索，可以看成是把贝尔曼最优方程等号右边部分扩展到一定深度，得到一个概率树，然后用启发式评估方法在叶子节点上进行v_*的近似（通常启发式搜索算法，比如A_*,大多都是基于片段式任务）。另外一个方法就是动态规划，它和贝尔曼方程的相似程度更加接近。很多强化学习的方法，都可以看成是对贝尔曼最优方程的近似求解，使用实际经历的转移情形来弥补难以完全得知环境动态性质的缺陷。我们会在后续章节陆续介绍几类方法。

3.9 Optimality and Approximation

我们已经定义了最优价值函数和最优policies。很明显，如果一个agent可以成功学到最优policy，那么它的表现肯定很好，但是实际情况往往不会发生。最大的阻碍应该就是计算资源的不足。

同时，有限的内存也是一个瓶颈。对于大型任务来说，我们通常没有足够的内存来保存每一个状态或者状态-动作的近似值。

听起来很令人沮丧，但是近似的思想在阻碍我们获得最优解的同时，也带给了我们一些机会，使得我们可以获得更有用的近似值。比如，对于一些状态来说，它们出现的可能性如此之低以至于就算我们在这些情形下做出了非最优解，对于最终的期望返回值的大小并没有太大影响。事实上，这些状态有时占据很大一部分比例。就拿之前提到的双陆棋来说吧，很大一部分的棋盘状态是不经常出现的，因此我们就可以损失在这些情况下做出较差决策的代价，来保证在经常出现的状态下做出更好的决策。这是强化学习方法区别于其他解决马尔科夫问题的近似方法的一个重要性质。

3.10 Summary

总结一下吧。强化学习，说白了就是一个学习在交互过程中如何做出决策，获得某个目标的问题。强化学习中，执行机构和环境之间不断交互作用。action是执行机构做出的决策（动作）；states是环境状态，是做出决策的基础；reward奖励是评估决策的基础。在执行机构之内的所有东西，都是完全可控、完全可知的；任何在执行机构之外，属于环境的东西，都是不完全可控、也不完全可知的。policy策略是状态和决策之间的函数映射关系。执行机构的目标是获得长远过程中的最大的reward。

return是未来reward的函数，也就是执行机构想要最大化的东西。它根据任务的不同可以有不同的定义。比如对于片段式的任务，undiscounted形式的return适用；对于连续性的任务，discounted形式的return适用。

环境如果满足马尔科夫性质，也就是环境中的每个状态都能完整保存历史信息，而不影响其预测未来状态的能力。换句话说，每个状态都独立于它的历史过程。满足马尔科夫性质的环境叫做马尔科夫过程（MDP）。有限马尔科夫状态（finite MDP）包含有限的状态集和动作集。大多数强化学习的理论或算法都需要马尔科夫过程的前提限制，但是其思想具有广泛通用性。

某个policy的value function，是基于当前的policy，来给每一个状态或者动作-状态对，赋予从那个状态开始一直到最终的期望return。最优value function则是赋予最大的期望return。如果基于某个policy的value function是最优的，那么这个policy也是最优的。尽管最优value function是唯一的，但是最优policy却不一定：因为任何一个基于最优value function的“贪婪”policy都是最优的。根据贝尔曼最优方程，理论上可以精确解出最优value function。

依据执行机构对外界环境的可知程度，可以有多种方式看待一个强化学习问题。在一个“完全已知”的问题中，执行机构可以有一个完全精确的环境动态模型。对应的，在“不完全已知”的问题中，一个精确的环境模型是不可能得到的。

即使我们有一个精确的环境模型，执行机构因为受限于计算资源和内存，也不太可能完全地利用它。因此，我们必须采用近似的方法。

一个完美的最优解的定义可以帮助我们理解理论性质。然而实际应用中我们只可能通过近似，来获得一定程度的准确。因此，在强化学习中，我们更关心当最优解无法获得因而必须做出一定程度的近似时，我们应该如何做。