增强学习导论

张会文
空间自动化研究室
沈阳自动化研究所，中国科学院

November 2, 2015

目录

Part I

问题

Chapter 2

评估式反馈

2.1 行为值方法

我们开始考察一些简单的用于估计行为值并且以此来进行行为选择的方法。在这章中，我们用$Q^*(a)$ 表示行为a的真实值，在t局时的估计值为$Q_t(a)$ 。回想一下行为的真值表示当行为被选择时获得的平均回报。一个自然的想法是通过行为被选择后获得的回报的平均值来估计行为值。换句话说t局之前行为a被选择了k_α 次，产生了$r_1, r_2, \ldots, r_{k_\alpha}$的回报，那么行为值被估计为：

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_\alpha}}{k_\alpha} \tag{2.1}$$

如果$k_\alpha = 0$,定义$Q_t(a)$是一些默认值，比如$Q_0(a) = 0$ 。当$k_\alpha \to \infty$时，有大数定律的到$Q_t(a)$收敛到$Q^*(a)$。我们称之为抽样均值方法。当然这只是估计值函数的一种简单方法，并不一定是最好的。尽管如此，让我们姑且停留在这个方法，并且转到如何利用值估计进行行为选择的问题。

最简单的选择规则是选择那个具有最高的行为值估计的行为，也即是在第t局中选择一个贪婪行为a^*，使得$Q_t(a^*) = \max_a Q_t(a)$。这种方法总是利用当前知识来最大化立即回报；而不考虑那些表面上次优的行为是否可能会更好。一个简单的替代策略是大部分时间按照贪婪算法选择行为，但是偶尔，比如以一个小的概率ε选择一个均匀的、独立于估计行为值的随机行为。我们称这种近似行为策略选择规则为：$\varepsilon - greedy$ 方法。这种方法的优点是，当局数无限增长时，每一个行为都会被抽样无限次，从而保证对于所有的a都有$Q_t(a)$收敛到$Q^*(a)$。这当然意味着选择最优行为的概率收敛到比$1 - \varepsilon$ 大的值，也即是基本稳定。但是这只是渐进的保证，对于方法的实际效果知之甚少。

为了粗略的评估贪婪算法和$\varepsilon - greedy$ 方法的相对有效性，我们通过一系列测试

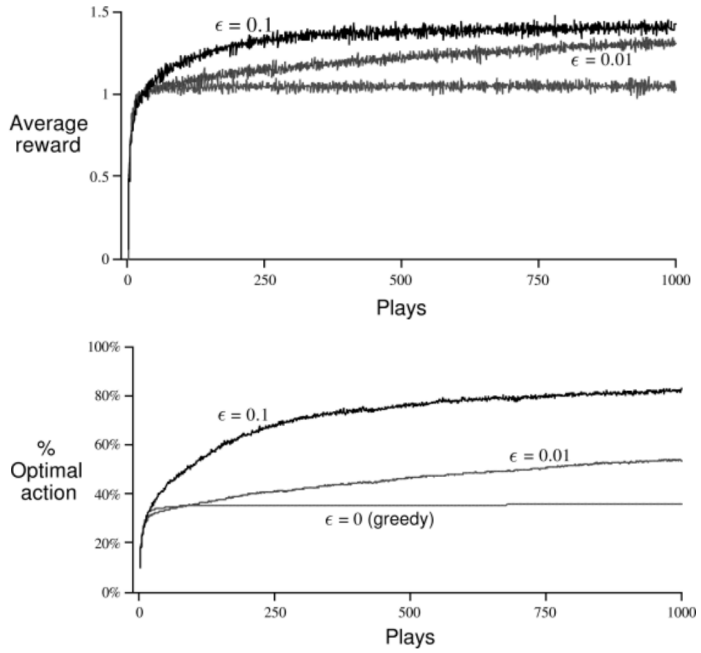


图 2.1: greedy和$\varepsilon - greedy$的表现

问题进行数值上的比较。这是一个2000次的随机产生的10个臂的赌博问题。对于每个动作a，回报值来自于一个均值为$Q^*(a)$方差为1的高斯分布函数。2000个赌博任务由$Q^*(a)$重新选择2000次产生，每次产生的值都服从标准高斯分布。对任务求均值，我们可以画出随着1000局经验积累后不同方法的表现，如图2.1所示，我们称这一套测试任务为10-臂测试平台。

图2.1比较了一种贪婪算法和2种$\varepsilon - greedy$算法，如上所示。2种方法都采用均值采样技术实现行为值估计。上图展示了期望回报随着经验增加。刚开始的时候贪婪算法的提升效果略微好于其他方法，但是之后稳定在一个低的水平。它每一步的收益仅仅是1，相比于最好的1.55。贪婪算法的长期表现很糟糕，因为他会陷入次优解。下图展示了贪婪算法在大概三分之一的任务中找到了最优行为。其他三分之二任务里，在刚开始最优行为的选择上是令人失望的，最终也没能逆转。$\varepsilon - greedy$算法最终表现更好因为它持续探索，提供了识别最优行为的可能。$\varepsilon = 0.1$的方法探索的概率大，因此通常会更早的找到最优值，但是仍然有9%的概率没有找到最优行为。$\varepsilon = 0.01$方法改善的速度慢，但是最终表现效果比前面好。

2.2 评估和指令的对比

上面考虑的n-臂赌博问题是纯粹靠评估反馈的一个例子。采取每个动作得到的回报值告诉了我们行为好坏的信息，但是对这个行为是否正确，它没有给出任何信息，

也即是这个行为到底是不是一个最优的行为。在这里，正确性是行为的一个相对属性，只有通过逐个的探索每个行为并比较获得的回报值才能决定最优行为。你不得不实施一些测试方法，凭借它产生动作、行为和结果，并且有选择性的保留那些最有效的行为。这是通过选择来学习，与通过教导的方式正好相反，所有的增强学习方法都必须以一种或某种方式使用它。

这和监督式学习截然不同，监督学习从环境获得的反馈明确的表示了那个行为是正确的。在这种情况下没必要进行搜索：无论你尝试那种行为，你都会被告知正确的行为应该是什么。你没必要尝试很多动作；教导式的反馈是典型的独立于行为选择的（所以事实上它不应该算是反馈）。在监督学习系统中，需要在参数空间搜索（例如：神经网络的权重），但是不需要在行为空间搜索。

当然，在某种程度上，监督学习通常应用在比$n-$臂赌博问题更加复杂的问题里。在监督学习里不是一个场景下的动作选择，而是大量的不同的情景，每一个情景都要做出正确反应。监督学习的问题是构造一个从场景到行为的映射，这个映射能够模仿具体环境下的的正确行为，并且对新的场景也能做出正确反应（泛化）。一个监督式学习系统不能被称作学习控制它的环境，因为它只是通过跟随接收到的指导信号，而不影响这个指导信号。它不是尝试让环境以某种方式变化，而是让自己按照环境的指示做出动作。

专注于重复的面临单个场景的特殊情况有助于简明的描述评估和教导的区别。假设有100 个可能的动作，你选择了32 号动作。评估式反馈针对选择的动作给你一个分数，比如7.2，然而教导式训练信息会其他哪一个动作会是正确的，比如67号动作。后者显然含有更丰富的训练信息。即使教导信号含有噪声，它仍然比评估式反馈信息量大。单个教导信号总是会直接的选择动作到一个更有的状态，然而评估式反馈必须要和其他动作进行比较才能做出抉择。

反馈式学习和教导式反馈的差别很大，即使只考虑2个行为和2个可能的回报。对于这些二值的赌博问题，我们称2个回报为成功和失败。如果你获得成功，那么你可能很合理的推断出你所选择的策略是正确的，如果你收到一个失败的信号，那么你可能推断出任何你没有选择的策略是正确的。你可以统计每个推断为正确动作的频率并且选择那些通常情况下正确的行为。我们称之为监督式算法，因为它很符合监督学习中对于单个输入模式的处理方式。如果回报是确定性的，那么监督算法的推断总是正确的并且表现良好。如果回报是随机的，那么情况就复杂多了。

在随机的情况下，一个特殊的二值赌博问题被定义成2个数，对应于每个可能的行为成功的概率。那么任务空间就是一个单位正方形，如下图所示，左上和右下象限表示相对容易的任务，这种情况监督式算法就能表现的很好。对于那些更好的行为成功的概率大于0.5，糟糕的行为成功的概率小于0.5。对于这些任务，那些被推断为正确的行为实际上超过一半的时间都是正确的。

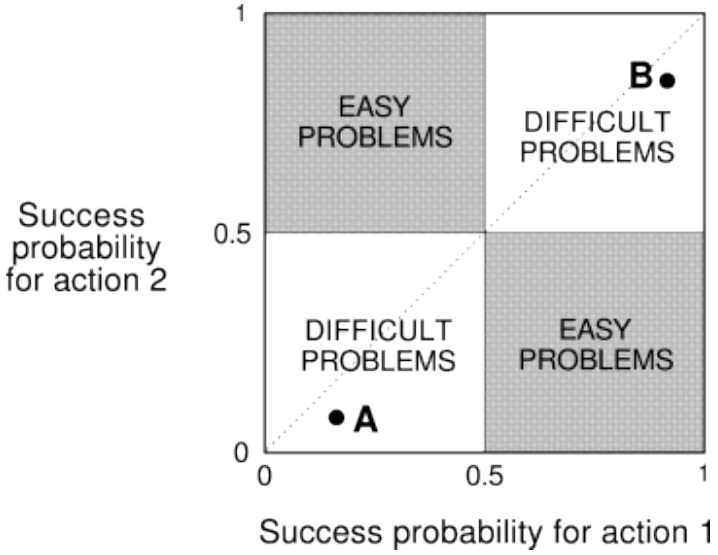


图 2.2: 二元赌博问题容易和复杂任务的的区域

但是，在图2.2 其他两个象限的二值赌博问题更加困难，用监督算法不能得到有效的解决。假如，考虑一个任务成功的概率是0.1或0.2，对应于图2.2 左下角困难的象限。因为两个行为都会至少产生80%的错误，任何一个以失败为指示，相应的另一个正确的方法都会在两个行为之间产生震荡，不会收敛到一个好的决策。现在考虑一个成功率为0.8或0.9的任务，对应于点B位于图2.2右上角的象限。在这种情况下两个行为总是会成功。任何以成功作为正确性评判的方法很容易陷入错误行为选择的死区。

图2.3展示了监督算法和其他几种算法在二值赌博问题上的平均行为。为了比较，同时用了2.2节的$\varepsilon-$ 贪婪行为值方法（$\varepsilon = 0.1$）。可以看出，在两个任务中，监督算法一半的时间能选择到好的行为。

图2.3也展示了2种方法的平均行为，被称作L_{R-I}和L_{R-P} 。这些方法都是来自于自主学习领域的经典方法，都遵循和监督算法相似的逻辑。2中方法都是随机的更新选择行为的概率，表示为$\pi_t(1)$ 和$\pi_t(2)$。L_{R-P} 方法推断正确行为的思路和监督算法相同，然后按照如下调整概率。如果在第t局中d_t被推断为正确的，那么$\pi_t(d_t)$ 增加一部分α ，表示当前值到1的距离：

$$\pi_{t+1}(d_t) = \pi_t(d_t) + \alpha[1 - \pi_t(d_t)] \tag{2.2}$$

另一个行为的概率按照相反的趋势调整，保证2概率和为1。图2.3中α 的值为0.1。L_{R-P}和监督算法的思想是一致的，只是他是随机的。他不是整体的对行为评判，L_{R-P}逐渐的增加他的概率。

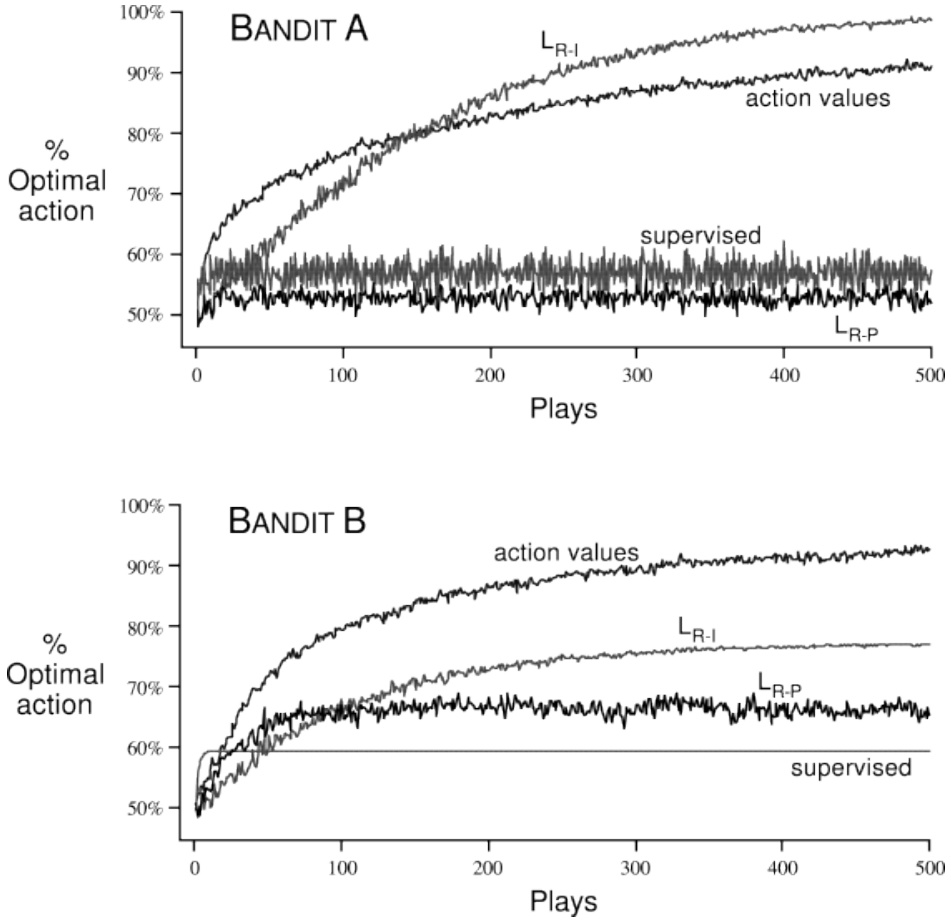


图 2.3: 选择算法在二值赌博问题上的表现，对应于图2.2的A和B

L_{R-P} 表示线性的，奖励惩罚，意味着式2.2的概率是线性的，并且同时对于输赢的局进行更新。L_{R-I} 表示线性的回报延迟（inaction）。这个方法和L_{R-P}相同除了它只利用赢的局更新概率，输的局全部被忽略。图2.3 表明L_{R-P}在二值赌博问题上表现比监督算法稍稍好一点，无论是A还是B。L_{R-I}在A点表现很好，但是在B上表现不好，并且在两种情况下学习速度都很慢。

二值赌博问题是一个很有启示性的例子，混合了监督学习和增强学习问题的特性。因为回报是二值的，使得只通过单个的回报来推断正确的行为成为可能。在一些类似问题的实例中，这些推断相当合理并且衍生出有效的算法。但是在其他情况下，这种推断就不合适并且表现糟糕。对于不是二值回报的赌博问题，比如一个10-臂的测试台，这些推断的思想如何被用于产生有效的算法不十分清楚。所有这些问题都是很简单的例子，但是我们已经看到了对于监督学习算法之外的需求。

2.3 增量式实现

目前为止我们讨论的行为值方法，对于值的估计都是通过观测回报的平均值实现的。对于每个动作a ，最简单的实现是记录选择a 后跟随的回报值。然后，t 时刻a行为的值函数可以通过式2.3 计算：

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_\alpha}}{k_\alpha} \tag{2.3}$$

$r_1, \ldots, r_{k_\alpha}$ 是前t 局游戏中选择行为a后获得的回报值。一个通过这种直接的方式实现的系统，它的内存和计算需求会随着时间无限增长。也即是选择a 后每一个附加的回报值都需要内存来存储并且为了决定$Q_t(a)$ 需要更多的计算。

也许你会怀疑，这并不是必要的。设计增量式的、常数阶的计算量的更新公式计算每次新回报后的平均值是很容易实现的。对某个行为，Q_k 表示k 次回报的平均值（不要和$Q_k(a)$ 混淆，他表示前k局中动作a的平均值）。鉴于这个定义，第$k+1$次的回报为为r_{k+1},那么所有$k+1$次回报的平均值可以计算为：

$$\begin{aligned} Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\ &= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k} r_i\right) \\ &= \frac{1}{k+1}\left(r_{k+1} + kQ_k + Q_k - Q_k\right) \\ &= \frac{1}{k+1}\left(r_{k+1} + (k+1)Q_k - Q_k\right) \\ &= Q_k + \frac{1}{k+1}\Big[r_{k+1} - Q_k\Big] \end{aligned} \tag{2.4}$$

当$k = 0$时，对于任意的Q_0都有$Q_1 = r_1$。这种实现只需要内存记录Q_k和k,对于式2.4每一个新的回报值计算量都很小。

式2.4的更新规则是本书中频繁出现的一种形式。通用形式是：

$$NewEstimate \longleftarrow OldEstimate + StepSize\Big[Target - OldEstimate\Big] \tag{2.5}$$

表达式$[Target - OldEstimate]$ 是估计中的误差。它通过趋向于目标而减少。目标表示一个假定的理想的移动方向，尽管它可能含有噪声。例如对于上面的例子，目标就是第$k+1$次的回报。

注意上述增量方法中的步长参数每一步时不同的。在处理动作a的第K次回报中，步长参数是$\frac{1}{k}$。在这本书中用符号α表示步长，或者更一般的情况用$\alpha_k(a)$。例如，上述抽样均值方法中增量实现的步长表示为$\alpha_k(a) = \frac{1}{k_\alpha}$。相应的，有时我们简写为$\alpha = \frac{1}{k}$ ，动作可以通过隐含的推出。

2.4 追踪一个非稳态的问题

目前为止讨论的平均值方法适用于一个稳态的环境，但是如果赌博随时间变化呢。正如前面所述，通常我们面临的增强学习问题是非稳态的。在这种情况下，给最近得回报一个大的权值是有意义的。解决这一问题最有名的方式是用一个常数的步长参数。例如，式2.4用于对过去k次回报均值进行更新的增量式更新策略被修正为：

$$Q_{k+1} = Q_k + \alpha\Big[r_{k+1} - Q_k\Big] \tag{2.6}$$

其中步长参数是一个常数，$0 < \alpha \leq$ 。这导致Q_k 成为过去回报的带权重的均

值，初始状态为Q_0 ：

$$\begin{aligned} Q_k &= Q_{k-1} + \alpha\big[r_k - Q_{k-1}\big] \\ &= \alpha r_k + (1-\alpha)Q_{k-1} \\ &= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} \\ &= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 \alpha r_{k-2} + \\ &\cdots + (1-\alpha)^{k-1}\alpha r_1 + (1-\alpha)^k Q_0 \\ &= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i \end{aligned} \tag{2.7}$$

我们称之为带权重的平均值，因为这些权值的和是：$(1-\alpha)^k + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} = 1$ ，你可以自己验算。注意哪些权重，$\alpha(1-\alpha)^{k-i}$,鉴于回报r_i依赖于多久之前回报被观测到。数值$1-\alpha$ 小于1，因此随着间隔回报个数的增加，对应到r_i 的权值减小。实际上，权值依据$1-\alpha$ 以指数级衰减。相应的，有事称之为指数的新近-权重均值。

有时候每一步都改变步长参数是方便的。用$\alpha_k(a)$ 表示在经过k 次选择a后处理回报的步长参数。正如我们说过，对于均值抽样方法$\alpha_k(a) = \frac{1}{k}$ ，通过大数定律能保证该方法能收敛到真正的行为值。当然对于任意的$\{\alpha_k(a)\}$序列收敛性是无法保证的。在随机近似理论中一个著名的结论保证了以概率1收敛的必要条件：

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \qquad and \qquad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty \tag{2.8}$$

第一个条件用来保证步数足够大，以至于最终能客服任何的初始条件和随机波动。第二个条件用来保证最终的步长足够小以致收敛。

考虑到均值采样的情况同时满足两个条件，但是对于常数步长的$\alpha_k(a) = \alpha$不是。后者不满足第二个条件，意味着估计值不会完全收敛，而是随着最新接受的回报值持续变化。正如我们上面提到的，这在非稳态环境中是可取的，并且实际上非稳态问题是增强学习里的规范。除此之外，满足式2.8 条件的步长参数序列通常收敛速度很慢，除非进行相当大的调整才能得到一个满意的收敛速率。尽管在理论工作中会考虑步长参数的收敛条件，但是很少用在实际应用和实践研究里。

2.5 初始值优化

目前为止我们讨论的所有方法某种程度上都依赖于初始行为值$Q_0(a)$的选择。用统计

的语言来说就是这些方法适有偏的。对于均值采样方法，当所有行为至少被选择一次的时候偏差会消失，但是对于步长为常数α的情况，偏差始终存在，即使如2.7给出的偏差会随着时间逐渐减小。在实际中，这种偏差通常不是一个问题，相反有时候非常有益。不好的地方是初始值估计变成了一个参数集，这些参数由用户来指定，如果只把他们全部设为0。好处是它提供了一个简单的方式，使我们能利用先验知识决定期望回报能达到某个水平。

初始行为值同时也提供了一种探索的简单方式。比如对于10-臂的试验台，我们替换原来的初始值0，把他们全部设为+5. 回顾在这个问题中，$Q^*(a)$ 选择为均值为0方差为1的正态分布。+5的初始值因此表现良好。这种优化鼓励行为值方法探索。无论哪个行为被初始选择，回报总是小于开始的估计；学习器变换到另一个行为，因为对于它收到的回报感到“失望”。结果是收敛之前所有的行为被尝试若干次。系统做了相当数量的探索，即使总是选择贪婪的行为。

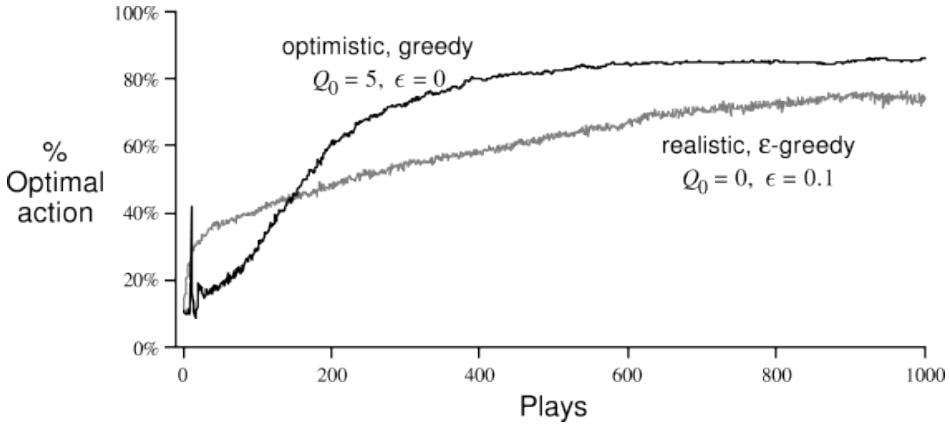


图 2.4: 10-armed testbed最优初始值估计的效果

图2.4展示了对于一个10个臂的赌博问题用初始值$Q_0(a) = +5$ 的一个贪婪算法。为了比较，同时使用了$\varepsilon - greedy$ 算法初始值$Q_0(a) = 0$。两种方法都适用固定步长$\alpha = 0.1$.刚开始，优化方法表现糟糕因为更多的探索，最终它的表现更好因为随着时间推移探索减少。我们称这种鼓励探索的技术叫做：最优初始值。我们称它为一种简单的能够有效处理静态问题的技巧，但是它远不是一种通用的有效的鼓励探索的方法。例如，它不适合于静态问题因为驱动它探索的动力本质上是暂时性的。如果任务变了，有了新的探索需求，这种方法就失效了。实际上，任何以特种方式专注于初始值的方法都不大可能适用于通用的非静态环境。开始的时间只有一次，因此我不应该过分专注于它。这个诟病也适用于均值抽样方法，该方法也将初始时间视为一个特殊事件，然后以相等权值平均后续的回报。尽管如此，但是所有这些方法相当简单，他们当中的一个或者几种简单方法的集合在实际中也足够了。在剩下的内容中我们将频繁使用这几种简单的探索技术。

2.6 增强比较

增强学习的一个核心的想法是：那些获得大的回报的行为应该被赋予更大的重复概率，相反回报小的行为出现的概率要小。但是学习器怎么知道是什么构成了一个大的或者小的回报？如果一个动作被选择了，环境返回一个大小为5的回报值，这到底是大还是小？为了做这样一个判断，用户必须把这个回报和某个标准或者参考水平比较，这个参考值叫做参考回报。一个自然的选择是参考回报取之前得到回报的平均值。换句话说，一个回报值被认为是大的如果它高于平均值，反之小如果低于平均值。基于这种思想的学习方法叫做增强比较方法。这个方法有时候比行为值方法更加有效。他也是行为−−评判方法的前身，这是我们后面会讲道的解决所有增强学习问题的一类方法。

增强学习通常不记录行为值的估计，而是维护一个总体的回报水平。为了在行为之中进行选择，系统会维护一个针对每个行为的偏好度。我们用$p_t(a)$表示在t中对行为a的偏好。根据sofrmax关系，偏好可以用来决定行为选择的概率，例如：

$$\pi_t(a) = Pr\{a_t = a\} = \frac{e^{p_t(a)}}{\sum_{b=1}^{n} e^{p_t(b)}} \tag{2.9}$$

$\pi_t(a)$表示在t局中选择行为a的概率。增强比较思想被用来更新行为偏好。每局之后，在当前局中被选择的行为a_t按照回报r_t和参考回报$\bar{r}_t$的差值进行更新：

$$p_{t+1}(a_t) = p_t(a_t) + \beta\big[r_t - \bar{r}_t\big] \tag{2.10}$$

β 是一个正的步长参数。这个方程暗含了高回报行为的选择概率应该被增加，低回报行为选择概率减小的思想。参考回报是所有最近接受回报的增量平均，无论哪个动作。经过2.10更新后，参考回报被更新为：

$$\bar{r}_{t+1} = \bar{r}_t + \alpha\big[r_t - \bar{r}_t\big] \tag{2.11}$$

这里，像以前一样$\alpha, 0 < \alpha \leq$是个步长参数。参考回报$\bar{r}_0$的初始值可以被设置为一个乐观的值以鼓励探索，或者根据先验知识。初始行为参考值可以被设为0.因为回报分布随着行为选择的改善而变化，所以常数α是一个好的选择。我们这里考察实际上非静态的学习问题，尽管潜在的问题是静态的。

增强比较方法很有效果，有时比行为值方法表现更好。

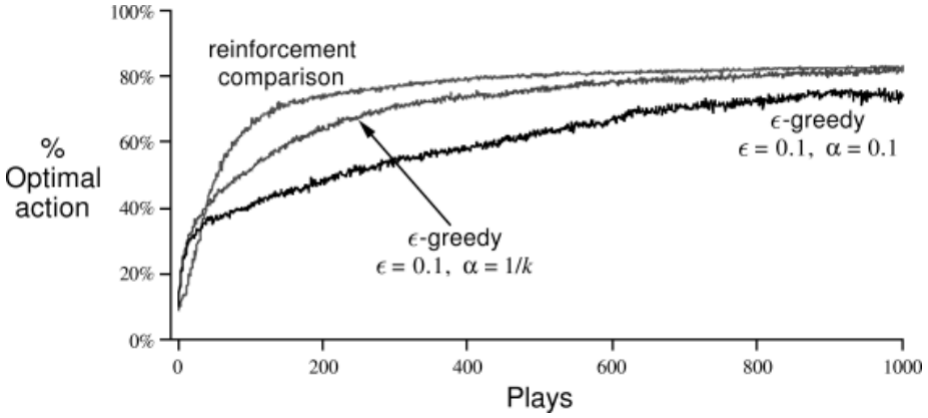


图 2.5: 10-armed testbed上增强比较方法和行为值方法对比

2.7 追赶方法

对于$n-$臂赌博问题，另一类有效的学习方法适追赶学习方法。该方法同时修正行为值估计和行为偏好，行为偏好持续“追赶”贪婪行为，根据目前的行为估计值。在这个最简单的追赶方法中，行为偏好是概率$\pi_t(a)$，通过进行第t局的行为选择。每局过后，概率值被更新以使得贪婪行为更加可能被选择。用$a^*_{t+1} = \arg max_a Q_{t+1}(a)$表示$t+1$局的贪婪行为（或者随机抽取一个行为值，当存在好几个行为值相等）。然后选择$a_{t+1} = a^*_{t+1}$被增加和1做差的部分：

$$\pi_{t+1}\left(a^*_{t+1}\right) = \pi_t(a^*_{t+1}) + \beta\left[1 - \pi_t(a^*_{t+1})\right] \tag{2.12}$$

相反其他的行为选择概率被减小，如下式：

$$\pi_{t+1}\left(a\right) = \pi_t(a) + \beta\left[0 - \pi_t(a)\right], \qquad for\ all \quad a \neq a^*_{t+1} \tag{2.13}$$

值函数$Q_{t+1}(a)$的更新可以用前面讨论过的方法，比如观测回报的抽样平均，式2.1所示。

图2.6展示了上述追赶算法的表现，当行为值估计采用均值采样的情况（$\alpha = \frac{1}{k}$）.这些结果中，所有行为a初始概率取为$\pi_0(a) = \frac{1}{n}, \beta = 0.01$ 。

2.8 联想搜索

目前的章节中我们只考虑了非联系的任务，这种情况下没必要联系不同情景下的不同行为。在这些任务中，当任务时静态的时候，学习器要么学习一个最优行为，要么当任务是非静态的时候，学习器追踪一个随时间变化的最优行为。但是在通常的

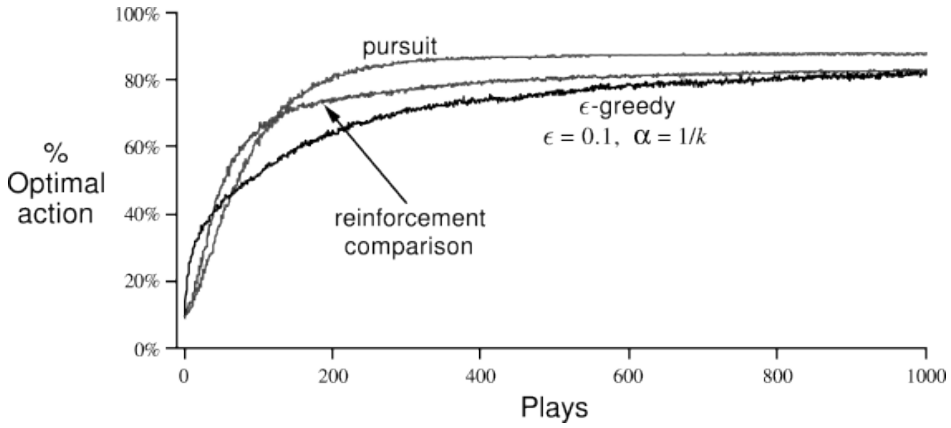


图 2.6: 追赶方法和其他几个方法的比较

增强学习问题中不止一个场景，并且目标是学习一个策略：一个从状态到最优行为的映射。为了全盘考虑问题，我们简洁的讨论一个恶最简单的方法能够把非联系任务扩展到联系的情况。

作为一个例子，假设有几个不同的n臂赌博问题，在每一局中随机的处理当中的一个。因此赌博问题在没局之间随机变化。你将面临一个单个的、非静态的赌博问题，该任务中真值随机变化。你可以使用本章中前面讨论能够处理非稳态问题中的某种方法，但是除非真值变化缓慢，否则这些方法效果都不好。但是假设你选择了一个赌博任务，你会得到一些这个任务的显著性的特性。或许你正面临一台老虎机，这个老虎机能够随着行为值的变化改变他的表现颜色。现在你可以通过颜色信号来学习一个联系每个任务的策略。当面临那个任务时，例如如果是红色，用1号臂，如果是绿色，用2号臂。使用正确的策略，在没有信息辨认不同赌博任务的情况下，通常你会做得很好。

这是一个联系搜索任务的例子，之所以这么说是因为它涉及了在最优行为搜索的试错学习和最对状态的最优行为的联系。联系搜索任务介于n-臂赌博问题和增强学习问题之间。它像是增强学习问题因为它引入了策略学习，但是鉴于每个行为只影响立即回报，这一点更像赌博问题。如果行为会影响下一个状态和回报，那就是一个增强学习问题。我们在下一章中讨论这个问题，并且通过剩余的章节讨论他的分支。

2.9 总结

在本章中我们展示了一些平衡探索和利用的简单方法。利用$\varepsilon - greedy$算法在部分时间进行随机抽取，softmax方法通过当前的估计行为值给行为发生概率分等级，追赶方法保持每一步的动词趋向于当前的贪婪行为。对于实际有用的算法而言这些简单

的方法真是我们能做的最好选择吗？目前为止，答案似乎是“Yes”。尽管他们很简洁，但是在我的观念里这些方法都能称为时当前最好的。有一些其他的更高级的方法，但是他们的复杂性和假设使得他们在我们实际关注的整个增强学习问题中不实用。第5章开始我们展示一些部分的用本章简单的方法解决的增强学习问题。

尽管本章探索的方法是目前为止最好的，但是他们远不是解决探索和利用平衡问题的满意方法。我们简洁的总结一下本章目前的一些思想，虽然还不十分实用，但是它会引导我们找到风阿红的解决方案。

一个很有用的想法是利用行为值估计的不确定性指导和鼓励探索。例如假设有2个行为，他们的估计值略小于那个贪婪的行为的估计值，但是他们的不确定度差异很大。一个估计接近确定；或许是那个行为已经被尝试了很多次然后观测到了很多的回报。这个行为估计值的不确定性如此之低，以至于他的真值不大可能高于贪婪行为的值。另一个行为了解的并不多，值估计不确定。这个行为的真值很有可能会好于贪婪行为。显然的，探索第二个行为比第一个更有意义。

这种思想导致了区间估计方法。这些方法为每一个行为值估计一个置信区间。也即是说我们不学习说一个行为值接近10，相反他们以95%的置信度说它位于9和11之间。然后那个具有更高置信上限的行为被选择。这种方法鼓励那些不确定但是有机会成为最佳行为的行为的探索。一些情况可以保证最优的行为应该在95%置信区间之内。不幸的是由于被用于区间估计的统计方法的复杂性使得该方法在实际中存在问题。除此之外，这些方法潜在的统计假设通常也不能满足。尽管如此，用置信区间的思想，或者其他不确定度度量方法来鼓励特殊行为的探索还是很合理并且引人入胜的。

也有众所周知的利用贝叶斯优化来平衡探索和利用的算法。实际做起来这个方法的计算很棘手，但是可能会有高效的方法来接近它。这个方法中我们假设我们已知问题实例的分布，也就是真是行为值集合的概率。给定任何的行为选择，我们可以计算出每个可能立即回报的概率和行为值的联合后验概率分布。演化之后的分布变为问题的信息状态。给定一个范围，比如说1000局。我们可以考虑所有可能的行为，所有可能的回报，所有可能的下一步动作，所有下一步的回报等等。基于假设，回报和每个可能的行为链的概率可以被确定，我们只需要挑出最好的。但是概率树增长的相当快；即使只有2个动作2个回报，将会有2^{2000}个叶子节点。这个方法很有效的将赌博问题变成了一个增强学习问题。最后，我们可能会用增强学习方法来近似这个最优解。但是这是当前的一个话题并且超出了这本书的范围。

经典的平衡n-臂赌博机中探索和利用的方法是计算一个特殊的函数叫做*Gittins indices*。这提供了赌博问题更通用的一种解决方案，但是它假设问题的先验分布是已知的。不幸的是，无论是这个理论还是他的计算适用性都没能泛化到我们后面将要讨论的增强学习问题。

Chapter 3

增强学习问题

3.1 智能体-环境接口

增强学习方法描述了以与环境交互获得目标的广泛的一个框架。我们把学习机器和决策者叫做一个智能体。与他交互的东西，包括所有智能体之外的东西叫做环境。

增强学习问题的描述，离散时间序列$t = 0, 1, 2, 3, \ldots$，在每个时间步长t,智能体获得环境状态的表示，$s_t \in \mathcal{S}$，$\mathcal{S}$表示所有可能状态的集合，基于这个状态选择一个行为，$a_t \in \mathcal{A}(s_t)$ 是当前状态s_t可选行为的集合。一个时间步长之后，某种程度上作为行为的后果，智能体接收到一个数字的回报 $r_{t+1} \in \mathfrak{R}$，然后到达一个新的状态 s_{t+1}.图3.1表述了这种交互：

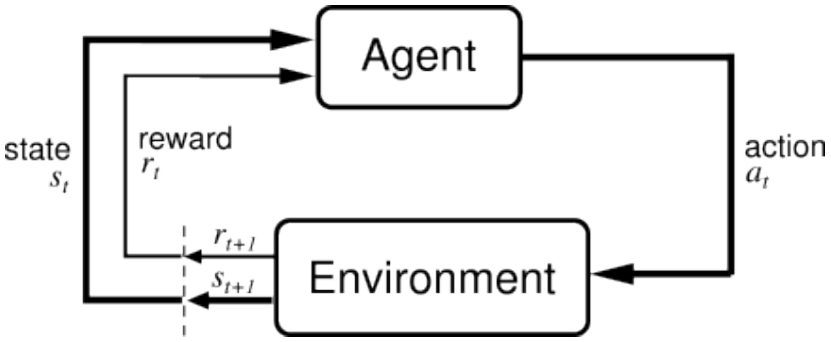


图 3.1: 增强学习中的智能体–环境交互

把从状态到选择行为概率的映射称为智能体的*policy*即策略或者政策，用π_t表示。增强学习方法阐述了智能体如何根据他的经验改变策略。目标是最大化回报量。

这个框架具有概括性和灵活性能够以不同的方法应用到很多不同的问题。比如，时间不长不必要是时间时间的一个固定的区间；他们可以指任意的连续的决策和行为阶段。行为可以是低层次的控制，比如应用到机械臂的马达的电压，或者高

层次的决策，比如是否吃午餐或者去学校。相似的，状态亦可以是非常广泛的形式。他们可以完全由低层次的感知决定，比如直接的传感器读数，或者也可以是高层次的、抽象的，比如室内物体的象征性描述。状态的组成可以是过去感知的记忆，也可以完全是精神上的、主观的。例如一个智能体可以处于不确定物体位置的状态，或者被一些清楚定义的感觉惊讶。相似的，一些行为可能完全是精神上的或者计算的。例如一些行为能够控制智能体选择思考的内容，或者它应该专注的地方。通常意义上，行为可以使任何我们想要学习所做的决定，而行为可以是任我我们知道在做决定时可能有用的东西。

特别的，智能体和环境的界限通常和机器人或者动物躯体的物理界限不同。通常界限更加接近智能体。例如机器人的马达、机械连杆以及他的硬件传感器应该被当做环境的一部分而不是智能体的一部分。相似的，如果我们把这个框架用在人或者动物身上，肌肉、骨骼和传感器官应该被认为是环境的一部分。同样的，回报假定是自然体或者人工学习系统内部计算的，但是也被当做智能体的外部环境。

通常的规则是任何不能被智能体任意改变的东西被称作是外部的，因此是环境的一部分。我们不假设环境中所有的东西对智能体都是未知的。例如智能体通常会知道回报函数作为选择的行为和当前状态的函数，应该如何被计算。但是回报的计算被认为是外部的因为它定义了智能体面临的任务，因此随意改变它超出了智能体的能力之外。实际上，有时候智能体了解了它周围环境机制的一切但是仍然面临困难的增强学习任务，正如我们完全知道拼图游戏例如鲁比克魔方的工作机理，但是仍然没办法觉得它。智能体-环境界限表示了智能体绝对控制的极限，而不是他的知识。

……

增强学习被认为是通过交互学习的目标导向问题的可观的学习框架。无论具体的问题是什么，问题都可以被简化为智能体和环境之间三个信号的来回传递：1）智能体做的决策（action），2）决策制定基于的信息（state），3）定义智能体的目标（reward）.

- **生物反应器：**行为是传递给底层控制系统的目标温度和搅拌率；状态可能是热电偶和其他的传感器读数；回报是反应器产生的有用的化学物质的速率。

- **拾取放置机器人：**行为是加在每个关节的马达的电压；状态是最近得关节角度和速度的读数；回报是1如果成功完成拾取和放置。

- **资源回收机器人:**行为包括主动搜索罐子，保持静止等待某人带来罐子，或者回去充电；状态是当前时刻的电量；回报大多时候是0，当获得罐子变为正的。

3.2 目标和回报

增强学习的目标由环境传递的回报信号构成。每个时间步长，回报是一个数字$t_t \in \mathfrak{R}$,不太正式的说法就是智能体的目标是最大化获得的总回报。这意味不是最大化立即回报，而是长时间的累积回报。

3.3 回报

假设时间步长t之后智能体获得的序列回报表示为：$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$，那么如何精确的定义回报呢，也即是我们要最大化的那个东西，通常回报R_t定义为回报序列的一个具体函数。最简单的方式是回报之和：

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \tag{3.1}$$

T是终止时间步长。终止时间步长有一个很自然的解释，也就是我们把智能体-环境交互分成几个子系列，叫做一个情节，比如一个游戏的一局，走迷宫或者任何重复形式的交互。这种具有情节的任务叫做插话式任务。所有状态加上终止状态表示为 $mathcarS^+$.

当一个任务时无限的或者不能分成片段的时候，我们叫做连续任务.

另外一个概念是折扣。智能体选择动作使得折扣回报最大化。比如：

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.2}$$

γ是个参数，$0 <= \gamma <= 1$,叫做折扣率或者折扣系数。

参数γ反映了智能体考虑的是否长远。当$\gamma = 0$的时候，回报只和r_{t+1}有关，也就是立即回报。这种情况下我们说智能体是目光短浅的。但是当智能体的每个动作只影响下一个动作的时候，我们可以通过最大化立即回报来最大化式3.2.当γ接近1的时候，说明了目标对未来考虑的越多，更有远见。

3.4 片段式和连续式任务的统一描述

如前面所述，增强学习有两类不同的任务。其中一种可以分成多个片段，而另一种是连续的。实际中有时考虑第一种，有时考虑第二种，但是通常情况下是二者皆有。因此建立一个统一的能够同时适用于2种任务的描述是有必要的。

我们在式3.1定义了一个有限项的和来表示回报，在式3.2定义了一个无限项的和

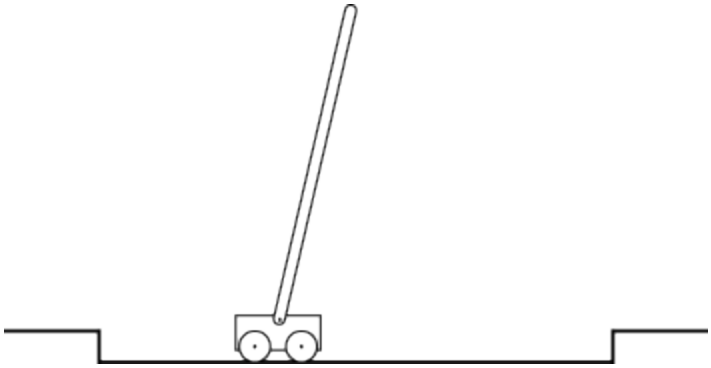
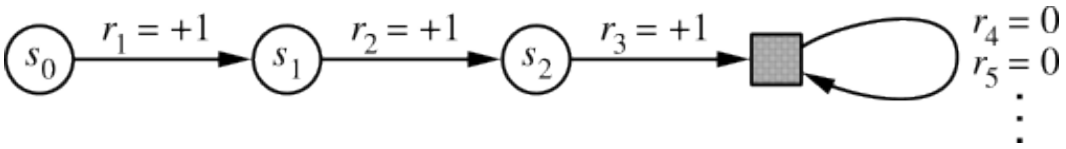


图 3.2: 倒立摆问题

来表示连续任务。怎么从有限到无限，我们可以设想有限项的终止状态是进入了一个特别的吸收状态，这个状态只能过渡到自己本身，并且只产生0的回报。考虑如下的状态转化图：



如上图，方框表示吸收状态。从s_0开始那么回报序列表示为 $+1, +1, +1, 0, 0, 0, \ldots$。那么有限的序列就可以看成是后面有无限个0的无限序列，那么就可以统一成无限的形式，如下所示：

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{3.3}$$

当$T = \infty$就是式3.2,当$\gamma = 1$就是式3.1,但它和二者都不同。剩下书本中都用这个表示。

3.5 马尔科夫性

在增强学习框架中，智能体依据状态采取决策。这节讨论状态信号需要什么，我们应该希望和不希望状态提供哪些信息。特别的我们考虑状态的一个性质叫马尔科夫性。

这本书中状态指一切智能体能够获得的信息。当然状态信号应该包括立即感知比如传感的测量，但是不仅这些。状态表示是原始信号高级处理的结果，或者他们是从感知序列建立的复杂结构。比如人眼看物体只需要一瞬间，但是却建立了场景丰富详细的描述。总之，状态的建立可以基于立即感知，同时伴有先前的状态或者

过去的一些感知的记忆。

另一方面，我们不应该期望状态能够告诉我们环境的所有信息，甚至是所有的对决策有用的信息。

理想情况下，我们需要的状态信号是能够简介的概括过去的感知，以便所有相关信息能够被保留。这通常需要的不仅仅是立即的感知，也绝不是所有过去感知的完整历史。一个能够成功保留所有相关信息的状态信号被称作是马尔科夫的（*Markov*）。比如西洋棋的位置——当前棋局上所有棋子的位置——是马尔科夫的因为它概括所有导致该局面的重要的位置序列。肯多序列信息丢失了，但是对于未来重要的信息保留了下来。相似的炮弹的未来的航行只和当前的位置和速度有关。这种特性有时候被称为“路径无关”。这里的*path*指信号的历史信息。

用数学描述这个性质。简单起见，考虑有限个状态的情形。这样就可以用求和和概率而不是积分和概率密度来分析。当然是可以扩展到连续的情况的。考虑一个通常环境中智能体如何在$t+1$时刻对t时刻采取的行为做出反应。考虑因果关系，自然的想法是这个反应不仅依据当前行为，还包括先前已经发生的行为。如下：

$$P_r\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} \tag{3.4}$$

针对上式，如果说一个状态信号具有马尔科夫性，换句话说也就是智能体在$t+1$时刻的反应只取决于在td的行为，在这种情况下环境的动态性可以表述为：

$$P_r\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\} \tag{3.5}$$

上式对于所有的s', r, s_t, a_t都成立。一个状态信号具有马尔科夫性，被叫做马尔科夫状态当且仅当3.4和式3.5相等。

- **平衡杆状态：**在该问题中一个状态是马尔科夫的如果状态信号能够精确的获取，这些状态包括小车的位置和速度以及杆相对于小车的角度和角速度。理想情况下这是一个马尔科夫信号。实际上由于传感器失真、延迟，或者其他的影响，比如杆的弯曲、车轮温度、杆的轴承以及各种形式的反冲，都会影响系统的行为。这些因素违背了马尔科夫性。

  但是位置和速度状态通常表现良好。甚至用一个粗糙的状态把状态位置分为左中右三个。这种情况下显然是非马尔科夫的，但是问题依然效果良好。实际上有可能是粗糙的表示使状态较小促进了学习机的学习速度和忽略无用的信息。

- **抽牌，暗扑克：**In draw poker, each player is dealt a hand of five cards. There is a round of betting, in which each player exchanges some of his cards for new ones, and then there is a final round of betting. At each round, each player must

match or exceed the highest bets of the other players, or else drop out (fold). After the second round of betting, the player with the best hand who has not folded is the winner and collects all the bets.

3.6 马尔科夫决策过程

一个增强学习任务如果满足马尔科夫性则被称作马尔科夫决策过程，或者*MDP*。如果状态和行为是有限的，叫做有限马尔科夫决策*finite MDP*. Finete MDPs在增强学习理论中相当重要。通过它你可以理解现在增强学习中90% 的东西。

给定任何的状态s和行为a，下一步可能的状态$s\prime$的概率为：

$$\mathcal{P}_{ss'}^{a} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{3.6}$$

这些数字被称为转移概率.相似的，给定任何的状态s和行为a，下一步可能的状态s'，下一次的期望回报值为：

$$\mathcal{R}_{ss'}^{a} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{3.7}$$

回收机器人的例子

状态是电池的电量，有俩个状态，$\mathcal{S} = \{high, low\}$.

行为action有3个，分别是主动搜搜、等待和回去充电。对应每个状态的可选行为如下：

$$\mathcal{A}(high) = \{search, wait\}$$

$$\mathcal{A}(low) = \{search, wait, recharge\}$$

可见当电量高的时候回去充电显然是不合理的，所以这个状态下没有充电的行为。

给出状态转移表以及下一步的期望回报如图：

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
high	high	search	α	$\mathcal{R}^{\text{search}}$
high	low	search	$1-\alpha$	$\mathcal{R}^{\text{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$\mathcal{R}^{\text{search}}$
high	high	wait	1	$\mathcal{R}^{\text{wait}}$
high	low	wait	0	$\mathcal{R}^{\text{wait}}$
low	high	wait	0	$\mathcal{R}^{\text{wait}}$
low	low	wait	1	$\mathcal{R}^{\text{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

状态转移表

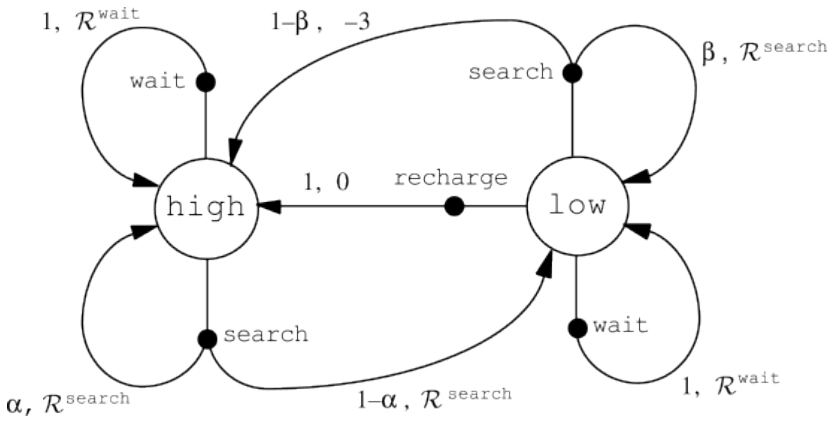


图 3.3: 状态转移图的例子

注意这里从某个状态出去采取同一行为后转移到不同状态的概率之和应该是1.这符合概率的性质。

3.7 值函数

几乎所有的增强学习方法都基于值函数——状态的函数（状态动作对的函数）——的估计。值函数表示了智能体在以给定状态下能够表现多好，或者给定状态下采取某个行为的好坏。这里的好坏用未来的期望回报表示。而回报和采取的策略有关，所有值函数的定义和具体的策略有关。

回忆下，策略π 表示对于每个状态和动作到采取某个行为的概率$\pi(s,a)$的映射，也就是把状态行为对映射成一个数值，这个数值表示在状态s下采取a的概率。非正式的，某个策略π下状态s的值函数表示为$V^{\pi}(s)$, 他是从状态s之后的的期望回报，表示为：

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s\right\} \tag{3.8}$$

$V^{\pi}(s)$叫做策略π的状态值函数。

相似的，定义在状态 s采取动作 a的价值表示为$Q^{\pi}(s,a)$,从状态s开始，后面的遵从策略π获得的期望价值：

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s, a_t = a\right\} \tag{3.9}$$

Q^{π} 叫做策略π的行为值函数。 V^{π}和Q^{π}都可以通过经验估计。这二者的区别是前者是一状态为前提，从这一个状态出去可能有多个行为，每个行为选定后又可能到达多个状态。后者限定了s下的动作a之后能到达状态的累积回报，当状态出现次数无限大的时候，回报的平均值收敛于期望。所有我们可以用抽样的方法估计值函数，称为蒙特卡罗方法。第五章会讲到。状态无限的时候，计算这些平均值不符合实际的，所有用参数方程表示，所谓的值函数逼近，在第8章讲到。

值函数的一个基本性质是满足迭代：

$$\begin{aligned}
V^{\pi}(s) =& E_{\pi}\left\{R_t \mid s_t = s\right\} \\
=& E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s\right\} \\
=& E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\middle| s_t = s\right\} \\
=& \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\middle| s_{t+1} = s'\right\}\right] \\
=& \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right]
\end{aligned} \tag{3.10}$$

上式暗含着动作a来自集合$\mathcal{A}(s)$,下一个动作s'来自于状态集合$\mathcal{S}$，或者是对于片段式问题是集合$\mathcal{S}^+$. 方程3.10叫做V^{π}的贝尔曼方程.它表达了状态值和它后继状态

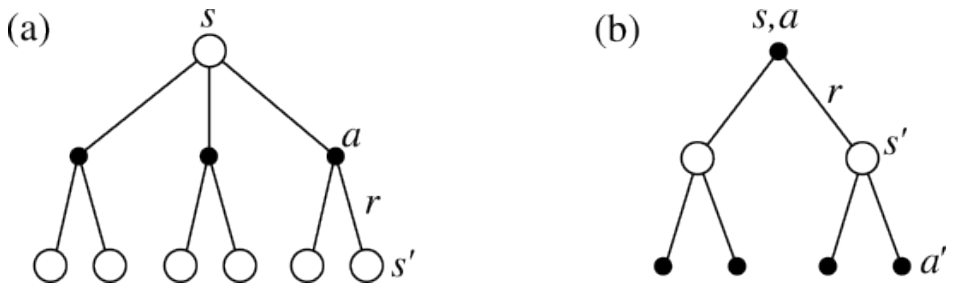


图 3.4: (a) V^{π} 和（b）Q^{π} 的备份图

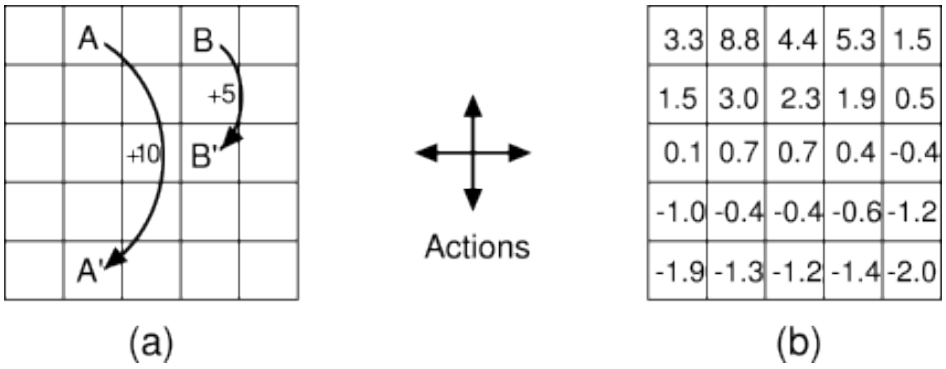


图 3.5: 网格例子(a) 动态回报和（b）等概率随机策略的状态值函数

值的关系。考虑一下图3.4a表示状态之间的关系。每一个空心圆代表一个状态，每一个实心圆代表一个状态-行为对。从状态 s开始，根节点在顶部，智能体可以动作集中的任何一个——图3.4有3个。对于每一个动作都可能到达下一步几个不同的状态 s'，伴随着一个汇报r.贝尔曼方程3.10对所有的可能发生的状态进行加权平均。

值函数是贝尔曼方程的唯一解。图3.4叫做备份图。它表示了更新的过程以及备份操作。不同于状态图的是备份图的节点没必要不同，比如一个结点可能是它自身的后继结点。

图3.5用矩形来表示每个状态。每个网格表示一个状态，在每个网格处有东西南北四个行为选择。对于每个选择有不同的回报。

从图（b）看出边缘的状态的值小，是因为可能出界，所以带来负数的乘法。最好的状态是状态A=8.8，可以看到从A出发的立即回报是10，最后行为值确实8.8¡10,这是因为A的后继状态位于边缘，带来了负的回报。

3.8 最优值函数

解决增强学习任务意味着找到一个策略能够获得最大的回报。值函数是定义在策略上的一个偏序。一个策略π比策略π'好当且仅当 $V^{\pi}(s) >= V^{\pi'}(s) \quad for all s \in \mathcal{S}$.最优策略对应的状态值函数叫做最优状态值函数，表示为V^{*}，定义为：

$$V^*(s) = \max_\pi V^\pi(s) \qquad for\ alls \in \mathcal{S} \tag{3.11}$$

最优策略通常也有共同的最优行为值函数，表示为Q^*，定义为：

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \qquad for\ alls \in \mathcal{S} \quad a \in \mathcal{A}(s) \tag{3.12}$$

最优行为值函数和最优状态值函数的关系：

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a\right\} \tag{3.13}$$

因为V^*是策略的值函数，必须满足贝尔曼方程的自治条件。但是因为他是最优值函数，V^*的自治条件可以被写成一个钟特殊的形式而不用参考具体的策略。这就是贝尔曼最优方程。直觉上，贝尔曼最优方程表达了这样一个事实：最优策略下的状态值必须等于从该状态开始的最优的期望动作获得的回报：

$$\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s,a) \\
&= \max_a E_{\pi^*}\left\{R_t \middle| s_t = s, a_t = a\right\} \\
&= \max_a E_{\pi^*}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\} \\
&= \max_a E_{\pi^*}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s, a_t = a\right\} \\
&= \max_a E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a\right\} \qquad (3.14) \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \qquad (3.15)
\end{aligned}$$

上面的两个方程是对于V^*贝尔曼最优方程的两种形式。对于Q^*的贝尔曼优化方程是：

$$\begin{aligned}
Q^*(s,a) =& E\left\{r_{t_1} + \gamma \max_{a'} Q^*(s_{t+1,a'}) \middle| s_t = s, a_t = a\right\} \\
=& \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right]
\end{aligned}$$

图3.6图形化的表示了未来状态和行为的跨度，基于最优贝尔曼方程。这些和针

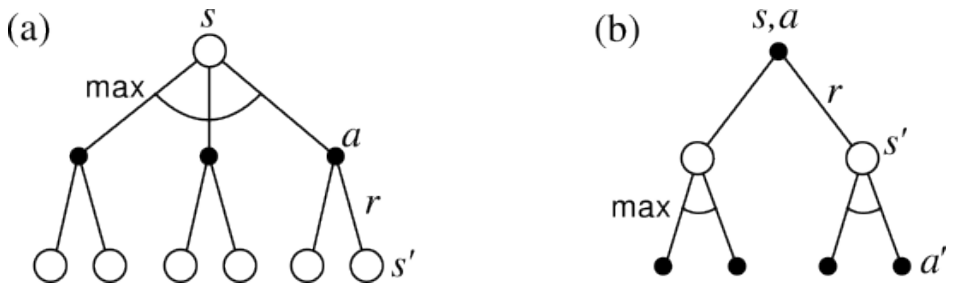


图 3.6: V^*和Q^*的备份图

对V^π和Q^π的备份图相同除了在智能体选择点多加的圆弧，用来表示选择的最大值而不是给定某个策略的期望值。

对于有限的MDPs，贝尔曼方程具有独立于策略的唯一解。贝尔曼最优方程实际上是个系统方程组，有N个状态N个未知数。如果环境的动态性是已知的，也即是在状态s选择行为a获得的回报值和过渡到下一个状态s' 都是已知的。那么可以用各种用于解非线性方程的方法求解出V^*.也可以求解相关的Q^*。

一旦我们有了V^*，求解最优决策变得相对容易。对于每个状态s，将会有一个或多个行为回报的最大值可以通过贝尔曼优化方程求出。任何给这些行为分配非0概率的决策都是一个最优决策。你可以看成是一步搜索。如果你有了最优值函数V^*，那么在一步搜索之后出现的最优行为将会是最优的。另一种说法是任何一个对最优值函数采用贪婪策略的策略是最优的。贪婪这个词被用在计算机领域，用来描述所有的只考虑局部的或者立即回报的搜索或决策过程，而不考虑这样的一个选择可能会阻止程序求得更有的替代解。结果是他描述了只基于短期结果而选择行为的决策。使用V^*的美妙之处在于如果你使用了它去估计一个短期的结果——具体来说，就是一步搜索的结果——然后一个贪婪的策略实际上在长期来看也是最优的，这是因为我们感兴趣的V^*已经考虑了将来可能行为的回报结果。

用Q^*同样是的最优行为的选择变得容易。使用Q^*，用户没必要提前做一步搜索：对于任意的状态s，它都能找到一个行为最大化$Q^*(s,a)$。采取状态-动作对的表示比只有状态值函数更复杂，但是通过行为值函数做最优决策的时候没必要考虑后继状态以及他们的值，也就是不用考虑环境的动态性。

3.9 优化和近似

我们定义了最优值函数和最优策略。按照前面的方法只需要求解贝尔曼方程就能得到一个最优策略，但实际上这种情况很少发生。因为当状态很多时，计算贝尔曼方程需要付出相当大的计算代价。另外内存的限制也是一个问题。这些问题迫使我们寻找一个近似的解决方法。例如有些状态对于智能体来说很少出现，或者对于回报

的贡献值很低，这样我们就可以忽略这些状态。专注于在频繁出现的状态的决策，忽略不太出现的状态。这是区分增强学习方法和其他近似求解MDP问题方法的关键特性。

Part II

初级的求解方法

在这一部分我们描述三类基本的求解增强学习问题的方法：动态规划，蒙特卡罗方法和时间差分学习。所有这些方法求解整个问题，包括延时回报。

每一类方法有它的优缺点。动态规划方法具有很好的数学基础，但是需要完整的、精确的的环境模型。蒙特卡罗方法不需要模型并且概念简单，但是不适合一步一步的增量式计算。最后，时间差分方法不需要模型并且完全增量式，但是分析困难。这些方法在效率和收敛速度上也有区别。在第三部分我们探索如何结合这些方法以便获得各自的最好特性。

Chapter 4

动态规划

动态规划这个词指的是用于在马尔科夫决策过程中针对确定模型求解最优策略的一类算法。经典的DP算法由于对环境的完美假设和大量的计算代价使得它在增强学习中用途有限，但是他们依然有重要的理论意义。

DP算法或者通常的增强学习算法的核心思想是用值函数来指导和构建最佳策略的搜索。本章将讨论DP算法如何被用于计算在第3章讨论的值函数。正如这里讨论的，如果我们知道了最优值函数V^* *or* Q^* 就能很轻易的获得最优策略，V^* *or* Q^*满足下面的方程：

$$\begin{aligned} V^*(s) &= \max_a E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a\right\} \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \end{aligned} \tag{4.1}$$

$$\begin{aligned} Q^*(s,a) &= E\left\{r_{t_1} + \gamma \max_{a'} Q^*(s_{t+1,a'}) \middle| s_t = s, a_t = a\right\} \\ &= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right] \end{aligned} \tag{4.2}$$

4.1 值估计

首先我们考虑如何对于任意的策略π计算状态值函数。在DP里叫做策略估计。也经常被称作预测问题。回顾第三章，对于所有的$s \in \mathcal{S}$

$$
\begin{aligned}
V^{\pi}(s) =& E_{\pi}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t = s\right\} \\
=& E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1} \mid s_t = s\right\} \qquad (4.3) \\
=& \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right] \qquad (4.4)
\end{aligned}
$$

$\pi(s,a)$是对于策略pi处于状态s时采取动作a的概率。

如果环境的动态特性是完全已知的，那么式4.4就是一个具有$|\mathcal{S}|$个未知数$(V^{\pi}(s), s \in$ $\int),|\mathcal{S}|$ 联立的线性方程.原则上，这个解是直接的。但是我们的目的是用迭代的方法。考虑一个近似值函数序列$V_0, V_1, V_2, \cdots,$ 每一个都是从$\mathcal{S}^+$到$\mathfrak{R}$的映射。初始的近似值V_0任意选择（除了终止状态，如果有的话必须为0），每一个后继的近似解通过3.10的更新规则获得。

$$
\begin{aligned}
V_{k+1}(s) =& E_{\pi}\left\{r_{t+1} + \gamma V_k(s_{t+1} \mid s_t = s\right\} \\
=& \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \qquad (4.5)
\end{aligned}
$$

实际上在某些条件下保证V^{π}存在，那么当$k \to \infty$时序列$\{V_k\}$收敛到V^{π}。这种算法叫做迭代策略估计。

为了获得V_{k+1}我们需要某个状态s的所有后继状态，所有期望立即回报，一步转换的概率。这种方法我们叫做全备份。之所以叫做全备份是因为它是基于所有可能的下一个状态而不是下一个状态的抽样。图3.4就是全备份的例子。

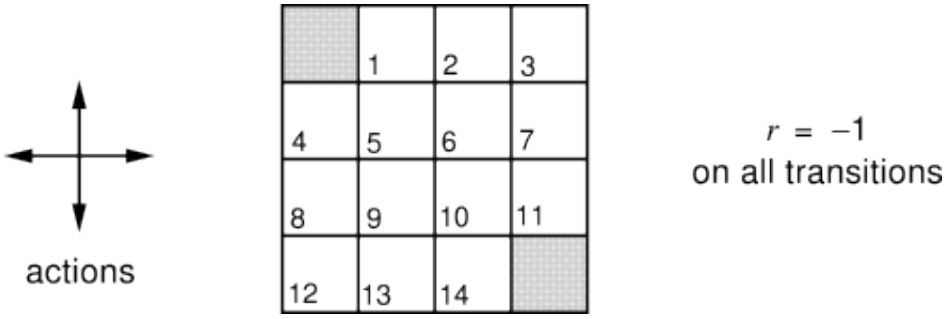
我们在编程实现这个算法时，如4.5给出，可以建立两个数组，一个用于存旧的值，一个是新的值。更新时有两种思路，一是更新新的，旧的都不变。另一种是更新了一个新的值就地替换原来的旧的值，这样只需要一个数组，并且这种方式往往收敛的速度更快，因为它总是尽快的使用最新的值来进行更新。

例4.1考虑一个4×4的网络世界如下所示：每个状态有4个动作，东西南北。灰色表示的是终止状态，每一次行为尝试都获得-1的立即回报。如果状态处于边界，并且行为迫使状态脱离表格时，假设这个行为又回到自己本身。

Algorithm 1 策略估计迭代方法.

Input: 策略π，要估计的策略；
Output: 输出最优状态值，$V \approx V^{\pi}$;

初始化 $V(s) = 0$,对于所有的$s \in \mathcal{S}^{+}$;
repeat
  $\triangle \leftarrow 0$
  for each $s \in \mathcal{S}$: **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right]$
    $\triangle \leftarrow \max(\triangle, |v - V(s)|)$
  end for
until $\triangle < \theta$（一个很小的正数）
return V^{π}



采用前面的策略估计迭代方法对值函数更新,求解过程见图4.1

4.2 策略改善

我们计算值函数的目标是为了找到更好地策略。假设对于任意的策略π我们已经确定了值函数V^{π}.现在我们想知道对于某些状态s我们是否应该改变策略而选择行为$a \neq \pi(s)$。我们已经知道了采取当前状态时表现的好还，但是如果是一个新的策略结果会更好还是更坏？一种考虑这个问题的方式是选择一个行为a然后遵循存在的策略π。这种方式下的值表示为：

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\right\} \\ &= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right] \end{aligned} \tag{4.6}$$

评价的关键是这个值是否大于或者小于$V^{\pi}(s)$。如果更大——也就是说在状态s选择a然后在根据策略π选择比始终根据策略π选择要好，那么用户可以期望当每

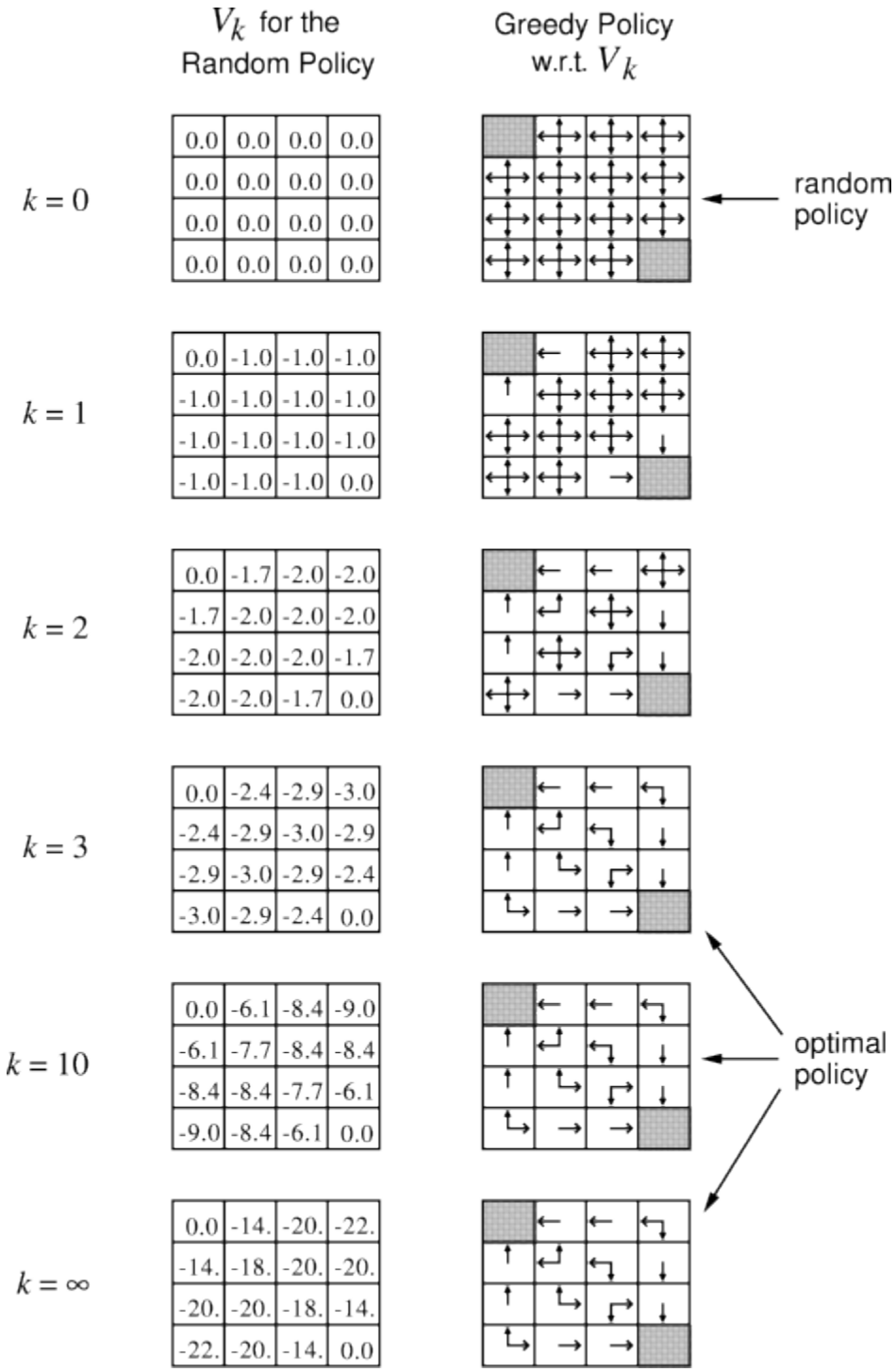


图 4.1: 例4.1的策略迭代过程

次面临s时a都是更好的选择，那么新策略整体上会更好。

这是一般结果的一个特例叫做策略提升理论.令π和π'分别表示任意对确定性决策满足：

$$Q^{\pi}(s,\pi'(s)) >= V^{\pi}(s) \tag{4.7}$$

那么我们说π'要好于π'，或者至少一样好。上式对于任意一个$s \in \mathcal{S}$成立。

$$V^{\pi'}(s) >= V^{\pi}(s) \tag{4.8}$$

更进一步，如果4.7对于任何状态严格不等，那么有式4.8至少对于一个状态严格大于。这个结论特别的被用在我们之前讨论的两个策略，一个原始的策略π和改变的策略π'——和策略π完全相同除了$\pi'(s) = a \neq \pi(s)$。显然式4.7对于所有的状态成立，除了s.

策略提升理论背后的思想很简答，我们从式4.7开始持续展开Q^{π}，再利用4.7直到我们得到$V^{\pi'}(s)$。

$$\begin{aligned}
V^{\pi}(s) &\leq Q^{\pi}(s,\pi'(s)) \\
&= E_{\pi'}\left\{r_{r+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\right\} \\
&\leq E_{\pi'}\left\{r_{r+1} + \gamma Q^{\pi}(s_{t+1},\pi'(s_{t+1})) \mid s_t = s\right\} \\
&= E_{\pi'}\left\{r_{r+1} + \gamma E_{\pi'}\left\{r_{r+2} + \gamma V^{\pi}(s_{t+2}) \mid s_t = s\right\}\right\} \\
&= E_{\pi'}\left\{r_{r+1} + \gamma r_{t+2} + \gamma^2 V^{\pi}(s_{t+2}) \mid s_t = s\right\} \\
&\leq E_{\pi'}\left\{r_{r+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^{\pi}(s_{t+3}) \mid s_t = s\right\} \\
&\vdots \\
&\leq E_{\pi'}\left\{r_{r+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots | s_t = s\right\} \\
&= V^{\pi'}(s)
\end{aligned}$$

到此为止，给定一个一直策略我们很容易通过改变某个状态的行为实现一个新的策略。很自然的想法是改变所有状态的可能行为，选择$Q^{\pi}(s,a)$最大的行为。换句话说，考虑一个贪婪策略π'，通过下式给定：

$$\begin{aligned}\pi'(s) &= \arg\max_a Q^\pi(s,a) \\ &= \arg\max_a E\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a\} \\ &= \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]\end{aligned} \tag{4.9}$$

通过策略提升定理4.7我们知道上式构造出的策略的表现不会比原来的差。这种通过对值函数采取贪婪策略而使得策略得到改进的过程叫做策略提升。

4.3 策略迭代

一旦一个策略π 通过使用V^π的到了改善，得到了策略π',然后我们可以计算$V^{\pi'}$改善π'而获得π''。因此如此重复我们就能获得一系列自动改善的策略和值函数：

$$\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*,$$

$\underrightarrow{E}$表示策略估计 $\underrightarrow{I}$ 表示策略改善。每一次的迭代都得到了更有的策略。对于有限的MDP问题，策略集合是有限的，所有总能通过有限次迭代收敛到最优值。这种寻找最优值的方法叫做策略迭代.

4.4 值迭代

策略迭代的一个缺点是每一次迭代都会涉及策略估计，也就是当前情况下的值函数计算，这样可能需要多次的更新状态机的值而拖延迭代计算。另一点是有的时候尽管值估计没有收敛但是已经得到最优，浪费了计算。

实际上，每次策略迭代的策略估计能够被缩短，并且不影响收敛性。一个重要的特殊的情况是策略估计在一次更新后停止。这个算法叫做值迭代.

$$V_{k+1}(s) = \max_a E\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\} = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right] \tag{4.10}$$

这里注意策略迭代和值迭代的区别。策略迭代中策略更新计算的是在每个状态s下各个行为的期望值，然后用这个值覆盖掉原来的，而值迭代用的是用某个行为的最大值更新原来的值。

值迭代算法的步骤如下：

Algorithm 2 策略迭代.

1.初始化
$V(s) \in \Re$,$\pi(s) \in \mathcal{A}(s)$　对于所有的$s \in \mathcal{S}^+$;

2.策略估计
repeat
　$\triangle \leftarrow 0$
　for each $s \in \mathcal{S}$: **do**
　　$v \leftarrow V(s)$
　　$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right]$
　　$\triangle \leftarrow \max(\triangle, |v - V(s)|)$
　end for
until $\triangle < \theta$（一个很小的正数）

3.策略迭代
$policy - stable \leftarrow true$
for each $s \in \mathcal{S}$ **do**
　$b \leftarrow \pi(s)$
　$\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]$
　if $b \neq \pi(s)$ **then**
　　$policy - stable \leftarrow false$
　end if
end for
如果策略稳定，停止，否则转向步骤2

例*4.3*：赌博问题 一个赌徒依靠掷硬币赌钱，如果正面朝上他将会获得和下注筹码一样的回报，如果相反，它将会输掉下注的钱。当赌徒赢够100美元，或者输光他的钱，游戏结束。这个问题可以被看成片段式的MDP问题（每一次下注就是一个片段）。状态是他手上现有的资本，$s \in \{1, 2, 4, \ldots, 99\}$;行为就是赌注的大小，$a \in \{0, 1, 2, 4, \ldots, \min(s, 100 - s)\}$ 。除了赌徒达到了目标奖励为1之外，其他的过渡状态奖励都是0。策略就是根据资本决定赌注大小。用p表示硬币正面朝上的概率。用值迭代的方法可以求出最优策略，如图4.2所示：

4.5 异步动态规划

目前为止讨论的DP算法的一个主要缺点是他们都需要对MDP问题的整个状态集进行操作。当状态集很大的时候，单步的刷新也可能代价不菲。

异步的*DP*算法不是对状态集采取系统的刷新，而是在线的迭代。算法以任意得次序更新状态，使用任何可用的其他状态值。一些状态可能已经刷新了很多次，而其他状态的值可能只更新了一次。但是为了正确收敛，一个异步算法必须持续备份

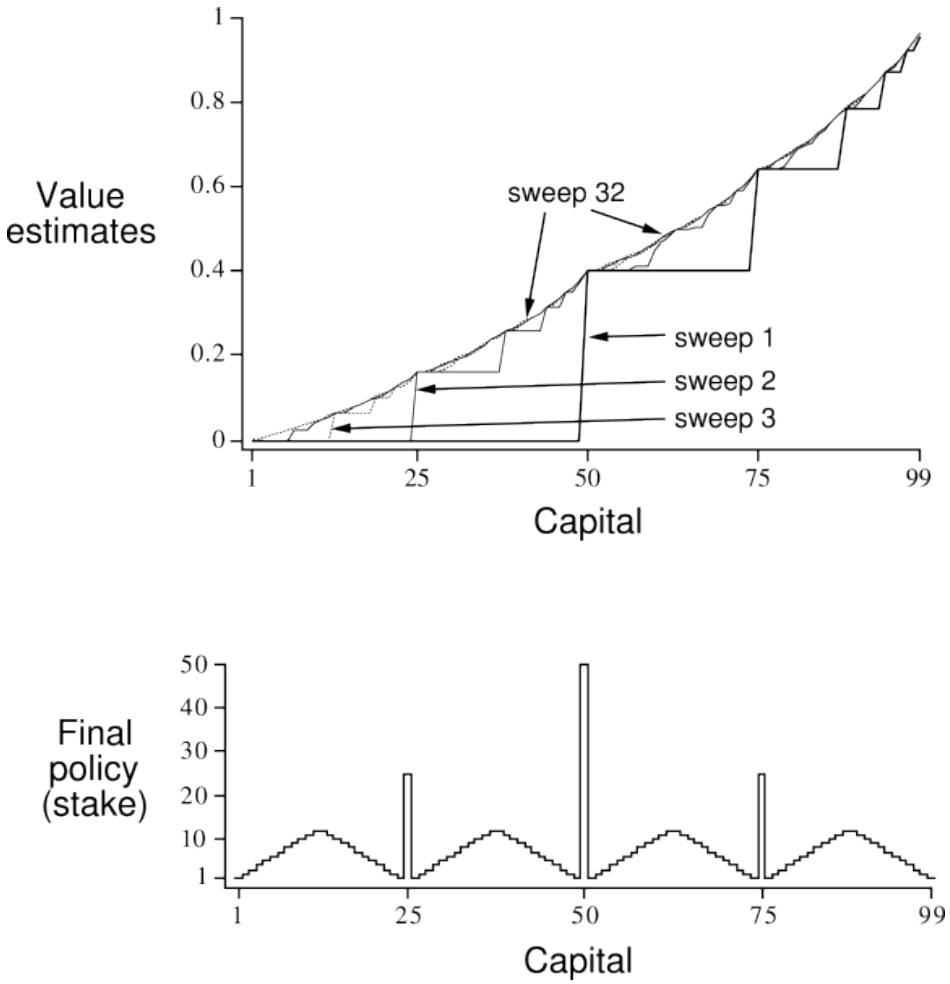


图 4.2: 赌博问题的解，$p = 0.4$.上图是值函数。下图是最终的策略

Algorithm 3 值迭代.

1.初始化
$V(s) \in \Re$,可以为任意的，比如$V(s) = 0$, 对于所有的$s \in \mathcal{S}^+$;

repeat
  $\triangle \leftarrow 0$
  for each $s \in \mathcal{S}$: **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')]$
    $\triangle \leftarrow \max(\triangle, |v - V(s)|)$
  end for
until $\triangle < \theta$（一个很小的正数）

return 一个确定的策略π，满足：
$\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')]$

所有状态值。异步DP算法对于需要更新的状态的选择具有很大的灵活性。有了这种灵活性，我们可以通过选择更新状态来提高算法的改善速度。我们可以排序刷新的次序来是状态之间的值传递更加高效。我们甚至可以跳过某些状态的备份，如果该状态和最优行为关联很小。第9章会讨论相关思想。

异步算法也使得计算和实时交互的融合更加容易。

4.6 广义的策略迭代

策略迭代包括两个同时，交互过程：使得值函数与当前策略一致（policy evaluation），另外一点是针对当前的值函数指定贪婪策略（policy improvement）。在策略迭代中，这两个过程交互进行，一个开始另一个结束，但是实际上这个是没有必要的。比如值迭代，每两次策略更新之间只有一次单步的策略估计。在异步DP算法中，评估和改善过程的间隔更加精细化。在某些情况从一个过程到另一个过程之间只有一个状态被更新。只要两个过程持续的更新所有的状态，最终的结果都是相同的。

我们使用广义的策略迭代（GPI）表示更一般的思想：让策略估计和改善进行交互，且独立于间隔和其他二者的细节。GPI的整体方案如下图4.3表示：

可以看出评估和改善交替进行，相互促进，但是最总收敛到一个最优的解。此时该值函数下的策略是一致的。也可以用下图来理解这个问题：

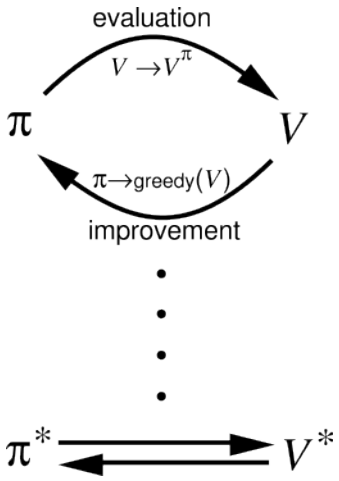
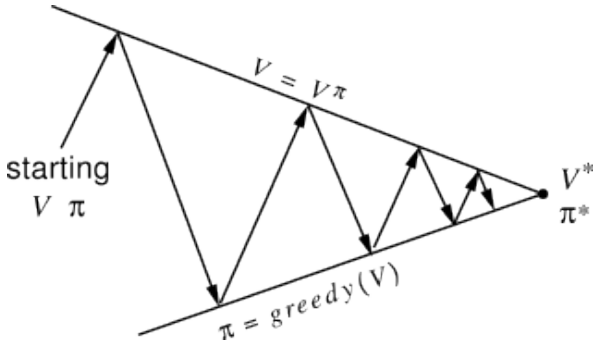


图 4.3: 广义策略迭代：值函数和策略函数交互直到最优并且相互吻合



4.7 总结

在本章我们对于动态规划算法作为解决有限MDP问题的基本概念有了了解。策略估计指的是给定策略下值函数的迭代计算。策略改善指的是在给定值函数下一个改善的策略。把这二者放在一起，获得了策略迭代和值迭代方法，这是DP算法中最流行的两种算法。在给定MDPs的完全先验知识的情况下，任何一种方法都能可靠的求解出最优策略和值函数。

经典的DP算法在整个状态集上刷新，对每一个状态都做全备份操作。每一个备份通过其后继状态的值以及出现的概率更新自身的值。全备份和贝尔曼方程紧密相关：他们略有不同的就是全备份是一种分配的描述。当备份不再导致值的变化时，就收敛了。正如我们有四个主要的值函数$(v^{\pi}, v^*, Q^{\pi}Q*)$,相应的有4个贝尔曼方程和4个相应的全备份。备份操作可以通过备份图直观的认识。

深入看待DP算法，实际上，几乎所有的增强学习方法都可以被看待为广义的策略迭代获得的。GPI是围绕近似策略迭代和近似值函数交互过程的一般性思想。一个过程以给定策略为基础进行策略估计，改变值函数的值使之更接近该策略的真实值。另一个过程是根据值函数进行策略改善，改变策略使之变得更好。

Chapter 5

蒙特卡罗方法

这一章我么首次考虑值函数估计和最优解挖掘的学习方法。和前面章节不同，这里我们假设对环境知识一无所知。蒙特卡罗方法只需要在线的或者模拟与环境的交互来抽样状态、动作、回报的序列。在线的经验学习是令人震惊的因为他不需要任何环境的动态先验知识，然而依然能够获得最优行为。模拟的经验学习是强大的。尽管也需要一个模型，但是这个模型只需要产生式样的过渡概率，不需要对于动态规划方法的所有可能过渡的完整概率分布。令人惊讶的是很多其工况下，根据期望概率分布抽样产生经验是容易的，但是不能获得分布的显示形式。

蒙特卡罗方法解决增强学习问题的方法基于抽样回报的均值。为了保证定义好的回报是可获取的，我们只针对片段式任务定义蒙特卡罗方法。也即我们假设经验被分成很多个片段，无论选取何种选择片段最终都会终止。只有完成了一个片段值函数和策略才会改变。因此从片段的概念上蒙特卡罗方法是增量式的，但是不是逐步增量式。“蒙特卡罗”这个词经常被广泛的用在具有很大随机性的估计问题上。这里我们使用它主要是基于完全的回报均值。

尽管蒙特卡罗方法和DP方法有很大不同，但是他们的核心思想是一脉相承的。不仅是对于值函数的计算，也包括通过交互获得最优解。在DP那一章，我们首先考虑策略估计，V^π 和Q^π 的计算对于一个固定的任意策略，然后策略改善，最后广义的策略迭代。这些都能扩展到蒙特卡罗方法。

5.1 蒙特卡罗策略评估

对于一个给定的策略，首先考虑用蒙特卡罗方法学习状态值函数。回顾一下一个状态的值时他的期望回报——期望的未来折扣累积回报。一个显然的方法是通过经验估计，然后简单的平均观测到那个状态之后的回报。当更多的回报被观测到时，平均值应该收敛到期望值。这是所有蒙特卡罗方法暗含的思想。

特别的，假设我们想要估计$V^{\pi}(s)$——策略π下状态s的回报，基于一系列遵循策略π并且经过状态s的片段。在每个片段中每次s的发生叫做一次访问。every-visit MC方法是平均那些经过s的片段的回报来估计$V^{\pi}(s)$。first-visit MC方法只是平均第一次访问s后的回报的平均。这两种方法很相似但是理论特性上略有不同。First-visit MC研究的很广泛，回溯到1940s，也是我们本章中着重讲述的方法。第七章考虑every-visit MC方法。every-visit MC方法的步骤如下：

Initialize:
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
(a) Generate an episode using π
(b) For each state s appearing in the episode:
$R \leftarrow$ return following the first occurrence of s
Append R to $Returns(s)$
$V(s) \leftarrow \text{average}(Returns(s))$

图 5.1: First-visit MC方法

例5.1二十一点是俱乐部有名的纸牌游戏。目标获得纸牌的点数之和尽量大且不超过21点。花牌的点数为10，ace的点数可以是1或者11.我们考虑每一个玩家和庄家独立的竞争。游戏开始庄家和玩家各发2张牌。庄家的牌一张向上一张向下。如果玩家的点数刚好是21点（一张ace和10点），称为黑杰克(*natural*)。这种情况下玩家赢除非庄家也是21点，形成平局。如果玩家没有21点，他可以要求更多的牌(*hits*)，一张一张的直到停牌(*natural*)或者超过21 点(*goes bust*)。如果超过21 点，玩家就输了；如果玩家选择停牌，那么轮到庄家。庄家要求要牌或者停牌根据固定的策略——点数之和大于14停牌，否则继续要牌。如果庄家超过21 点，那么玩家赢；否则结果的输赢或者平局取决于最后谁的点数更接近于21。

对于这个游戏每一局是一个片段。输、赢、平局对应的回报分别是-1、+1和0.玩家的行为包括要牌或者停牌，状态依赖于玩家的牌和庄家翻开的牌。玩家的决策基于3个变量：目前牌数目和（12-21），庄家翻开的牌以及是否有10点的ace。总共构成了200个状态。

注意在这个问题中的first-visit 和every-visit MC是一样的，因为在一局游戏中不可能出现两个相同的状态。图5.2展示状态值函数的估计。对于ace作为11点时的情况更加不确定和没有规律这是因为这些状态不是很普遍。经过500.000局之后获得值函

数的近似解。

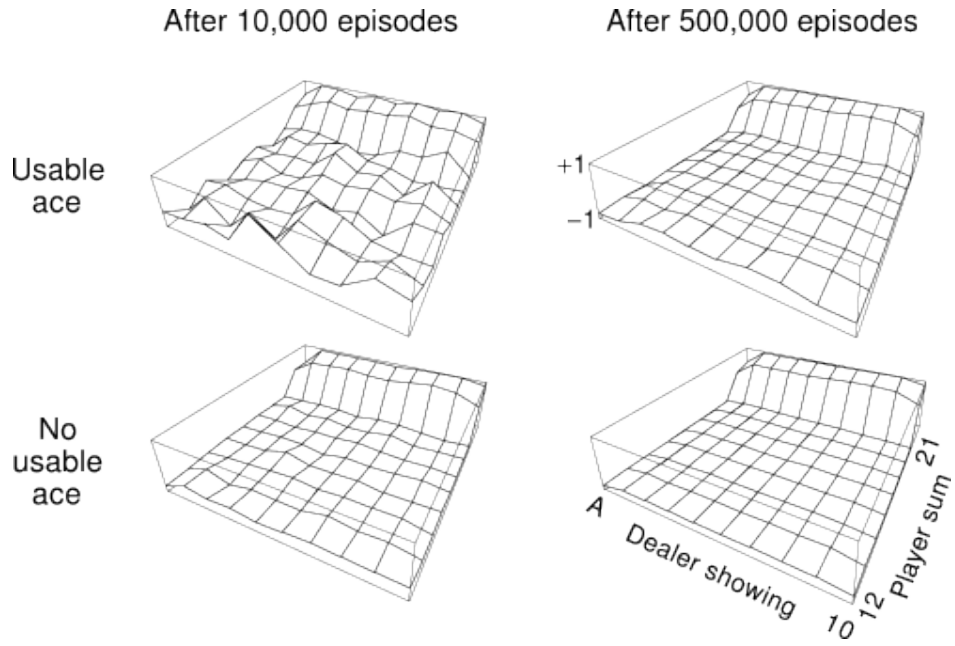


图 5.2: 在20点和21点才停牌的策略下黑杰克问题的近似状态值函数

蒙特卡罗和DP的比较：1）从备份图上来说，DP包括所有的可能的转移概率，蒙特卡罗只包含抽样到的game中的状态；2）蒙特卡罗法状态值的估计是相互独立的。换句话说该方法不是"booststrap"——通过估计其他值来获得自身的估计。这个有点使得我们可以只估计出一个状态子集的值；3）通过抽样省去了繁琐的DP里的计算。

5.2 蒙特卡罗行为值函数的估计

如果模型无法获得，那么估计行为值比状态值更加有用。有了模型，仅仅状态估计就足够来做决策，但是对于MC算法不是这样。必须明确的估计每一个行为的值以便做出有用的决策。因此MC算法首要的目标是估计Q^*。为了实现这个目的，首先考虑另一个策略估计问题。

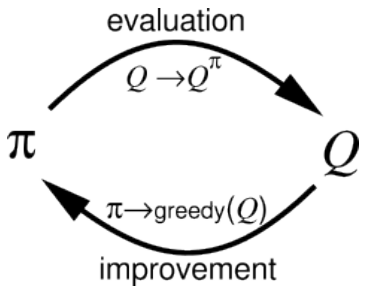
对行为的策略估计问题是估计$Q^\pi(s,a)$.采用MC方法也是一样，对每个出现状态s，然后选择了行为a的地方统计其回报，然后求平均。

采样会出现的一个问题是有些状态-行为对可能从来没有被访问到。那么对于这些状态行为就没有回报，就没法使用MC方法完成其值函数的估计。这通常是一个维护探索的问题。为了保证策略迭代对于所有行为值有效，我们必须保证持续的探索。一种解决的方法是明确每一个片段在一个特定的状态行为对，这样就能保证每个状态-行为对具有非0的概率被选择。我们称之为探索初值假定。

这种假定有时时有效的，但是显然不是通常意义上可以依赖的方法，尤其是当面临实际的环境交互时。一种常用的替代策略是假设策略是随机的以非0的概率选择所有动作。后面将会讨论2种常用的该方法的变种。暂时先用探索初值假定完成MC控制方法的表述。

5.3 蒙特卡罗控制

现在我们开始考虑MC方法如何被用于控制，也就是近似最优策略。大体上的思想还是遵循GPI框架，如下图



开始之前，让我先考虑一个MC版本的策略迭代。在这个方法中，我们交替的实现策略评估和策略改善，起始于任意的策略π_0,如下所示：

$$\pi_0 \xrightarrow{\text{E}} Q^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} Q^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} Q^*,$$

策略改善通过选择符合当前值函数的贪婪策略获得。在这种情况下我们有了行为值函数，因此不需要模型来构造贪婪策略。对于任何的行为值函数Q,相应的贪婪策略就是要选择最大值的Q函数：

$$\pi(s) = \arg\max_a Q(s,a) \tag{5.1}$$

为了获得MC方法的收敛保证我们做了两个不太可能的假设。一个是初始探索假定，另一个是具有无限个片段。为了获得实际可用的算法我们必须去掉这些假设。先考虑第二个假设。

有2个解决方法。第一个是紧紧把握近似的思想。做出测量和假设来获得估计误差的幅值和概率，在每次策略评时采取足够的步数来保证这些界限足够小。这个方法可能在保证正确收敛到某种水平的近似值上是完全令人满意的。但是，实际中仍然需要许多个片段。

第二个避免无限片段评估的方法是在策略改善之前放弃完全的策略评价。在每一步趋向值函数Q^{π_k}时，我们不期望得到足够接近的实际值。一个极端的形式就是值

迭代，在这个方法中每两次策略改善之间只进行一次策略评价的迭代。在线更新的值迭代版本更极端。

基于片段的评价和改善是MC算法策略评估很自然的想法。在每一个片段后，观测的回报被用于策略更新，然后在所有该片段访问过的节点处进行策略改善。一个完整的简单的算法如图5.3所示。我们称这种方法为*Monte carlo ES*,指的是*Monte carlo ES* with Exploring Starts.

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$\pi(s) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list

Repeat forever:
(a) Generate an episode using exploring starts and π
(b) For each pair s,a appearing in the episode:
$R \leftarrow$ return following the first occurrence of s,a
Append R to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each s in the episode:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

图 5.3: *Monte carlo ES*

例5.3：Solving Blackjack图5.4展示用*Monte carlo ES*方法获得的最优策略和Q值：

5.4 在线策略MC控制

我们应该如何避免不大可能的探索初始假设呢？一般的方式只有智能体不断的选择那些行为才能保证所有的动作被选择无限次。有两种方法来保证这个事实，就是我们所说的*on-policy*方法和*off-policy*方法。*on-policy*方法尝试评价和改善自己做决策的策略。这一部分我们展示*on-policy* Monto Carlo方法。

在*on-policy*控制方法中策略通常是软的，意味着$\pi(a,a) > 0$。对于*on-policy*方法有很多可能的变种。一种可能的方式是逐渐改变策略值趋向一个确定的策略。第2章中讨论的很多方法提供了这个机制。这部分我们展示的*on-policy*方法使用$\varepsilon - greedy$策略，以为在大部分的时间我们选择具有最大行为估计值的行为，但是以ε的概率随机的选择一行为。也就是所有非贪婪的策略有个最小的选择概率

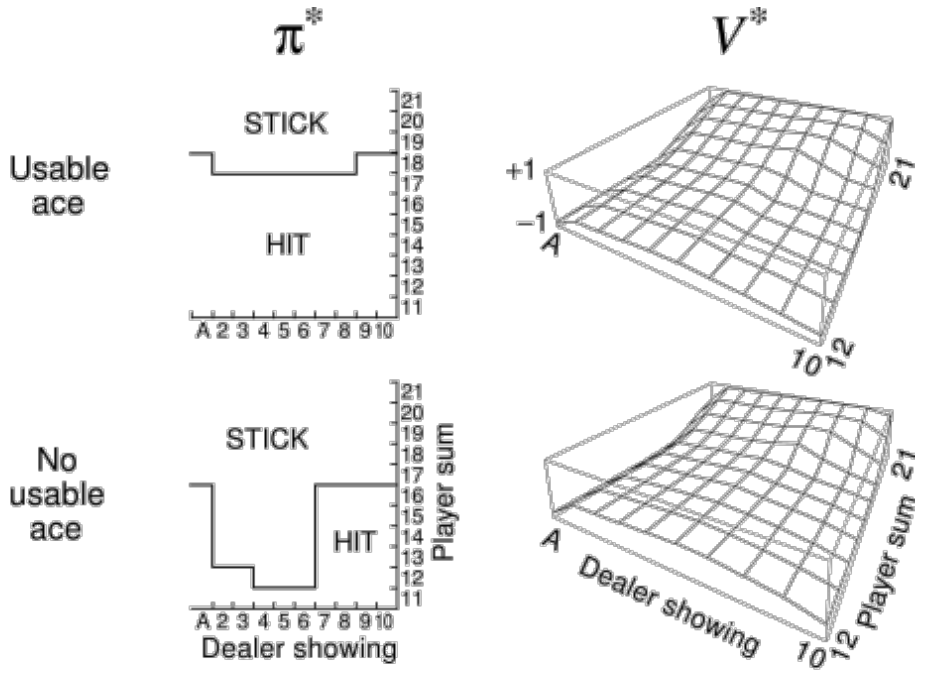


图 5.4: *Monte carlo ES*

在$\frac{\varepsilon}{\mathcal{A}(s)}$,剩下的大部分概率，$1-\varepsilon+\frac{\varepsilon}{\mathcal{A}(s)}$,给了贪婪行为。$\varepsilon-greedy$策略是$\varepsilon-soft$策略的例子——定义为对于所有状态和行为$\pi(s,a)\geq\frac{\varepsilon}{\mathcal{A}(s)}$的策略。在所有的$\varepsilon-soft$策略中$\varepsilon-greedy$是最接近贪婪策略的。

Monto Carlo *on-policy*控制方法的大体思想还是GPI。但是没有探索初值假定的条件，我们不能仅仅通过使得策略变得贪婪来改善策略，因为这样会阻止对于非贪婪策略的进一步探索。幸运的是，GPI不需要策略一直趋向贪婪策略，只需要向贪婪策略前进就行。在我们的*on-policy*方法中我们只让他趋向$\varepsilon-greedy$策略。对于任何的$\varepsilon-greedy$策略π,$\varepsilon-greedy$的Q^{π}被保证被原来的好。

任何对应Q^{π}的$\varepsilon-greedy$策略是$\varepsilon-soft$策略π的改善版可以由策略提升理论保证。用π'表示$\varepsilon-greedy$策略，策略提升理论成立因为：

$$
\begin{aligned}
Q^{\pi}(s,\pi'(s)) &= \sum_{a}\pi'(s,a)Q^{\pi}(s,a) \\
&= \frac{\varepsilon}{\mathcal{A}(s)}\sum_{a}Q^{\pi}(s,a) + (1-\varepsilon)\max_{a}Q^{\pi}(s,a) \\
&\geq \frac{\varepsilon}{\mathcal{A}(s)}\sum_{a}Q^{\pi}(s,a) + (1-\varepsilon)\sum_{a}\frac{\pi(s,a)-\frac{\varepsilon}{\mathcal{A}(s)}}{1-\varepsilon}Q^{\pi}(s,a) \\
&= \frac{\varepsilon}{\mathcal{A}(s)}\sum_{a}Q^{\pi}(s,a) - \frac{\varepsilon}{\mathcal{A}(s)}\sum_{a}Q^{\pi}(s,a) + \sum_{a}\pi(s,a)Q^{\pi}(s,a) \\
&= V^{\pi}(s)
\end{aligned}
\tag{5.2}
$$

总结一下为了避免初始状态假定，我们引入了随机策略。也就是在我们决策的时候以一定的概率选择那些不是最大回报的行为值。这样提供了探索的可能性，保证了所有状态能被访问到。另外经过上面的证明说明了这样费贪婪的选择方式也会不断改进策略的性能。这样我们得到的是一个随机策略，也是$\varepsilon - soft$策略最好的选择。完成的算法由图5.5给出：

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list
$\pi \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
(a) Generate an episode using π
(b) For each pair s,a appearing in the episode:
$R \leftarrow$ return following the first occurrence of s,a
Append R to $Returns(s,a)$
$Q(s,a) \leftarrow$ average$(Returns(s,a))$
(c) For each s in the episode:
$a^* \leftarrow \arg\max_a Q(s,a)$
For all $a \in \mathcal{A}(s)$:

$$
\pi(s,a) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}
$$

图 5.5: ε-*soft Monte carlo*算法

5.5 用一个策略的经验评价另一个策略

目前为止我们使用概率方法的思想是先给你一个策略，用这个策略产生一些片段，对这个片段进行策略评价，得到值函数，然后再更新策略。总的来说策略的更新基于值函数，值函数的获取基于抽样的片段。那么当针对一个策略π我们没有它产生的片段，却有另一个策略π'产生的片段，这种情况下我们是否依然能估计V^{π}或者Q^{π}呢？

答案是肯定的。为了使用π'估计π，我们需要在π下采取的每个动作在π'下也发生过，也就是$\pi(s,a) > 0$意味着$\pi'(s,a) > 0$.用$p_i(s)$和$p'_i(s)$分别表示两个策略下在状态s后出现相同序列的概率。这样就可以通过相对的给吕估计相对的回报值。那么在状态s观测到n_s回报后的想得到的MOnto Carlo估计是：

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i}} \tag{5.3}$$

显然这里的$p_i(s)$和$p'_i(s)$是未知的。幸运的是我只需要知道他们的比率，而不需要知道各自的值，所有我们可以在没有环境动态性知识的情况下获得这个值。用$T_i(s)$表示第i个片段中涉及状态s的终止时间，那么

$$p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}^{a_k}_{s_k\, s_{k+1}}$$

然后：

$$\frac{p_i(s_t)}{p'_i(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}^{a_k}_{s_k\, s_{k+1}}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) \mathcal{P}^{a_k}_{s_k\, s_{k+1}}} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

这就是5.3中的需要的比值，它只依赖于两个策略和环境动态性无关。

5.6 无策略的蒙特卡罗控制

现在我们来讨论无策略方法。回想在线策略方法的显著特征是它在使用控制的同时进行值估计。在off-policy中这两个过程被分开了。用来产生行为的策略叫做行为策略，实际上可能和用于估计和改善策略的策略无关，叫做估计策略。这种区分的好处是估计策略可能是确定性的（比如贪婪的），然而行为策略可以是任何可能的行为。

Off-policy MC控制用上一节讲道的用一个策略估计另一个策略值函数的方法。通过遵循行为策略来学习估计策略的改善。这技术需要行为策略需要以非0的概率选

择那些估计策略可能选择的行为。为了探索所有可能，我们需要行为策略是软的。

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$N(s,a) \leftarrow 0$; Numerator and
$D(s,a) \leftarrow 0$; Denominator of $Q(s,a)$
$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:
(a) Select a policy π' and use it to generate an episode:
$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$
(b) $\tau \leftarrow$ latest time at which $a_\tau \neq \pi(s_\tau)$
(c) For each pair s, a appearing in the episode at time τ or later:
$t \leftarrow$ the time of first occurrence of s, a such that $t \geq \tau$
$w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k,a_k)}$
$N(s,a) \leftarrow N(s,a) + wR_t$
$D(s,a) \leftarrow D(s,a) + w$
$Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$
(d) For each $s \in \mathcal{S}$:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

图 5.6: *off-policy Monte carlo*控制算法

5.7 增量式实现

蒙特卡罗方法可以通过第2章中的描述的扩展方法实现增量式计算。使用增量式计算可以增量式的处理每一次新的回报，而不需要计算和内存的增加。

蒙特卡罗方法和赌博问题有2点不同。第一MC通常涉及多个场景，也就是每一个状态的平均处理方法不同。第二：MC的回报分布式不稳定的，因为回报基于策略，而策略一直在变化。

在2.5节中我们考虑的平均式纯粹的算术平均，也就是每个回报的权重都相同。假设我们想实现一个有权重的平均，也就是每一个回报R_n具有不同的权重w_n，我们要计算：

$$V_n = \frac{\sum_{k=1}^{n} w_k R_k}{\sum_{k=1}^{n} w_k} \tag{5.4}$$

比如在5.5节中使用的方法，权重$w_n(s) = \frac{p_n(s)}{p'_n(s)}$.权重平均也有一个简单的增量式

更新规则。W_n表示前n 次权重之和。

$$V_{n+1} = V_n + \frac{w_{n+1}}{W_{n+1}} [R_{n+1} - V_n] \tag{5.5}$$

在这里$W_{n+1} = W_n + w_{n+1}, W_0 = 0.$

5.8 总结

本章讲述的MC方法使用通过抽样的片段获取的经验来估计值函数的最优策略。这至少有三个优点。首先，它可以直接通过和环境交互学习到最优行为，不需要环境的信息。第二，他们可以通过仿真或者抽样模型使用。第三，对于蒙特卡罗方法来说，只针对状态集的自己的估计也是有效和方便的。第四个优点是它受马尔科夫性的影响小。

在设计MC控制方法是我们仍然遵循第4章介绍的GPI策略。GPI涉及策略评价和策略改善的交互过程。MC方法提供了策略评价过程的替代方法。不是用一个模型来计算每一个状态的值，只是简单的平均从这个状态开始的回报值。因为状态的值时期望值，平均值会是真是值的很好的近似。MC方法可以基于片段实现策略评价和策略改善的混合，同时也能基于片段的增量实现。

保持充分的探索是MC方法的一个主要议题。仅仅选择当前估计最好的值时不够的，因为然后没有替代行为的回报，他们甚至可能从来不能学习到即使他们实际上是最好的。一种解决的办法是假设片段的开始会随机的覆所有的状态-行为对。另外两种方法是*on -policy*和*off -policy*方法。

所有用于增强学习的MC方法最近才被明确定义。他们的收敛特性还不明显，他们在实际中的效果也测试不多。目前，他们首要的意义是他们的简介性以及和其他方法的关系。

MC方法和DP方法有2点不同。首先，它基于抽样经验，因此不需要模型直接学习。第二，他不是boostrap的，也就是他不是基于其他状态的值估计来更新值估计。

Chapter 6

时间差分学习

如果要确定增强学习领域一个核心和新颖的思想，毫无疑问是时间差分*temporal-difference*(TD)学习。TD学习是蒙特卡罗思想和动态规划思想的结合。像MC 算法，TD 算法可以直接从经验学习而不需要环境的模型。像DP算法，TD算法部分的基于其他的学习的估计来更新估计，而不用等到最后的结果。MC、DP 和TD 算法的关系是增强学习理论里老生常谈的话题。这章我们开始讨论TD算法。在我们开始之前，我们会看到这些思想和方法能够混合在一起并且以多种方法结合。特别的，在第7章我们讨论$TD(\lambda)$ 算法，它无缝的集成了TD和MC方法。

6.1 TD预测

TD算法和MC算法都使用经验来解决预测问题。给定一个遵循策略π的经验，两个方法都更新他们的估计V和V^{π}。概略的来说，MC方法等待直到某个状态之后的回报变成已知，然后用回报作为$V(s_t)$。一个简单every-visit MC 方法适用于一个非静态的环境是：

$$V(s_t) \leftarrow V(s_t) + \alpha\big[R_t - V(s_t)\big] \tag{6.1}$$

称这个方法为$constant-\alpha$ MC.但是MC 方法必须等到一个片段结束了才能决定增量$V(s_t)$（因为那个时候R_t才已知），TD需要等到下一个步长时间。在时间$t+1$它立刻形成一个目标并且利用观测的回报r_{t+1}和估计V_{t+1}来做有益的更新。最简单的TD算法，也就是TD(0) 如下：

$$V(s_t) \leftarrow V(s_t) + \alpha\big[R_{t+1} + \gamma V_{t+1} - V(s_t)\big] \tag{6.2}$$

实际上MC的目标更新是R_t，但是TD算法的更新是：$R_{t+1} + \gamma V_{t+1}$。在第3章我们知道：

$$
\begin{aligned}
V^\pi(s) =& E_\pi \{R_t \mid s_t = s\} && (6.3)\\
=& E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right\} \\
=& E_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s \right\} \\
=& E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) \middle| s_t = s \right\} && (6.4)
\end{aligned}
$$

概括的说就是MC方法用的是6.3作为目标的估计，而DP算法用6.4作为目标的估计。MC方法是一个估计因为在6.3中期望值是未知的；一个抽样的回报用来替换真实的期望回报。DP的目标是一个估计不是因为期望值——应该有环境模型完全提供，而是因为$V^\pi(s_{t+1})$是未知的，用当前的估计$V_t(s_{t+1})$来替换。TD的目标是估计因为两个原因：通过抽样获得期望值和用当前的估计V_t替换真实的V^π。因此，TD算法结合了MC算法的抽样和DP算法的boostraping.

图6.1表示了TD(0)算法的完整步骤，6.2表示了它的备份图。我们把TD和MC方法的更新叫做抽样备份因为他们提前一步考虑后继抽样的状态，使用后继状态的值以及回报计算备份值，然后相应的改变原始状态的值。抽样备份和全备份不同在于前者基于单个的抽样后继值恶如不是所有可能后继状态的完整的分布。

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s') - V(s)\big]$
    $s \leftarrow s'$
  until s is terminal

图 6.1: 估计V^π的表格时TD(0)算法

例6.1：开车回家每天你工作后开车回家，你尝试预测回家路上要花费多长时间。当你离开办公室时，你记下时间、星期几和一切相关的东西。比如你在星期五六点离开办公室，你估计可能回到家要花费30分钟时间。当你到你车上时是6:05，

图 6.2: TD(0)的备份图

你注意到开始下雨了。通常下雨交通会慢，因此你重新估计可能还需要35分钟，也就是总共40分钟。15分钟后你开过了高速路段，一切良好。当你离开到第二条公路时你把你的估计时间缩减为35分钟。不幸的是此时你呗一个开的很慢的卡车堵在了后面，道路太窄没办法超车。你不得不跟随卡着直到6:40你变到侧边街道。3分钟后你回家了。状态、时间和你的预测序列如下表：

	Elapsed Time	*Predicted*	*Predicted*
State	*(minutes)*	*Time to Go*	*Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

这个问题中回报是整个路程花费的时间。因此对于每个状态的回报就是从哪个状态开始实际需要的时间。状态值时每个状态期望的要走的时间。也即是上表中的第二列。

使用MC方法的一个问题是只有当你走完了全程（回到家）你才知道实际的回报，也就是离线的情况。直到最终结果已知才能开始学习吗？假设另一天你又估计你需要30分钟才能回家。但是你一出门就陷入交通堵塞。25分钟过去了你仍然在高速路缓缓前行。你现在估计可能再需要25分钟才能到家，总共50分钟。你必须到家之后才能增加你的估计值吗，也就是说必须回家之后你才知道状态的变化。对于MC方法就是如此，因为你不知道你实际需要多长时间回家，也就是你不知道回报是多少，因此你没办法做更新。

相反对于TD算法，你可以立即学习，改变你的初始估计到50分钟。考虑上面的问题，图6.4是和6.3同样的预测，除了它的改变策略是用TD规则。每一个错误和预测变化成比例，这就是TD预测。

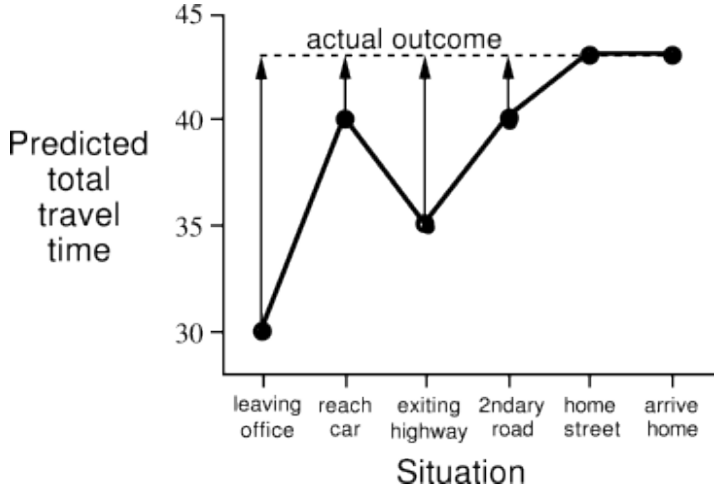


图 6.3: MC方法的建议修改时间

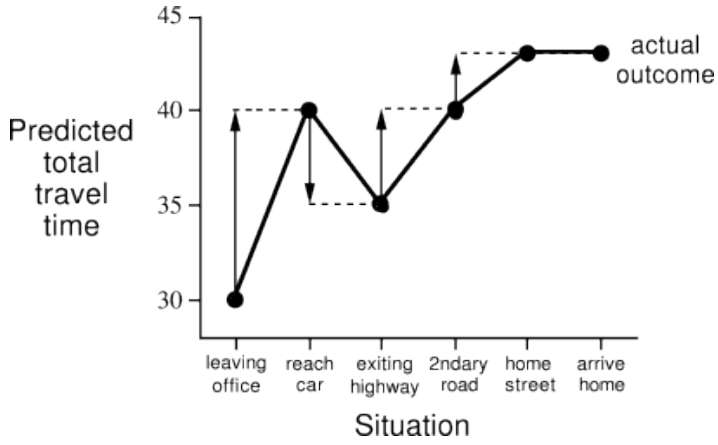


图 6.4: TD方法的建议修改时间

6.2 TD预测方法的优势

TD方法学习的估计值部分的依赖于其他的估计值。他们从猜测学习猜测——bootstap。这是一件好事吗？TD算法相对于MC和DP有什么优点？这部分简要的探讨一下。

显然相比于DP算法TD算法不需要环境模型，也就是回报以及下一状态的概率分布。

下一个TD算法相对于MC算法的显著优点是TD算法的实现天然上是在线的，增量式的。对于MC算法用户必须等到片段的结束，因为只有此时回报值才能已知，然而采用TD算法只需要等一个时间步长。出人意外的这通常被证明是关键的原因。一些应用具有很长的片段，以至于等到片段结束才学习显得太慢了。其他的应用是连续的任务根本就没有片段。最后，正如我们前面所说明的，一些MC方法对于采取实

验性行为的片段必须忽视或打折扣，这极大地降低了学习速度。TD算法则不容易受这些问题的影响因为它从每次过渡学习而不管采取什么后续策略。

TD算法合理吗？当然从下一个学习一个猜测而不需要等到实际的结构是方便的，但是我们仍然能保证收敛到正确的答案吗？令人幸福的是，答案是肯定的。对于任何固定的策略π，上述描述的算法已被证明当满足一定的条件时收敛到V^{π}，意味着对于一个足够小的常数步长参数，或者当步长参数依据随机近似条件（2.8）减小时，以概率1收敛。大多的收敛证明被用于基于表格形式的算法，但是也适用于一般线性函数逼近的例子。这些结果在后两张讨论。

TD和MC都收敛于正确的预测，那么问题是哪一个更快呢？这是一个开放的问题，没有数学上的证明哪一个收敛的更快。实际上对于一些随机策略的问题TD算法比constant-α MC算法收敛的更快。如下面的例子。

例6.2 ： 随机行走假设一个小的马尔科夫过程如图6.5所示。状态都从中间开始，然后以相等概率向左或者向右。当片段在右边结束得到+1的回报，其他的都为0.因为这个策略是没有折扣的并且是片段式的，状态真事的值函数就是从该状态开始在最右边终止的概率。因此真实的状态值函数是，从A到E——1/6,2/6,3/6,4/6,5/6.6.6展示了TD(0)算法的效果。6.7展示了TD算法和MC算法各自的误差。初始值$V(s) = 0.5$.

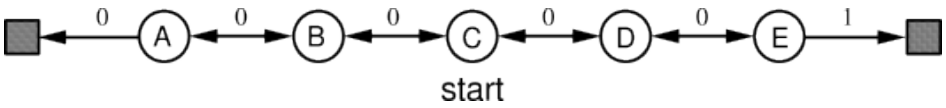


图 6.5: 产生随机行走的马尔科夫过程

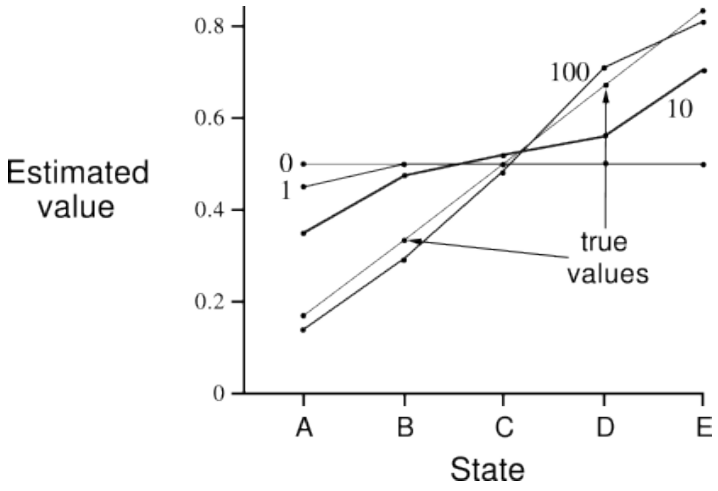


图 6.6: TD方法学习到的状态值

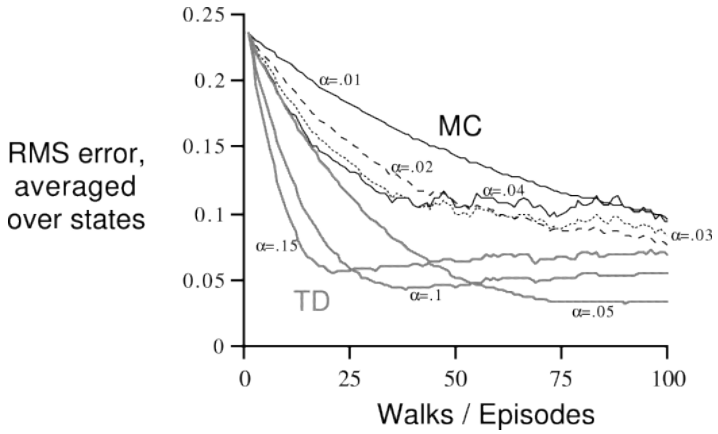


图 6.7: 不同步长不同方法的误差比较

6.3 TD(0)的优化

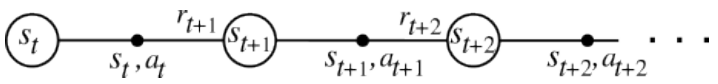
假设只能获得有限数量的经验，比如说10个片段或者100个时间步长。在这种情况下，一个通用的增量式学习方法的策略是是重复的学习这些经验直到方法收敛到一个答案。给定一个近似值函数V，在每一个时间步长t，当一个非终止状态被访问时，增量由6.1和6.2算出。但是值函数只更新一次，用所有增量的和。然后重复直到收敛。我们称之为批更新因为更新发生在处理完每一批训练数据之后。

在批更新下，TD(0)确定的收敛到一个答案并且独立于步长参数α,只要α足够小。MC方法也收敛到一个确定的答案，但是是一个不同的解。理解这两个解的不同将有助于我们理解这两种方法的不同。

6.4 Sarsa:在线策略TD控制

我们现在现在讨论使用TD预测算法解决控制问题。像往常一样，我们遵循GPI的框架，只是这次在预测的部分采用TD算法。正如MC方法一样，我们面临探索和利用的平衡问题，再一次同样的解决方法有2中：在线策略和离线策略更新。在这一部分我们讨论在线策略控制方法。

第一步是学习一个行为值函数而不是状态值函数。特别的，对于一个在线策略方法我们对于目前行为策略π下的所有状态所有行为估计$Q^{\pi}(s,a)$。这个可以通过和上面描述的学习V^{π}的TD算法来实现。回想一个包含交替的状态和状态-行为对的序列：



前面我们考虑从状态到状态的过渡并计算了状态的值函数。现在我们从状态-行

为对到状态-行为对的过渡，并且学习状态-行为对的值函数。形式上这些情况是相同的：他们都是带有回报过程的马尔科夫链。TD(0)算法下保证状态值收敛的理论也适用于行为值的对应算法：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big] \tag{6.5}$$

这个更新在每次非终止态的过渡时更新。如果s_{t+1}是终止的，那么$Q(s_{t+1}, a_{t+1})$被定义为0.这个规则使用一个事件的五元组的每一个元素$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$,这使得一个状态-动作对过渡到下一个状态动作对。这个五元组产生了这个算法的名字Sarsa。

设计一个基于Sarsa预测方法的在线控制策略是很直接的。在所有的在线策略算法中，我们对于每个行为策略π持续估计Q^{π}，然后同时改变π使之相对于$Q^{p}i$是贪婪的。Sarsa的一般控制图如图6.8所示：

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

图 6.8: Sarsa：一个在线策略TD控制算法

Sarsa算法的收敛性依赖于策略对Q的依赖性。比如，可以使用$\varepsilon - greedy$或者$\varepsilon - soft$策略。根据和Satinder Singh的私下交流，Sarsa依概率1收敛到最优策略和行为值函数只要状态对能被无限次访问，收敛性受限于贪婪策略，但是这个结论还没在文章中发表出来。

图6.9表示一个标准的网格世界，但是有一点不同：网格的中间有一股向上的侧风。标准的动作是上下左右，但是在中间区域合力作用的结果是下一个状态被往上移动，幅度的大小随着列的不同而不同。风力大小由列下面的值给出，表示为往上移动的单元数。

在图6.9中，S表示开始状态，G表示目标。风力的影响说明如下：比如当前位于G右侧的单元格，动作是向左，由于风力作用，所以最终达到的地方是G的上方。如图中红色箭头所示。图6.9表示了Sarsa的学习结果，可见曲线斜率越来越大，说明单位片段的步长减小了，换句话说学习的速度越来越快了。比如干刚开始的一个片段，从S到G可能要60个步长，经过学习和策略的改善使得决策更有效了，自然而然

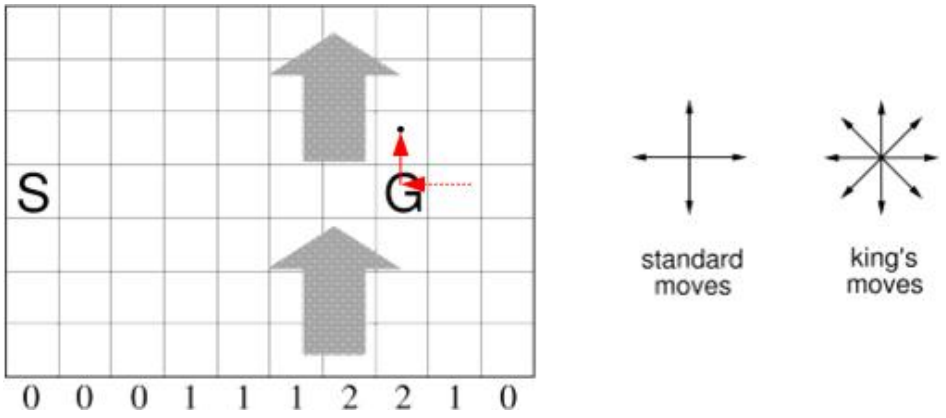


图 6.9: 有风力影响的网格世界问题

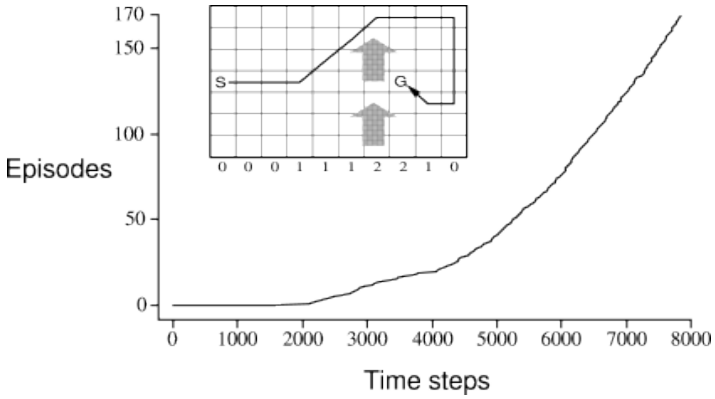


图 6.10: 应用Sarsa算法的结果

需要的步骤就少了。插图中表示的是用贪婪策略的最优解。这里要说明的是不能使用MC方法，因为MC方法要求每一次都要有回报，而对于该问题的某个片段可能由于决策问题而永远到不了目标，那么下一个片段的学习就无法继续。

6.5 Q-学习：离线策略TD控制

在增强学习领域一个重要的突破就是离线策略TD算法也就是Q-学习的出现（Watkins,1989）。它最简单的形式，一步Q学习，定义为：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big] \tag{6.6}$$

在这种情况下，学习的行为值函数Q直接近似到Q^*——最优额的行为值函数，独立于采取的策略。这极大的简化了算法的分析并且促使了早期的收敛证明。这个策略依然有效在于它决定了那个状态-行为对被访问和更新。但是正确的收敛所需要的是所有的对持续被更新。这是所有方法收敛的一个必要条件。Q-学习算法的步骤

如下：

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$;
  until s is terminal

图 6.11: Q-学习：一个离线策略TD控制算法

Q-学习的备份图是什么？规则6.6更新状态-动作对，因此最上面的节点，备份图的根节点必须很小且具有行为节点。行为节点的备份依然来自于行为节点，最大化下一个状态可能的行为值。其备份图见图6.13.

例6.6：悬崖行走网格世界的例子比较了Sarsa和Q-学习算法，突出了二者的区别。考虑图6.12的例子。这是一个标准的没有折扣，片段式，具有初始和结束状态，通常的行为产生上下左右的移动。每次过渡的回报都是-1除了浙西进入标记为“悬崖”区域的过渡。进入这个区域获得-100的回报并且把智能体拉回起始点。下面的图展示了用$\varepsilon - greedy$方法做行为选择的Q-学习和Sarsa的表现性能，$\varepsilon = 0.1$。经过一次过渡，Q学习学习到了最优策略，直接沿着悬崖的边缘向右走。不幸的是，这个结果会偶然的掉进悬崖因为$\varepsilon - greedy$选择的随机性。另一方面，Sarsa考虑了行为选择学习到了上面格子的一种时间更长但是更安全的路径。尽管Q-学习学习到了最优策略，他的在线性能不如Sarsa——学习一个大概的策略。当然，当ε逐渐减小时两种方法都能渐进收敛到最优策略。

6.6 行为-评判方法

行为-评判方法是TD算法，它具有一个独立的内存结构来明确的表示策略而独立于值函数。策略的结构被称为*actor*，因为它被用作选择行为，估计的值函数被称为*critic*，因为它评价来自于*actor*的行为。学习永远是基于策略的：评判的学习来自于当前*actor*采取的任意策略。批评采取一种TD误差的形式。这个标量的信号作为*critic*的唯一输出，同时用来指导*critic*和*actor*的学习，图6.14所示：

*critic-actor*方法是增强比较方法（section 2.8)思想到TD学习和所有增强学习问题的自然扩展。典型的，*critic*是一个状态值函数。在每一次行为选择之后，*critic*评估新的状态来决定情况是否比预想的好或者坏。这个评估就是TD误差：

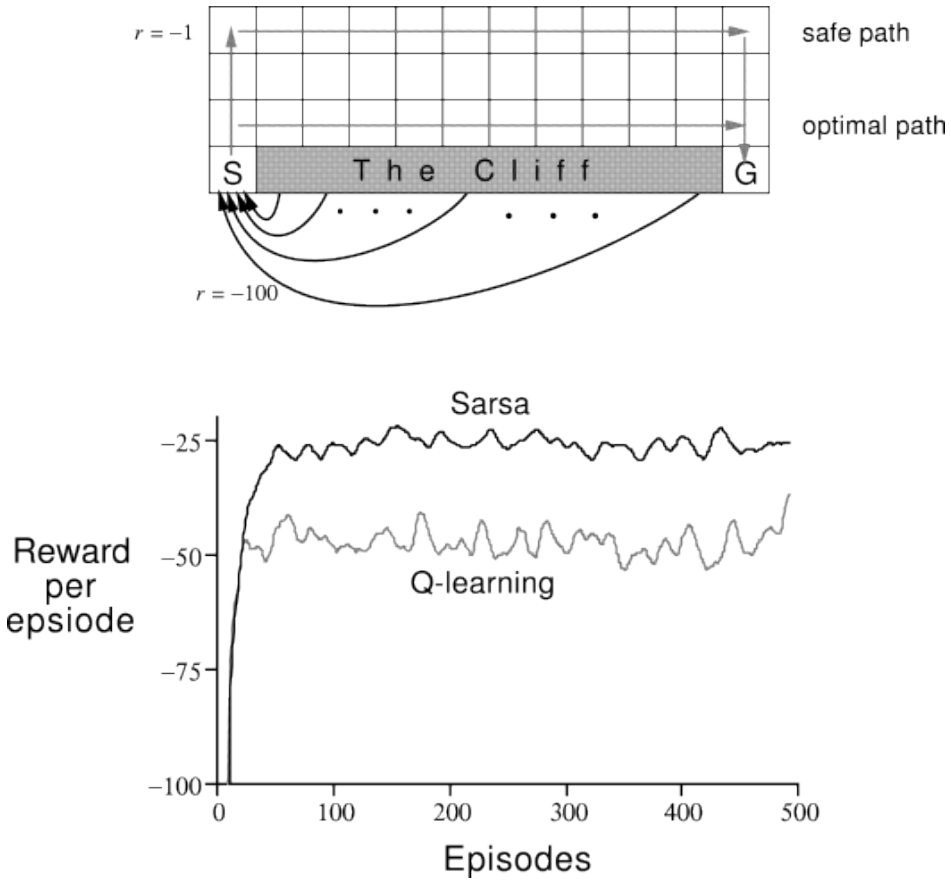


图 6.12: 悬崖行走问题，结果来自于单轮训练

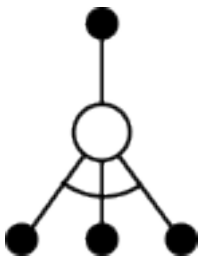


图 6.13: Q-学习的备份图

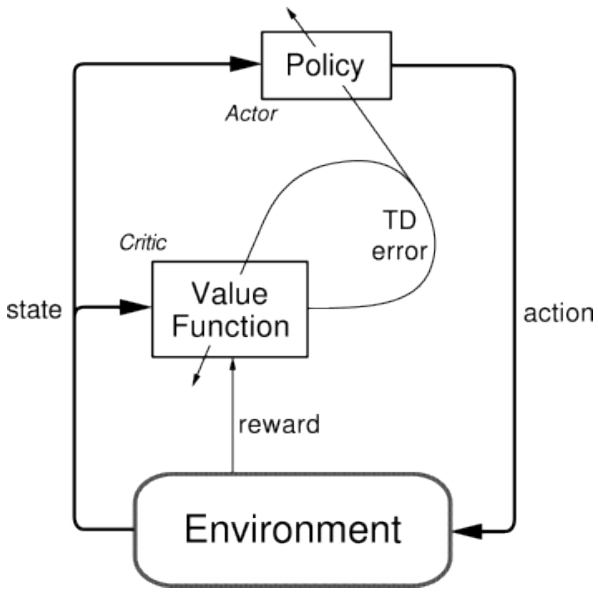


图 6.14: *critic-actor*架构

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

V 是由评判器得到的当前值函数。TD误差被用来改善选择的行为，s_t时刻选择的行为a_t.如果TD 误差是正的，意味着选择行为a_t的趋势要加强，然而如果TD误差是负的，意味着趋势应该减弱。假设行为用Gibbs softmax 方法产生：

$$\pi_t(s,a) = Pra_t = a|s_t = s = \frac{e^{P(s,a)}}{\sum_b e^{p(s,b)}}$$

$p(s,a)$表示t时刻actor的可修正的策略参数，指示了在每个状态s选择行为a的趋势。然后上面描述的加强或减弱可以通过增加或减小$p(s_t,a_t)$实现，比如通过：

$$p(s_t,a_t) \leftarrow p(s_t,a_t) + \beta\delta_t$$

β是另一个正的步长参数。

这只是*critic-actor*方法的一个例子。其他的变种使用不同的方式选择行为，或者利用下一章描述的资格迹一类的方法。另一类常用的变种，如增强比较方法，包含额外的因子来改变分配给采取动作的信誉度。例如，其中一个最常用的因子和选择行为a_t的概率负相关。产生如下的更新规则：

$$p(s_t,a_t) \leftarrow p(s_t,a_t) + \beta\delta_t\big[1 - \pi_t(s_t,a_t)\big]$$

这些问题早期被探索，主要用于立即回报的例子并且没有完全更新。

很多早期使用TD方法的学习系统是*critic-actor*方法。从那时开始，更多的注意力集中在学习值函数的方法和根据估计值唯一的决定策略。这些分歧都是历史事件了。例如，你可以想象在一个中间框架中值函数和独立的策略被学习。无论什么，*critic-actor*方法仍是现在的兴趣热点主要基于下面两个优点：

- 选择行为需要的计算少。
- 它可以学习一个明确的随机策略。

In addition, the separate actor in actor-critic methods makes them more appealing in some respects as psychological and biological models. In some cases it may also make it easier to impose domainspecific constraints on the set of allowed policies.

6.7 对于非折扣连续任务的R-学习

R-学习是一种离线策略控制方法，用于没有折扣和不能分成片段式任务的高级增强学习方法。在这个例子里用户可以在每个时间步长获得最大回报。一个策略π的值函数定义为在该策略下每个步长期望回报的均值，表示为：

$$\rho^{\pi} = \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} E_{\pi}\{r_t\}$$

假设过程是各态历经的（在任何策略下能够以非零概率到达任何状态）因此ρ^{π}不依赖于其实状态。从任何状态开始，从长远来看回报的均值是一样的，但是有一个过渡。一些状态暂时好于均值回报，其他接受的回报比均值差。这个过渡定义了状态值：

$$\tilde{V}^{\pi}(s) = \sum_{k=1}^{\infty} E_{\pi}\{r_{t+k} - \rho^{\pi} | s_t = s\}$$

行为状态对的值函数定义如下：

$$\tilde{V}^{\pi}(s,a) = \sum_{k=1}^{\infty} E_{\pi}\{r_{t+k} - \rho^{\pi} | s_t = s, a_t = a\}$$

我们称这些为相对值因为他们是相对于平均值。

在没有折扣的情况下，不同的优化方法之间区别不大。但是，对于多数实际的情况简单根据每个步长的均值回报决策已经足够了，换句话说，根据ρ^{π}。现在让我们考虑所有保留最大值ρ^{π}为最优的策略。

除了利用相对值，R-学习是基于离线GPI的一个标准的TD控制算法，更像Q-学习。它维护两个策略，一个行为策略和一个估计策略，加上一个行为值函数和一个估计平均回报。行为策略用于产生经验；它可能是关于行为值函数的$\varepsilon - greedy$策略。估计策略涉及GPI。他是行为值函数的贪婪策略。如果π是估计策略，那么行为值函数Q是$\tilde{Q}^{\pi}$的近似，平均回报ρ是ρ^{π}的近似。完整的算法由图6.15给出。

Initialize ρ and $Q(s,a)$, for all s, a, arbitrarily
Repeat forever:
$s \leftarrow$ current state
Choose action a in s using behavior policy (e.g., ε-greedy)
Take action a, observe r, s'
$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r - \rho + \max_{a'} Q(s',a') - Q(s,a)\right]$
If $Q(s,a) = \max_a Q(s,a)$, then:
$\rho \leftarrow \rho + \beta \left[r - \rho + \max_{a'} Q(s',a') - \max_a Q(s,a)\right]$

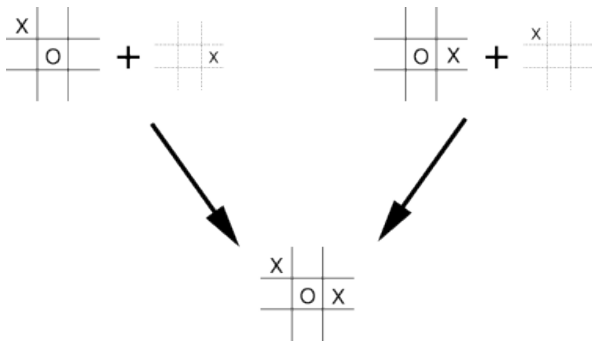
图 6.15: R学习

6.8 游戏，后继状态，和其他的特殊例子

在本书中我们试图展示一个统一的方法适应于广泛的一类任务，但是当然不可避免有一些特别的例子应该用特殊的方式更好的解决。例如，我们通用的方法实际行为值函数的学习，但是在第一章我们展示了一个TD算法用于学习井字游戏中的更像是状态值函数的东西。如果我们更仔细的看待那个问题，很闲那个学习的函数在通用意义下既不是状态值函数也不是行为值函数。传统的状态-值函数智能体估计那些有机会选择行为的状态，但是在井字游戏中的状态值函数在智能体做了移动之后才估计。我们称之为afterstates，相应的值函数叫做afterstates值函数。afterstates值函数对于我们对环境初始部分已知但是没必要完全知道的情况有用处。例如，在游戏中通常我们知道每走一步的立即回报。我们知道象棋每一步移动可能到的地方，但是我们不知道对手怎么回应。afterstates值函数是一种自然的利用这种知识的方法因此会产生更加有效果的学习。

在井字游戏中设计针对afterstates的算法更有效率是显而易见的。传统的行为值函数把位置和移动映射到一个估计值。但是很多位置-移动对产生相同的结果位置，如下面的例子：

在这个例子中位置-移动对不同但是产生了相同的“afterposition”，因此必须给予同样的值。一个传统的行为值函数必须单独的获得两个值函数，但是一个afterstate值函数会立即相等的评估二者。任意一个左边的位置-动作对的学习会立刻被转化到右边的对。



Afterstates在很多任务中都有出现，不仅仅是游戏。比如在排队问题中有类似把顾客分给服务员、拒绝顾客或者是放弃的信息的行为。在这种情况下行为实际上由他们的理解影响决定，是完全已知的。比如，在前面讲的访问控制排队，一个更有效率的学习方法是把动态环境分成行为的立即回报，这个是确定的和已知的，不知道的随机过程是顾客到达和离开的时间。Afterstates是行为之后空闲服务员的数量，但是事在随机过程产生下一个传统的状态之前。通过afterstates学习afterstates值函数将会是那些产生相同空闲服务员数量的行为分享经验。这会大大的缩减学习时间。

描述所有特殊问题相应的特殊的学习算法是不可能的。但是，这本书提出的原则具有广泛的适用范围。比如afterstates依然适用于GPI框架。

6.9 总结

这一章我们引入了一个新的学习方法，TD学习，并且展示了它如何被用于增强学习问题。像往常一样，我们把问题大体上分为预测问题和控制问题。TD算法是解决控制问题的MC方法的替代。在两种情况下，控制问题的扩展都借助了通用策略迭代的思想。正式这种近似策略和值函数的这种交互方法使得他们趋向最优解。

构成GPI的两个过程中，预测问题驱使值函数精确的预测出当前策略下的回报。另一个过程驱使策略根据当前的值函数局部的进行改善。当前一个过程机遇经验时，一个复杂的问题产生了——如何进行充分的探索。正如第5章，我们通过使用解决这个问题使用的on-policy和off-policy方法把TD 控制算法分成两类。Sarsa和*actor-critic*方法是在线的，Q-学习和R-学习是离线的方法。

本章中阐述的方法是增强学习中使用最广泛的。这大概是由于他们很大的简洁性：他们可以在线使用，计算量小，通过与环境交互学习经验；他们可以完全用单个方程在一台小的电脑程序上实现。下一章我们扩展这些算法，让他们略微复杂但是更重要和强大。所有的算法会保留他们的核心：他们能够在线处理经验，使用相对小的计算量，同时通过TD误差修正。目前章节中介绍的TD算法的特殊例子都应

该是单步的，表格式的无模型的TD算法。下面三章我们扩展到多步的形式，使用值函数近似的形式而不是表格，并且包括环境的模型（规划和动态规划的联系）。

最后，在这一章我们讨论TD算法都是在增强学习的环境中，实际上TD不局限于增强学习。他们是学习对动态系统做长期预测的通用方法。比如TD算法可以被用来预测金融数据、生命周期、选举结构、气候模式、动物行为、电站需求和顾客购物。只有TD算法被作为一种纯粹的预测方法，独立于在增强学习的使用，他们的理论特性才得到了很好的理解。即使如此，这些其他的TD学习方法潜在的应用没有被广泛的探索。

Part III

一个统一的视角

目前为止我们讨论了三大类解决增强学习问题的方法：动态规划，芒特卡罗方法和时间差分学习。尽管每一个是不同的，但他们实际上不是替代防方案，意味着你可以选一种或者另一种。通常同时使用几种不同的方法是很合理的，也是令人满意的，也就是说使用一种联合的方法。对于不同的任务或者任务不同的部分用户可能想要强调当中的一种方法，但是这些选择可以在使用方法时合理的决策，而不是在他们被设计的时候。在第3部分我用一个统一的视角来看待第2部分中展示的三种不同的初级的解决方法。

这部分介绍的统一性不是晦涩的类比。我们开发具体的算法，这些算法内涵了一种或者几种初级方法的核心思想。首先我们介绍资格迹的机制，泳统一MC和TD方法。然后我们引入函数近似，能够在状态和行为空间泛化。最后我们重新介绍环境模型来说明DP的重要性和启发式搜索。所有这些都可以作为联合方法的一部分协同的使用。

Chapter 7

资格迹

资格迹是增强学习的一个基本的机制。比如在流行的$TD(\lambda)$算法中，λ涉及到了资格迹的使用。几乎所有的时间差分算法比如Q-学习、Sarsa，都能被结合到资格迹来获得更一般的更有效率的方法。

有2种方法来看待资格迹。一种更理论的，也是我们这里要强调的，资格迹是TD到MC方法的桥梁。当TD算法用资格迹扩展后，它产生了一类方法，张成了一个谱，一端是MC方法，另一端是TD算法。在中间的立即方法通常比任何一种极端的方法都要好。在这种意义上资格迹提供了统一TD和MC方法的有价值并且有启迪性的方法。

另一种看待资格迹的观点更机械。从这个角度出发，资格迹是暂时的记录了一个事件的发生，比如对某个状态的访问或者采取一个行为。资格迹标记录了一些和事件有关联的内存参数，作为经历学习变化的资格。当TD误差产生，只有那些有资格的状态或者行为才被分配信用或者惩罚，因此资格迹在事件和训练信息的间隙之间建立了一个桥梁。正如TD算法本身，资格迹是暂时信用分配的一个基本机制。

为了简单的表述这个原因，对于资格迹更理论的观点叫做前向观点，更机械是的观点叫做后视观点。前向观点更有利于理解使用资格迹方法计算了什么，然后后视观点更适合我们建立对于算法本身的直觉。在本章中我们介绍两种观点然后建立他们理解上的等效性，也就是他们从2种观点描述了同一个算法。首先我们考虑预测问题和控制问题。也就是我们首先考虑在某个固定策略下，资格迹如何作为状态的函数用于预测回报。只有在预测环境中探索了资格迹的两种简介之后我们把这个思想扩展到行为值和控制问题。

7.1 $n-$步TD预测

处于MC和TD方法之间的空间是什么？考虑策略π产生的抽样的片段的值估计V^{π}。MC方

法对于从该状态开始知道片段结束的所有状态的观测回报序列进行备份。另一方面简单的TD方法，只备份下一个节点的回报，使用下一步状态的值作为剩余回报的代替。然后一种中间的方法是基于一个中间数目的回报：超过一个但是少于终止状态的数目。比如，一个两步的备份将会基于最开始的两个回报和两步后估计值备份。相似的，我们有三步备份，四步备份等等。图7.1画出了$n-step$备份的图，从最左边的一步——简单的TD备份——到最右边的MC方法的备份。

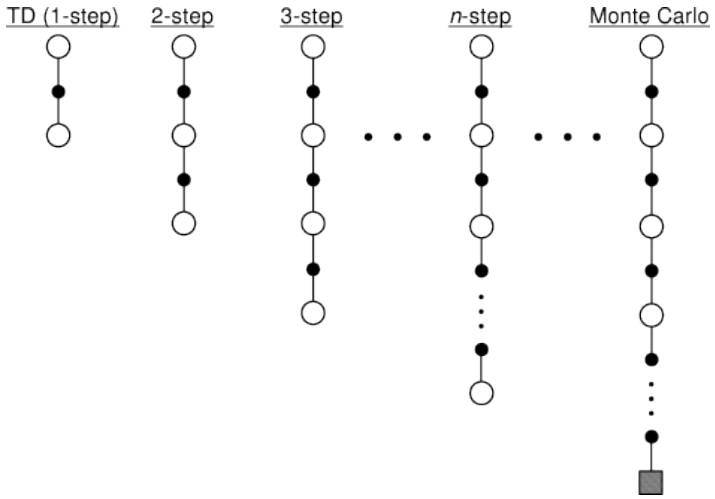


图 7.1: $n-step$备份图

这种使用$n-step$备份的方法仍然是TD方法因为他们依然依据它和后面估计的差别来改变早期的估计。现在后面的估计不是不是一步的了，而是n步后。把TD算法扩展到$n-step$的情形叫做做$n-step$ TD方法。前面介绍的都用的是一步备份，因此我们称之为一步TD方法。

更加正式的，考虑一个状态s_t之后的状态回报序列$s_t, r_{t+1}, s_{t+1}, r_{t+2}, \cdots, r_T, S_T$。我们知道在MC 备份中$V^\pi(s_t)$的估计$V_t(s_t)$是由整个回报更新的：

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$$

T 是片段的最后一个时间步长。让我们称这个数为备份的目标。鉴于MC备份的目标是期望回报。在一步备份中目标就是第一个回报加上折扣的下一个状态的估计值：

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$$

这是有道理的因为$\gamma V_t(s_{t+1})$代表了剩余的项：$\gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$。相应的两步目标是：

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$

这里的$\gamma^2 V_t(s_{t+2})$代表了项：$\gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$。一般的，$n$步目标是：

$$R_t^n = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) \tag{7.1}$$

这个数有时被叫做“修正的n步缩减回报”，因为回报在n步之后被缩减了，然后通过加上第n个下一个状态的的估计值来近似缩减的部分。这个术语是描述性的但是有点长。替换的我们用$R_t^{(n)}$表示时间t的n步回报。

当然，如果片段结束少于n个步长，那么缩减的n步回报发生在片段的结束，导致传统的完全回报。换句话说，如果$T - t < n$,那么$R_t^{(n)} = R_t$。因此，往后的n步总是返回完整的回报，一个无限步长的回报总是一个完整的回报。这个定义使得我们可以把MC 方法看做是无限步长回报的一个特殊例子。这与我们把片段式任务和连续任务等价用的技巧是一样的。在哪里我们把终止状态看做是一个状态，这个状态总是过渡到自己并且回报为0。利用这个技巧，所有n步回报持续到终止状态或者超过终止状态和完整的回报都是一样的。

相应的，对于一个n步的备份，对于$V_t(s_t)$的增量式：

$$\Delta V_t(s_t) = \alpha \Big[R_t^{(n)} - V_t(s_t) \Big]$$

α是一个正的步长参数。当然，对于其他状态的估计值的增量为0.我们把n步备份定义为一个增量，而不是一个直接的更新规则，就像我们之前做的一样，这是为了区分两种不同的更新方式。对于在线更新，更新在片段内完成，一旦增量被计算就更新。在这种情况下我们有：$V_{t+1}(s) = V_t(s) + \Delta V_t(s)$,对于所有的$s \in \mathcal{S}$。这是前面考虑的更新方式。对于离线更新，增量被向前累加不用于改变估计值，直到片段的结束。在这种情况下，$V_t(s)$在一个片段内是一个常量。如果在当前片段的值为$V(s)$，那么在下一个片段它的新值时$V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$。

n 步回报的期望值作为真实值函数的一个近似确保以某种方式被改善。对于任何的V，n步回报的期望值保证了是比V更好的一个估计。也就是在新的估计下的最大误差保证小于等于γ^n乘以在V下的最大误差：

$$\max_s \Big| E_\pi \big\{ R_t^{(n)} \big| s_t = s \big\} - V^\pi(s) \Big| \leq \gamma^n \max_s \Big| V(s) - V^\pi(s) \Big| \tag{7.2}$$

这叫做N步回报的误差减小特性。因为这个特性，用户可以证明在合适的技术条件下使用n步备份的在线或者离线TD预测都收敛于正确的预测。因此n步TD方法形成了一个有效的方法族，一步TD算法和MC方法只是极限的例子。

尽管如此，n步TD方法很少被使用因为他们实现起来比较困难。计算n步回报需要等待n步观察到结果的回报和状态。对于较大的n,这是有问题的，尤其对于控制的

应用。n步TD算法的重要性主要在于理论和对于相关的易于实现的方法的理解。下面几节我们会使用n步TD方法的思想来解释和证明根轨迹方法是正确的。

7.2 直视TD(λ)

备份不仅可以通过n步回报来确定，也可以是n步回报的均值。比如，可以做这样一个回报的备份－－它是一般是2步回报一半是4步回报：$R_t^{ave} = 1/2R_t^{(2)} + 1/2R_t^{(4)}$.任何一个回报集合都能这样被平均，甚至是无限集合，只要是回报成分的权值是正的并且和为1.总的回报拥有和单个n步回报相似的误差减小性质（7.2）,因此可以利用这个性质来构造备份保证一定的收敛性。平均产生了大量的新的一类算法。比如，用户可以平均一步和无限步备份来获得另一种关联TD算法和MC方法的方式。原则上，用户甚至可以把基于经验的备份和DP备份平均来获得一个简单的基于经验的和基于模型的方法的结合。（第9章）

一个平均更简单的备份成分的备份叫做一个复杂的备份。一个复杂备份的备份图包含所有的组成成分的备份图，在他们上面有一条横线，下面是权重分量。比如上面提到的一半2步备份一半4步备份的备份图如下：

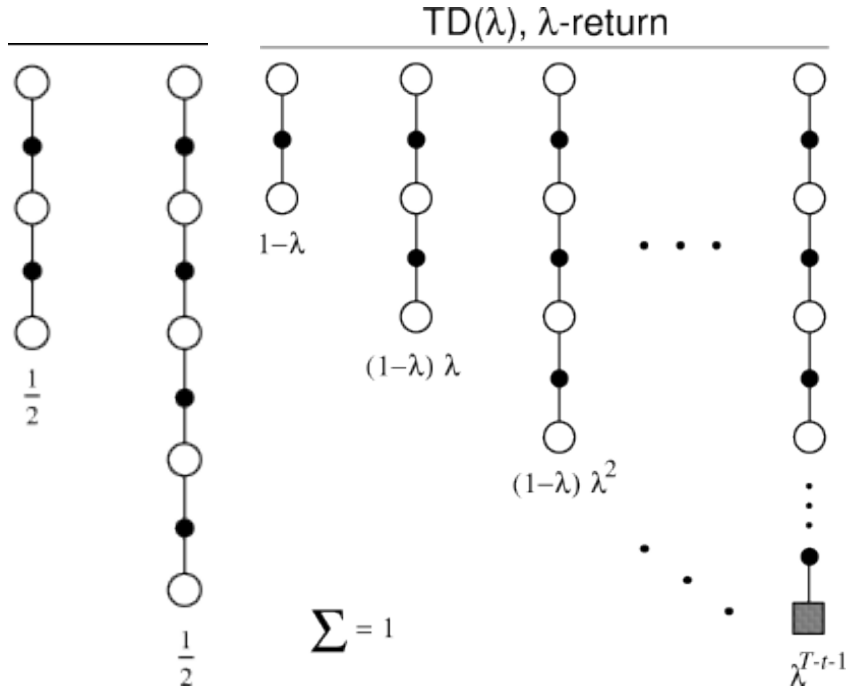


图 7.2: $TD(\lambda)$备份图

$TD(\lambda)$算法可以被理解成事一种特殊的平均n步备份的方式。这种平均包含所有n步的备份，每一个的权重都和$\lambda^{n-1} 0 \leq \lambda \leq 1$ 成比例(图7.2).标准化因子$1-\lambda$保证

权重和为1.结果的备份是一个回报，我们叫做λ回报，定义为：

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)}$$

图7.3阐明了权重序列。一步回报给予最大的权重$1-\lambda$；两步回报给出次大的$(1-\lambda)\lambda$；三步回报的权值为$(1-\lambda)\lambda^2$等等。权重随着每一步通过λ衰减。等到达终止状态后，所有后续的n不回报都等于R_t。如果我们可以把这些从主要项和中分离出去，产生：

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)} + \lambda^{T-t-1}R_t \tag{7.3}$$

这个方程是当$\lambda = 1$变得更清晰。在这种情况下主要想的和为0，剩余项减小为传统的回报R_t，因此当$\lambda = 1$,回顾到λ−回报就和我们称之为constant-α MC的MC算法一样了（6.1）.另一方面，当$\lambda = 0$,λ−回报变成$R_t^{(1)}$，一步回报。因此当$\lambda = 0$时，根据λ−回报的备份和一步TD方法的备份是一样的。

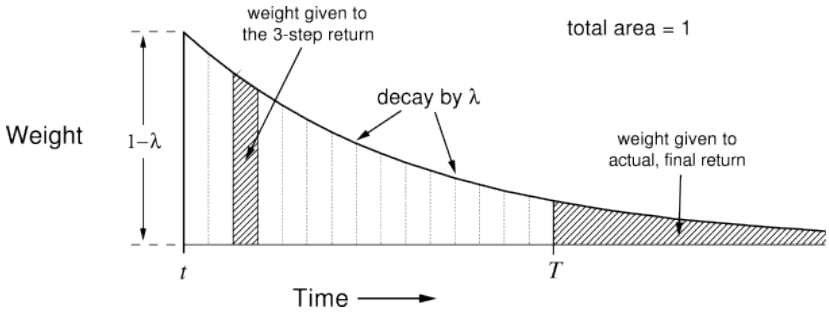


图 7.3: λ−回报中每个n步回报的权值

我们定义的λ−回报算法是一种用$\lambda - return$执行备份的算法。在每一步，它计算一个在哪一步出现的状态的增量$\Delta V_t(s_t)$：

$$\Delta V_t(s_t) = \alpha\big[R_t^\lambda - V_t(s_t)\big] \tag{7.4}$$

(对于其他的状态$s \neq s_t$当然增量$\Delta V_t(s) = 0$)。就像n步TD算法一样，更新也分为在线和离线两种方式。

我们目前采取的方法佳作理论上的，或者前向的学习的观点。对于每一个访问的状态，我们及时查看将来的状态并决定如何最好的结合他们。我们可以设想我们骑在一个状态流上，向前看每一个状态来决定它的更新，正如7.4一样。当我们往前看完了并且更新了一个状态，我们转移到下一个状态并且再也和前面的状态无关。

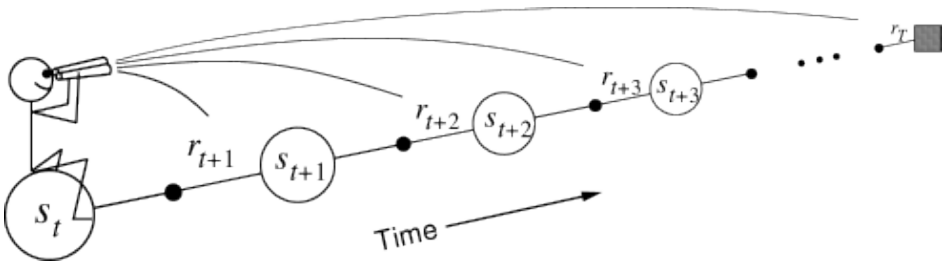


图 7.4: 理论的观点：我们通过观察未来的回报和状态来决定如何更新每一个状态

λ–回报算法是在TD（λ）中使用资格迹的一个前向观点的基础。实际上，后面的章节我们会讲到在离线的情况下，λ–回报算法就是TD(λ)算法。λ–回报和TD(λ)方法利用λ参数完成从一步TD到MC方法的变换。这个变换的完成的具体方式是有意思的，但是和n步方法通过改变n的方式没有明显的好坏。最终的，用λ的方式混合n步备份的强有力的动机通过一个简单的算法－－TD(λ)－－实现了。这是一个机制的问题而不是一个理论问题。下面几节我们会建立TD(λ)中资格迹的机制。

基于λ–回报的随机行走问题

图7.5展示了用λ–回报求解具有19个状态的随机行走问题的性能。注意到当λ取到中间值时效果最好。

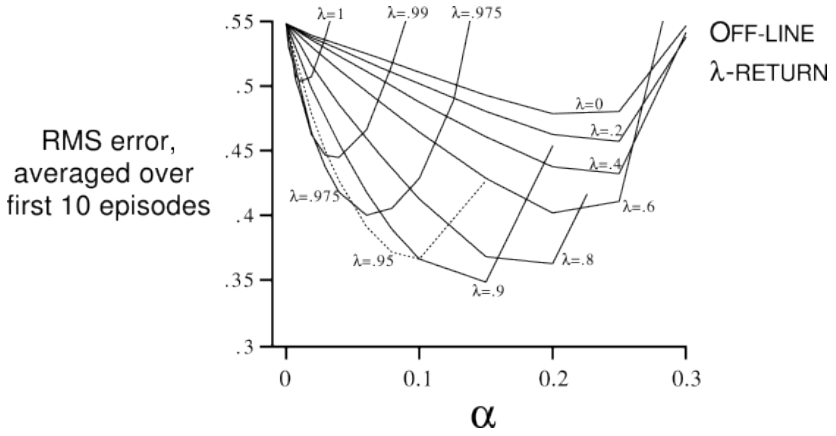


图 7.5: λ–回报算法在19个状态的随机行走问题上的表现

7.3 $TD(\lambda)$的后视观点

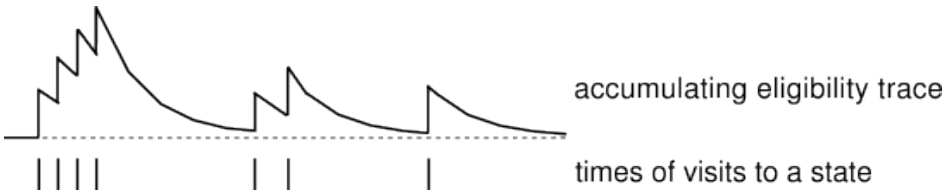
上一节中我们展示了如何用前向的观点来看待$TD(\lambda)$算法。相反的这一节我们定义机械的定义$TD(\lambda)$，下一节我们展示这个机制可以正确的实现前向观点。这种机械式的、后视的$TD(\lambda)$的观点是有益的因为他从概念上和计算上都很简单。特别的，前向观点本身不是能够实现的因为它的非因果性，在每一步使用的这是可能是很多

步之后发生的。后视观点提供一种因果的，增量的的近似前向观点的机制，并且在离线的情况下，可以精确的获得。

在$TD(\lambda)$的后视观点中，有一个额外的联系每一个状态的内存变量——资格迹。在时间t状态s的资格迹表示为$e_t(s) \in (R)^+$.在每一步，所有状态的资格迹通过$\gamma\lambda$，在该步中被访问的资格迹被增加1：

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & if\ s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & if\ s = s_t; \end{cases} \tag{7.5}$$

对于所有的非终止状态s，γ是折扣率，λ是前面介绍的参数。自此以后，我们称λ为迹衰减参数。这种资格迹叫做累积迹因为当状态被访问时它不断累积，然后当状态不被访问时迹渐渐的衰减，如下图所示：



在任何时间，迹记录着最近被访问的状态，这里的“最近”由项$\gamma\lambda$定义。迹指示了当一个增强信号发生时每个状态有资格经历学习变化的程度。我们考虑的增强事件是每时每刻的一步TD误差。比如状态值函数的TD误差：

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{7.6}$$

对于后视的$TD(\lambda)$，全局的TD误差信号成比例的触发了对所有近期访问状态的更新，用他们的非0迹表示：

$$\Delta V_t(s) = \alpha\delta_t e_t(s) \quad for\ all\ s \in \mathcal{S} \tag{7.7}$$

像往常一样，这些增量可以每一步更新形成在线算法，或者保存起来直到一个片段的结束产生一个离线算法。在任何一种情况下，方程7.5-7.7提供了$TD(\lambda)$算法的机械的定义。一个完整的在线$TD(\lambda)$算法表示如图7.6。

后视的$TD(\lambda)$是及时的往后看。在每个时刻我们查看当前的TD误差并且根据他在那个时候的状态资格迹把它向后分配给每一个先前的状态。我们可以假想我们骑在一个状态流上，计算TD误差，然后把他们分配到前面的每一个访问的状态，如图7.7所示，这里TD误差和资格迹结合在一起，通过7.7更新。

为了更好地理解往后看的思想，考虑每个不同的λ的影响。如果$\lambda = 0$，那么

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha \delta e(s)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
  until s is terminal

图 7.6: 在线的表格式的$TD(\lambda)$

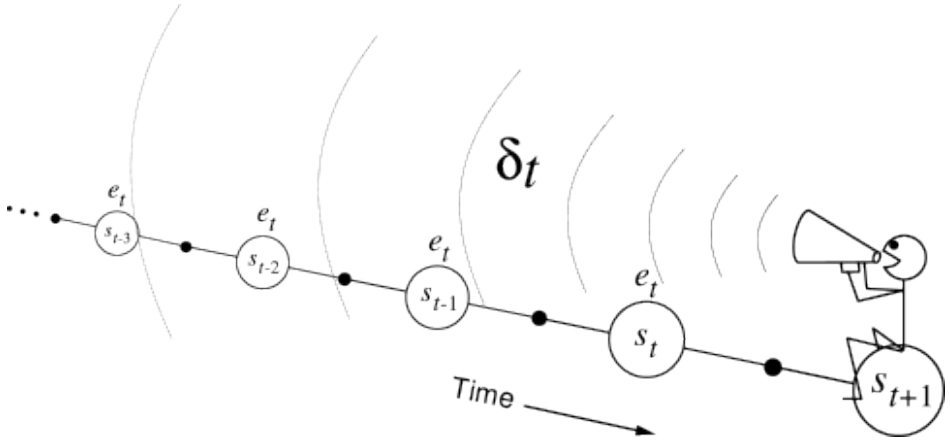


图 7.7: 向后看的机械式的观点，每一次更新依赖于当前的TD误差和过去事件的迹

由式7.5得到所有的迹为0除了对应于s_t的迹。因此$TD(\lambda)$更新7.7简化为简单的TD规则6.2.叫做TD(0).就图7.7而言，TD(0)就是当前状态的一个状态被TD误差改变。对于大的λ，但是不大于1，更多的前面的状态被改变，但是越远的状态改变越小因为它的资格迹更小，正如图所示。我们成为更早的状态得到TD误差更少的信用值。

如果$\lambda = 1$，给予早期状态的信用值每步衰减γ。这刚好是我们获得MC行为所做的事。比如，记住TD误差δ_t包含一个没有折扣的项r_{t+1}，当过去k步后他需要被打折扣r^k，就像任何返回的回报，这正是资格迹衰减获得的。如果$\lambda = 1, \gamma = 1$，此时资格迹始终不衰减。这时候方法的行为像是一个不打折扣的MC方法。如果$\lambda = 1$，算法也被叫做TD(1)。

TD(1)是实现MC算法的一种方法，更普遍比起前面介绍的那些方法，并且极大地增加了算法的适用范围。尽管早期的MC方法仅局限于片段式任务，TD(1)能够被用于不打折的连续的任务。更重要的是TD(1)能够增量式的在线实现。MC方法的一个劣势是它直到片段结束才学习到东西。比如，MC控制方法做了一些很糟糕的决策产生了不好的回报但是片段没有结束，那么智能体做那个决策的趋势在片段之内不会衰减。在线的TD(1)，另一方面，以n步TD的形式从不完整的正在进行的片段学习，这里的n步是从开始直到当前的状态。如果一些通常好的或不好的在片段之内发生了，基于TD(1)的控制算法可以立即学习并且在同一个片段内改变他们的行为。

7.4 前向观点和后项观点的等价

这节我们将证明离线的$TD(\lambda)$获得和离线的λ–回报算法相同的权值更新，正如上面定义的。在这个意义上我们统一了前向（理论上）后向（机械式）的$TD(\lambda)$的视角。用$\Delta V_t^{\lambda}(s_t)$根据$\lambda$–回报算法（7.4）获得的$V(s_t)$在时间$t$ 的更新，用$\Delta V_t^{TD}(s)$表示根据$TD(\lambda)$（式7.7给出）机械式定义得到的t 时刻状态s的更新。然后我们的目标是证明在整个片段上对于两个算法的所有更新的和是相等的：

$$\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=1}^{T-1} \Delta V_t^{\lambda}(s_t)\mathcal{I}_{ss_t} \quad forall s \in \mathcal{S} \tag{7.8}$$

$\mathcal{I}_{ss_t}$是一个指示函数，当$s = s_t$时等于1，否则等于0.

首先注意累积的资格迹可以明确的写成（非迭代的）：

$$e_t(s) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k}\mathcal{I}_{ss_k}$$

因此式7.8的左边可以写成：

$$\sum_{t=0}^{T-1}\Delta V_t^{TD}(s) = \sum_{t=0}^{T-1}\alpha\delta_t\sum_{k=0}^{t}(\gamma\lambda)^{t-k}\mathcal{I}_{ss_k} \tag{7.9}$$

$$= \sum_{k=0}^{T-1}\alpha\sum_{t=0}^{k}(\gamma\lambda)^{k-t}\mathcal{I}_{ss_t}\delta_k \tag{7.10}$$

$$= \sum_{t=0}^{T-1}\alpha\sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\mathcal{I}_{ss_t}\delta_k \tag{7.11}$$

$$= \sum_{t=0}^{T-1}\alpha\mathcal{I}_{ss_t}\sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k \tag{7.12}$$

现在我们计算式7.8的右边。考虑λ–回报算法的一个单独的更新：

$$\begin{aligned}\frac{1}{\alpha}\Delta V_t^{\lambda}(s_t) =& R_t^{\lambda} - V_t(s_t)\\ =& -V_t(s_t) \quad +(1-\lambda)\lambda^0\big[r_{t+1}+\gamma V_t(s_{t+1})\big]\\ &+(1-\lambda)\lambda^1\big[r_{t+1}+\gamma r_{t+2}+\gamma^2 V_t(s_{t+2})\big]\\ &+(1-\lambda)\lambda^2\big[r_{t+1}+\gamma r_{t+2}++\gamma^2 r_{t+3}+\gamma^3 V_t(s_{t+3})\big]\end{aligned}$$

检查括号里面的第一列——所有的r_{t+1}他们的权重因子是$1-\lambda$乘以权重指数λ。结果是所有权重因子的和为1.因此我们可以抽出所有第一列得到无权重的项r_{t+1}。相似的技巧用于抽取括号里的第二列，从第二行开始和为$\gamma\lambda r_{t+2}$。对每一列重复，得到：

$$
\begin{aligned}
\frac{1}{\alpha}\Delta V_t^{\lambda}(s_t) = &- V_t(s_t) \\
&+ (\gamma\lambda)^0\big[r_{t+1} + \gamma V_t(s_{t+1}) - \gamma\lambda V_t(s_{t+1})\big] \\
&+ (\gamma\lambda)^1\big[r_{t+2} + \gamma V_t(s_{t+2}) - \gamma\lambda V_t(s_{t+2})\big] \\
&+ (\gamma\lambda)^2\big[r_{t+3} + \gamma V_t(s_{t+3}) - \gamma\lambda V_t(s_{t+3})\big] \\
&\vdots \\
=&(\gamma\lambda)^0\big[r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\big] \\
&+ (\gamma\lambda)^1\big[r_{t+2} + \gamma V_t(s_{t+2}) - V_t(s_{t+1})\big] \\
&+ (\gamma\lambda)^2\big[r_{t+3} + \gamma V_t(s_{t+3}) - V_t(s_{t+3})\big] \\
&\vdots \\
\approx &\sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k \\
\approx &\sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k
\end{aligned}
$$

当上面的情况为离线更新的情况时近似是精确的，此时对于所有t ,V_t都是一样的。最后一步是精确的，因为所有省略的项δ_k包括了所有的终止状态之后的虚拟的状态。所有这些步具有零回报和零值；因此终止状态之后的δ也是0.因此，我们证明了在离线情况下式7.8右边可以写成：

$$
\sum_{t=0}^{T-1}\Delta V_t^{\lambda}(s_t)\mathcal{I}_{ss_t} = \sum_{t=0}^{T-1}\alpha\mathcal{I}_{ss_t}\sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k
$$

该式和式7.9一样，这就证明了式7.8。

在离线更新的情况下，只要α足够小上面的近似就很接近，因此V_t在片段之内变化很小。甚至是在在线的情况下我们可以期望$TD(\lambda)$的更新和$\lambda-$回报算法是相似的。

现在让我们假设在片段之内增量足够小，以至于在一个片段上$TD(\lambda)$的更新和$\lambda-$回报算法的更新基本上是一样的。仍然留有有趣的问题——一把片段之内发生了什么。考虑一个片段之间时间$t+k$更新的状态s_t。在在线$TD(\lambda)$下，在$t+k$的影响相当于我们做了一个$\lambda-$回报，把观察到的状态作为终止状态，但是伴有一个非0 的回报值等于当前的估计值。这种关系一步一步被维持下来。

用$TD(\lambda)$求解随机行走因为离线的$TD(\lambda)$等价于$\lambda-$回报算法，我们已经做过离

线的$TD(\lambda)$在19个状态的最忌行走问题；如图7.5所示。图7.8表示了在线$TD(\lambda)$的对比结果。注意到在更广泛的参数集上在线的算法表现的更好。在线算法一般都有这个规律。

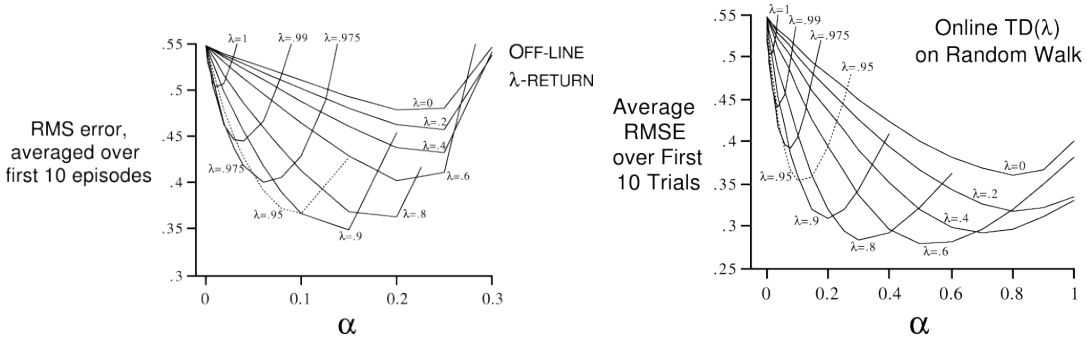


图 7.8: 在线$TD(\lambda)$求解行走问题

7.5 Sarsa(λ)

如何资格迹不仅仅是被用于预测，如TD(λ)，而是控制呢？像往常一样，一个主流的大众的想法是简单的学习行为值函数$Q_t(s,a)$，而不是状态值函数$V_t(s)$。在这部分我们介绍资格迹如何以一种直接的方式和Sarsa算法结合来产生一个在线策略的TD控制方法。资格迹版本的Sarsa叫做$Sarsa(\lambda)$，前面最原始的版本我们从此之后叫做一步Sarsa。

$Sarsa(\lambda)$的思想是应用TD(λ)预测方法到状态-动作对而不是状态。然后显然我们不仅需要一个每一个状态的迹，而是对于每一个状态-动作对。用$e_t(s,a)$表示状态动作对(s,a)的迹。其他方面这个方法就像TD(λ)，用状态-动作对替换状态变量，也就是$Q(s,a)$替换$V(s)$，$e_t(s,a)$替换$e_t(s)$：

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a), \quad for\ all s, a$$

这里:

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s,a)$$

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & if s = s_t\ and\ a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & 0therwise \end{cases} \tag{7.13}$$

图7.9展示了$Sarsa(\lambda)$的备份图。注意它和$TD(\lambda)$算法的相似性（图7.2）。第一

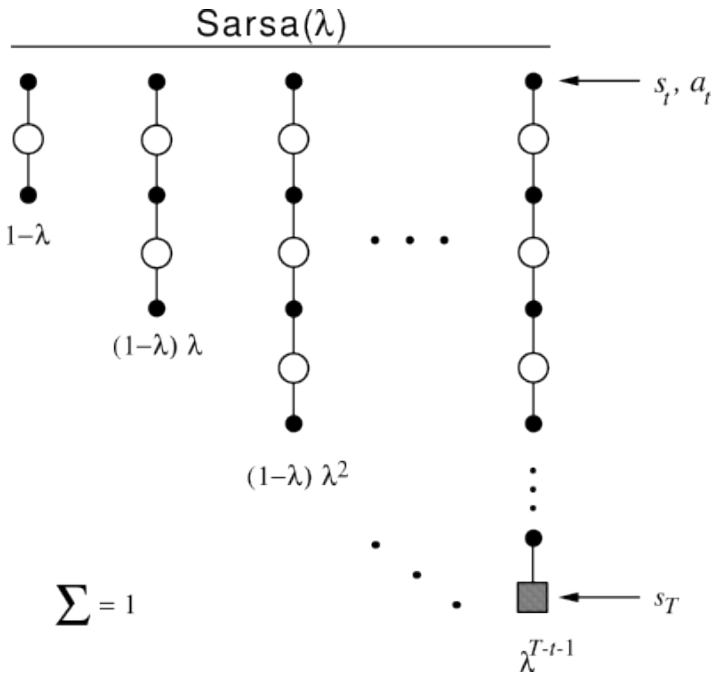


图 7.9: $Sarsa(\lambda)$的备份图

个备份往前看一整步到下一个状态-动作对，第二个往前看两步等等。最后一个备份基于完整的回报。每一步回报的备份和$TD(\lambda)$和λ–回报算法的一样。

一步Sarsa和$Sarsa(\lambda)$都是在线策略算法，意味着首先近似$Q^\pi(s,a)$——当期策略π的行为值函数，然后根据当前策略的近似值函数逐步的改善策略。策略改进可以有很多种方式，整本书里可以看到。比如，最简单的方式是使用针对当前行为值估计的$\varepsilon - greedy$策略。图7.10展示了完整的$Sarda(\lambda)$算法。

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

图 7.10: 表格式$Sarsa(\lambda)$

网格世界的迹资格迹的使用能够极大地增加控制算法的效率。原因可以从

图7.11中看出。第一个面板展示了在一个片段里智能体选择的路径，在一个高回报的地方结束，表示为*。在这个例子中，所有值初始为0，所有的回报为0除了最后的*位置有一个非零的正的回报。其他的两幅图展示了一步的Sarsa和$Sarsa(\lambda)$方法中哪一个行为值被加强。一步的方法只加强序列中最后一个导致高回报的行为值，然后迹方法加强了序列中的很多行为。加强的程度（表示为箭头的大小）随着步伐衰减。在这个例子中$\gamma = 1, \lambda = 0.9$。

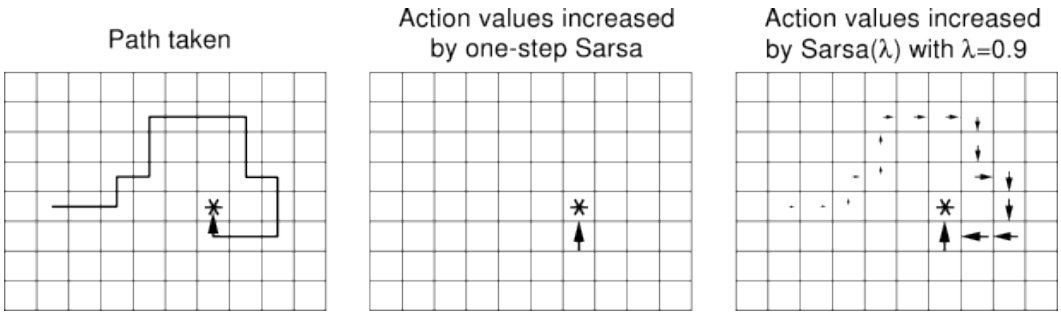


图 7.11: 网格世界的例子中由于使用了资格迹加快了学习的速度

7.6 $Q(\lambda)$

提出了两种结合资格迹和Q-学习的方法；我们称之为Watkins's $Q(\lambda)$和Peng's $Q(\lambda)$，根据提出他们的研究者。首先我们介绍Watkins's $Q(\lambda)$。

回想Q -学习是一种离线策略方法，意味着学习的策略和用于选择行为的策略不需要一样。特别的，Q-学习学习的是贪婪策略但是典型的情况下它遵从一个涉及探索行为的策略——随机选择的一个对于Q_t来说是次优的行为。因为这样，资格迹的引入需要格外仔细。

假设我们正在备份时间t的状态行为对s_t, a_t。假设下两个时间步长智能体选择贪婪策略，但是第三个选择一个探索的非贪婪的策略。在学习贪婪策略在s_t, a_t的值的时候可以使用后续的经验，只要后续的是贪婪策略。因此我们可以使用一步或者两步的回报，但是三步回报不可以。所有$n \geq 3$的$n-$步回报已经和贪婪策略没有必要的联系。

因此，不像是$TD(\lambda)$或者$Sarsa(\lambda)$，Watkins's $Q(\lambda)$在备份中不提前看直到片段的结束。它只考虑到最近的探索行为。可是除了这个区别，Watkins's $Q(\lambda)$和$TD(\lambda)$或者$Sarsa(\lambda)$很像。他们往前看直到片段结束，然而$Q(\lambda)$往前看直到第一个探索行为发生，如果片段内没有探索行为，那么就看到片段结束。实际上，精确来说，一步的Q-学习和Watkins's $Q(\lambda)$都看到第一次探索的前一步，利用行为值的知识。比如，假设第一个行为a_{t+1}是探索的。Watkins's $Q(\lambda)$仍然会对$Q_t(s_t, a_t)$做一步更新

到$r_{t+1} + \gamma \max_a Q_t(s_{t+1,a})$。通常，如果$a_{t+n}$是第一个探索行为，那么最长的备份是：

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_a Q_t(s_{t+n}, a)$$

假设为离线更新，图7.12展示了Watkins's $Q(\lambda)$算法前向观点的备份图，展示了所有组件的备份。

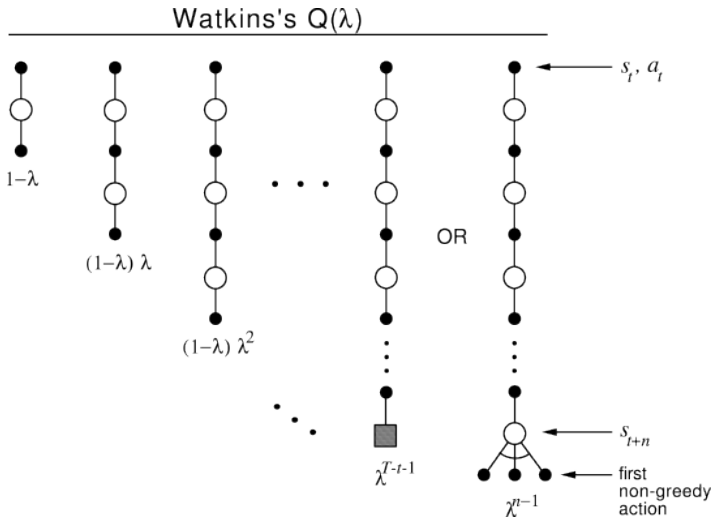


图 7.12: Watkins's $Q(\lambda)$的备份图，组件的备份序列要么结束于片段的末尾，要么是第一个非贪婪行为，取决于哪一个先出现

Watkins's $Q(\lambda)$的后视观点也很简单。资格迹和Sarsa(λ)的使用一样，除了当非贪婪行为被选择时设为0。迹更新最好被认为发生在两步之内。首先，所有的状态-行为对要么按照$\gamma\lambda$衰减，要么被设为0当探索行为发生。第二，当前状态和行为的迹被增加1。整体上的结果是：

$$e_t(s,a) = \mathcal{I}_{ss_t} \cdot \mathcal{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & if\ Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a); \\ 0 & otherwise, \end{cases}$$

算法的其余部分

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

其中，

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$$

图7.13以伪代码的形式展示了完整的算法

Initialize $Q(s,a)$ arbitrarily and $e(s,a)=0$, for all s,a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $a^* \leftarrow \arg\max_b Q(s',b)$ (if a' ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s',a^*) - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s,a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                  else $e(s,a) \leftarrow 0$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

图 7.13: 表格形式的Watkins's $Q(\lambda)$算法

不幸的是，每次遇到探索行为时砍掉迹失去了很多实用资格迹的优势。如果探索行为时有发生，在容易早熟的情况下，超过一步或者两步的备份很少出现，此时的学习将会比一步的Q-学习快一点点。Peng's $Q(\lambda)$是$Q(\lambda)$的一个替换版本，试图消除这个弊端。Peng's $Q(\lambda)$可以看作是$Sarsa(\lambda)$和Watkins's $Q(\lambda)$的一个混合体。

概念上，Peng's $Q(\lambda)$使用图7.14备份的混合方式。不像Q-学习，探索行为和贪婪行为没有区别。每一个子备份包含很多步的实际经验，但是只有最后一个在行为上做最大化处理。然后子备份要么是在线策略要么是离线策略。早期的过渡时在线策略的，然而最后的过渡使用贪婪策略。结果是，对于一个固定的非贪婪策略，在Peng's $Q(\lambda)$下Q_t既不收敛到Q^π，也不收敛到Q^*，而是介于二者之间。但是，如果策略渐渐的变得贪婪，那么方法也会收敛到Q^*。这些都还没有证明。尽管如此，实际上方法表现的很好。很多研究表明它的表现明显比Watkins's $Q(\lambda)$方法好，和Sarsa(λ)表现差不多。

另一方面，Peng's $Q(\lambda)$实现起来比Watkins's $Q(\lambda)$复杂。想要完整的获得实现的描述，看Peng and Willians(1994,1996)。用户可以假想第三个版本，让我们成为$naive Q(\lambda)$，它和Watkins's $Q(\lambda)$一样除了当探索行为发生时迹不重新设为0.这个方法比起Peng's $Q(\lambda)$可能只有某些好处，但是实现简单。我们对于这方法没有经验，也许他没有设想的那么朴素。

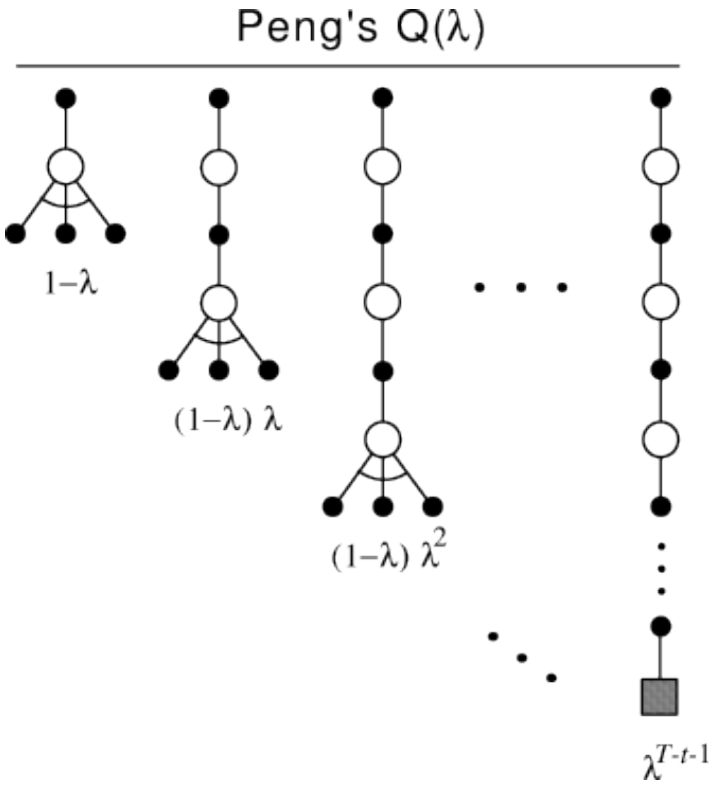


图 7.14: Peng's $Q(\lambda)$算法的备份图

7.7 Actor-Critic方法的资格迹

这一节我们介绍如何把6.6节介绍的Actor-Critic扩展到使用资格迹。这个很明确。Actor-Critic方法的critic部分只是简单的在线策略学习V^{π}。$TD(\lambda)$能够用于这点，每个状态一个资格迹。Actor的部分需要使用状态-行为对的资格迹。因此Actor-Critic方法需要两个迹的集合，一个用于状态，另一个用于每一个Actor-Critic对。

回顾一步的Actor-Critic方法通过下式更新：

$$p_{t+1}(s,a) = \begin{cases} p_t(s,a) + \alpha\delta_t & if a = a_t \; and \; s + s_t \\ p_t(s,a) & otherwise \end{cases}$$

δ_t是$TD(\lambda)$误差（7.6），$p_t(s,a)$是时间t状态s采取动作a的偏好。比如偏好通过softmax方法确定策略（2.3节）。我们把上面的方程泛化成资格迹的形式如下：

$$p_{t+1}(s,a) = p_t(s,a) + \alpha\delta_t e_t(s,a) \tag{7.14}$$

$e_t(s,a)$表示了当时状态-动作对的迹。鉴于上面提到的最简单的方式，迹可以按照$Sarsa(\lambda)$的方式更新。

在6.6节我们也讨论了一个更先进的Actor-Critic使用更新：

$$p_{t+1}(s,a) = \begin{cases} p_t(s,a) + \alpha\delta_t[1 - \pi_t(s,a)] & if a = a_t \; and \; s + s_t \\ p_t(s,a) & otherwise \end{cases}$$

为了泛化这个公式到资格迹我们可以使用和式7.14稍微不同的迹更新。当一个状态-动作对发生时不是把迹增加1，而是用$1 - \pi_t(s_t, a_t)$更新：

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 - \pi_t(s_t,a_t) & if \; s = s_t \; and \; a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & otherwise \end{cases} \tag{7.15}$$

7.8 置换迹

一些情况下可以通过使用一个稍微修正的迹来大大的改善性能，这个迹叫做替换迹。假设一个被访问的状态在迹完全衰减为0之前再次被访问。如果使用累积迹(7.5)，再次访问导致迹进一步增加，使他大于1，然后使用替换迹，迹被重新置为1。图7.15对比了这两种迹。 正式的，一个置换迹被定义为：

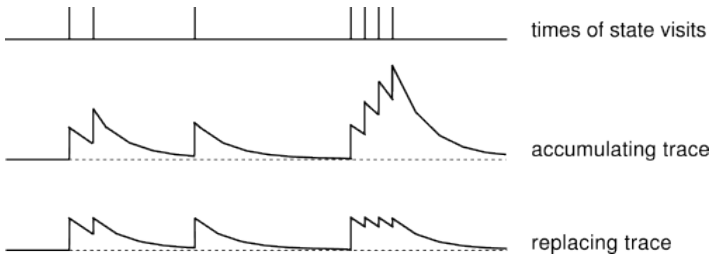


图 7.15: 累积迹和置换迹

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & if \; s \neq s_t \\ 1 & if s = s_t \end{cases} \tag{7.16}$$

预测或者控制使用置换迹的方法通常叫做置换迹方法。尽管置换迹和累积迹只是略微不同，但是在学习速率上产生很大的提高。图7.16比较了在19个状态的随机行走问题上传统的和置换迹版本的$TD(\lambda)$方法的性能。其他的更广泛的例子在下一章图8.9给出。

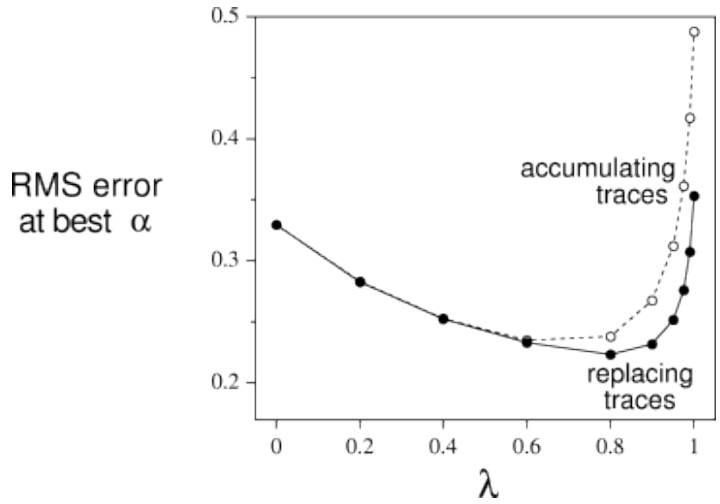


图 7.16: 两种迹的比较

7.9 实现问题

乍一看使用资格迹的方法似乎比一步方法更复杂。一个朴素的实现需要在每一个时间步长更新每一个状态的值估计和资格迹。这对于单指令，多重数据并行的电脑或者貌似有理的神经实现没有任何问题，但是对于传统的串行电脑实现是一个问题。幸运的是，对于典型的λ和γ 值所有状态的资格迹几乎总是接近0；只有那些近期被访问的状态的迹总是明显的大于0。只有这些状态实际需要被更新，因为其他的更新几乎没有效果。

实际上，传统电脑上的实现记录和更新为数不多的几个具有非0迹的状态。使用这个技巧，使用迹的计算代价基本上只是一步方法的几倍。精确的几倍依赖于λ和γ和其他的计算耗费。Cichosz (1995)展示了一个深一层的实现技术，进一步的减小了对于一个常量依赖的λ 和γ的复杂度。最后，需要注意的是对于表格式的迹的计算复杂度最糟糕在一些情况下式最糟糕的。当使用了值函数近似，不实用迹的计算优势通常减小了。比如，如果人工神经网络和后向传播被使用了，那么迹通常只会引起双倍的内存需求和计算代价。

7.10 变量λ

λ–回报可以变得和我们目前描述的方法大不相同只要我们让λ在每一步发生变化，重新定义迹更新为：

$$e_t(s) = \begin{cases} \gamma\lambda_t e_{t-1}(s) & if\ s \neq s_t \\ \gamma\lambda_t e_{t-1}(s) + 1 & if s = s_t \end{cases} \tag{7.17}$$

λ_t表示t时刻的λ值。这是一个高级的议题因为被增加的泛化能力从未被用于实际应用，但是它理论上是有趣的而且证明是有用的。比如，改变λ作为状态函数的思想：$\lambda_t = \lambda(s_t)$。如果一个状态的值估计具有很高的可能性被知道，那么充分的使用那个估计是有道理的，忽略任何在它之后的状态和回报。这意味着砍掉所有与该状态相关的迹一旦这个状态再次被访问，也就是对于某个特定状态的λ选为0或者很小的值。相似的，那些值很不确定的状态，也许因为那个状态的估计不可靠，可以让λ接近1.这导致他们的估计值在任何更新下影响很小。他们被跳过直到一个更了解的状态出现。由Sutton and Singh (1994)正式的探索了一些相关的思想。

上面的资格迹方程是变量λ一个后视的观点。相应的前向观点在$\lambda-$回报中有更一般的定义：

$$\begin{aligned} R_t^\lambda &= \sum_{n=1}^{\infty} R_t^{(n)}(1-\lambda_{t+n}) \prod_{i=t+1}^{t+n-1} \lambda_i \\ &= \sum_{k=t+1}^{T-1} R_t^{k-t}(1-\lambda_k) \prod_{i=t+1}^{k-1} \lambda_i + R_t \prod_{i=t+1}^{T-1} \lambda_i \end{aligned}$$

7.11 总结

资格迹和TD误差一起提供了一种有效的，增量的在MD方法和TD方法之间移动和选择的方式。迹可以不和TD误差一起使用来获得一种更简洁的效果，但是仅仅无技巧的。一种比如$TD(\lambda)$的方法可以通过部分经验实现预测，同时需要很少内存和很少无意义的变种。

Chapter 8

泛化和函数近似

目前为止我们假设值函数的估计可以用一个表格表示，每一个中每个状态或者状态-行为对有一个条目。这是一个特殊的清晰的有其启示性的例子，但是当然它只局限于具有状态数和行为数比较少的例子。问题不仅出现在大表格所需要的内存，当把数据精确的填充时也需要时间。换句话说，关键的问题是泛化。怎么使得一个有限的状态子空间的子集的经验很好的泛化到一个大很多的子集，从而产生一个好的近似。

这是一个严重的问题。在很多想要应用增强学习的任务中，大多数遇到的状态绝不会提前就经历过。在包含连续变量或者复杂意义的场景中经常遇到这种情况，比如一张图像。在这些任务上学习所有东西的唯一方式就是把先前的经验状态泛化到从没经历过的状态。

所幸的是，从抽样的泛化已经有了广泛深入的研究，我们没必要完全探索新的方法用在增强学习中。很大程度上，我们只需要结合增强学习方法和当前存在的泛化方法。我们需要的这种泛化叫做函数近似，因为它从一个想得到的函数（值函数）抽样，试图泛化他们来构建一个完整的函数的近似。函数近似是监督学习的一个例子，也是在机器学习，人工神经网络，模式识别和统计曲线拟合中首要研究的主题。原则上，所有在这些领域研究的方法都可以被用到增强学习。

8.1 值估计和函数近似

像往常一样，我们以从策略π产生经验来学习状态值函数V^{π}的估计问题开始。本章的新颖性在于近似值函数V_t代表的不是一个表而是用参数向量$\vec{\theta}_t$参数化的一个函数。这意味着值函数V_t完全依赖于$\vec{\theta}_t$，随着每个时间步长$\vec{\theta}_t$的变化而变化。比如值函数可以通过神经网络计算，参数是神经元的链接权重。或者可以通过决策树计算，参数定义为分支点或者叶子的值。一般情况下，参数的数量远远小于状态的数量，改变

一个参数会改变很多状态的值估计。结果是当一个状态被备份时，从那个状态产生的变化会影响很多其他的状态的值。

所有本书中涉及的预测方法都被描述为备份，也就是当更新到一个估计值函数等价于改变特殊状态的值到一个“备份值”。让我们用符号$s \mapsto v$，s是备份的状态，v是备份值或者目标，s的估计向前移动。比如，DP备份的值函数预测是：$s \mapsto E_\pi\{r_{t+1} + \gamma V_t(s_{t+1}) \mid s_t = s\}$，MC方法的备份是$s_t \mapsto R_t$，TD（0）的备份是：$s_t \mapsto r_{t+1} + \gamma V_t(s_{t+1})$，通用的$TD(\lambda)$备份是：$s_t \mapsto R_t^\lambda$。在DP情况下，任意的状态$s$ 都备份，在其他的情况在经验中遇到的状态s_t被备份。

把每个备份具体化为估计值函数输入输出行为的一个例子是很自然的理解。在某种意义上，备份$s \mapsto v$意味着状态s的估计值应该更像v。

以这种方式把每一个备份看做一个传统意义上的训练样本使我们能够使用现有的广泛存在的函数近似方法。原则上所有的监督学习相关方法都可以用。但是不是所有的函数近似方法都同样的适合在增强学习中使用。最先进的神经网络和统计方法都假设一个静态的训练集，基于它做了多次处理。然而在增强学习中，在与环境的交互中或者环境模型中，能够在线学习是很重要的，为了实现这个目标需要从增量式获得的数据中高效的学习。除此之外，增强学习通常需要函数近似方法能够处理非静态的目标函数（目标函数随时间变化）。比如，在GPI控制方法中我们经常试求学习Q^π当π是变化的。即使策略保持不变，如果目标值是有自举方法产生的，值函数依然是非静态的。不能很好处理这种非静态性的方法都不适合于增强学习。

什么性能参数适合评估值函数近似方法呢？大多的监督学习方法试图最小化输入P分布上的均方根误差（MSE）。在我们的值预测问题中，输入是状态，目标函数是值函数V^π的真实值，因此对于近似V_t的MSE误差用参数表示为：

$$MSE(\vec{\theta_t}) = \sum_{s \in \mathcal{S}} P(s)\big[V^\pi(s) - V_t(s)\big]^2 \tag{8.1}$$

P是不同状态的误差分布权重。这个分布很重要因为通常所有误差减为0是不可能的。毕竟，比起分量$\vec{\theta_t}$通常有更多的状态。值函数近似器的灵活性是有必要的。一些状态更好的近似只能以其他状态更糟糕的近似为代价。所以分布P指定如何权衡各个状态。

分布P同时也是训练样本中状态选择的依据，也是对这些状态做备份。如果我们想最小化某一状态分布的误差，那么把函数近似器看做来自同一分布的例子是有道理的。现在我们假设做了备份的状态的分布和误差权重分布P是一样的。

一个特别关注的分布是描述智能体在与环境交互过程中根据策略π选择行为后经历状态的频率，该策略的值函数也是我们要近似的。我们称之为在线策略分布，部分是因为他是在线策略控制方法备份的分布。最小化在线策略的误差专注于那

些实际发生的状态的函数近似，忽略那些从未发生的。在线策略分布也是用MC或者TD方法最容易获得训练样本。这些方法用来自策略π的抽样经验产生备份。因为通常在每一个经历中出现的状态都会产生备份，可用的训练样本的分布自然是根据在线策略分布。我们稍后会讨论到，在线策略分别比起其他分布能保证更好的收敛性。

我们需要关注最小化MSE还不完全清楚。值函数预测的潜在目标不同，因为我们最终的目标是用预测来辅助找到更好的策略。那个目的下的最好预测不见得就是最小的MSE。但是，还不清楚一个更有用的替代目标是什么。现在我们继续专注于MSE。

MSE的理想目标是找到一个全局最优解，在所有可能的$\vec{\theta}$,一个向量$\vec{\theta}^*$满足$MSE(\vec{\theta}^*) \leq MSE(\vec{\theta})$。对于线性的近似求这样的解是可能的，但是对于复杂的近似函数求解困难，所以有时候寻求一个局部最优的解。尽管这个保证让人不放心，但是对于非线性函数近似来说这可能是最好的结果了。

8.2 梯度下降方法

现在我们详细的探讨一类值函数预测中的函数近似学习方法，这些方法基于梯度下降。

在梯度下降法中，参数向量是一个列向量具有固定数目的实值分量，$\vec{\theta_t} = (\theta_t(1), \theta_t(2), \ldots, \theta_t(n))^T$，$V_t(s)$是对于所有状态的$\vec{\theta_t}$的一个平滑的可微的函数。现在我们假设每一步$t$，获得一个新的的样本$s_t \mapsto V^\pi(s_t)$。

假设样本出现的状态有相同的分布P，基于它我们通过式8.1试图最小化MSE。一个好的策略是基于观测到的样本最小化。梯度下降法通过每个样本后沿着误差下降最快的方向调整参数向量：

$$\begin{aligned}\vec{\theta}_{t+1} =& \vec{\theta_t} - \frac{1}{2}\alpha\nabla_{\vec{\theta_t}}\left[V^\pi(s_t) - V_t(s_t)\right]^2 \\ =& \vec{\theta_t} + \alpha\big[V^\pi(s_t) - V_t(s_t)\big]\nabla_{\vec{\theta_t}}V_t(s_t)\end{aligned} \tag{8.2}$$

α是正的步长参数，$\nabla_{\vec{\theta_t}} f(\vec{\theta_t})$表示函数$f$的偏微分。

现在可能还不明显为什么要选择一个很小的步长。为什么不直接沿着梯度方向移动直到消除所有的误差呢？实际上梯度下降法的收敛性通过步长参数的减小保证。如果减小满足标准的随机近似条件（2.7），梯度下降方法保证会收敛到一个局部最优值。

现在我们讨论第t个训练样本的目标输出v_t，$s_t \mapsto v_t$，不是$V^\pi(s_t)$的真实值，而

是它的近似。比如，v_t可能是$V^\pi(s_t)$ 的噪声版本，或者他是前面部分提到过的备份值。在这种情况下我们不能准确的执行8.2，因为$V^\pi(s_t)$是未知的，但是我们可以通过v_t替代$V^\pi(s_t)$来近似它。这产生了通用的用于状态值预测的梯度下降法：

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\big[v_t - V_t(s_t)\big]\nabla_{\vec{\theta}_t} V_t(s_t) \tag{8.3}$$

如果v_t是一个无偏估计，也就是$E\{v_t\} = V^\pi(s_t)$，对于每个t，那么$\vec{\theta}_t$在随机近似条件下回收敛到局部最优。比如，样本中的状态是通过策略π和环境交互产生的状态。用R_t表示每个状态S_t跟随的回报。因为一个状态真实的状态值是跟随他的期望回报值，根据定义MC方法目标$v_t = R_t$是$V^\pi(s_t)$的无偏估计。利用这个选择，通用的梯度下降法收敛到局部最优值。

相似的，我们可以用n步TD回报的平均作为v_t。$TD(\lambda)$的梯度下降用$\lambda-$回报$v_t = R_t^\lambda$，把它作为$V^\pi(s_t)$近似产生前向观点的更新：

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\big[R_t^\lambda - V_t(s_t)\big]\nabla_{\vec{\theta}_t} V_t(s_t) \tag{8.4}$$

不幸的是，对于$\lambda < 1, R_t^\lambda$不是$V^\pi(s_t)$的无偏估计，因此这个方法不收敛到局部最优。当DP目标是$v_t = E_\pi\{r_{t+1} + \gamma V_t(s_{t+1} \mid s_t)$情况也是一样的。尽管如此，这些自举方法也是相当有效的，对于重要的特别的例子可以提供性能保证，本章后面我们会探讨。现在我们强调这些方法和通用梯度下降的关系（式8.3）。尽管式8.4的增量本身不是梯度，把它看成是梯度下降方法是有益的。

梯度下降$TD(\lambda)$的的后向观点是：

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\delta_t\vec{e}_t \tag{8.5}$$

δ_t是TD误差，

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{8.6}$$

$\vec{e}_t$是资格迹的列向量，每个对应$\vec{\theta}_t$的分量，更新为：

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t) \tag{8.7}$$

当$\vec{e}_0 = \vec{0}$，一个完整的在线梯度下降$TD(\lambda)$由图8.1给出：

两种基于梯度的函数近似方法在增强学习中有广泛的应用。一个是多层人工神经网络用误差反向传播算法。第二个流行的方法是线性形式。

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
  $\vec{e} = 0$
  $s \leftarrow$ initial state of episode
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $\vec{e} \leftarrow \gamma\lambda\vec{e} + \nabla_{\vec{\theta}} V(s)$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    $s \leftarrow s'$
  until s is terminal

图 8.1: 在线梯度下降$TD(\lambda)$用于V^π估计

8.3 线性模型

梯度下降函数近似一个最重要特殊的例子是近似的函数V_t是参数向量的$\vec{\theta}_t$的线性的函数。对于每个状态s，相应的有一个列特征向量$\vec{\phi}_s = (\phi_s(1), \phi_s(2), \phi_s(2), \ldots, \phi_s(n))^T$，和$\vec{\theta}_t$具有同样的分量个数。特征可以从状态中用很多种不同的方式构造，我们罗列一些可能的方法。一旦特征建好了，近似状态值函数表示为：

$$V_t(s) = \vec{\theta}_t^T \vec{\phi}_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i) \tag{8.8}$$

在这种情况下近似值函数被称为是参数线性的，或者简单的线性的。

对于线性的近似值函数用梯度下降是自然的。梯度计算公式为：

$$\nabla_{\vec{\theta}_t} V_t(s) = \vec{\phi}_s$$

使用线性近似后式8.3简化了很多，并且对于线性的方程，只有一个全局最优，所以任何收敛到局部最优的方法都会收敛到近似的全局最优点。对于前面讨论的$TD(\lambda)$算法，满足随进近似条件和步长减小条件时会收敛，但是这个收敛不是最小误差向量$\vec{\theta}^*$，而是一个附近的参数向量$\vec{\theta}_\infty$，它的误差限制在：

$$MSE(\vec{\theta}_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} MSE(\vec{\theta}^*) \tag{8.9}$$

也就是说渐进的误差不超过最小误差的$\frac{1-\gamma\lambda}{1-\gamma}$倍。当$\lambda$趋向于1，界限趋近最小误差。

除了理论的结果，线性模型方法引起大家的兴趣是因为实际中就数据和计算而

言它都是非常高效的。是否是这样关键的地方取决于就特征而言如何表示这些状态。选择合适的特征的过程其实是一个加入先验知识的过程。直觉上，特征应该和任务的自然特征相关，比如移动机器人的位置，剩余电量以及近期的声呐读数等等。

通常，特征是自然特性的结合。这是因为线性的形式没有表达特征之间关联的能力，比如特征i只有在特征j不出现的情况下才是好的。比如倒立摆问题中一个大的角速度是好是坏依赖于角位置。对于这些有交互的情况我们需要引入一些耦合特征。

8.3.1 粗糙编码

假设一个任务的状态集是连续的二维的。这种情况下的状态是一个点，有2个分量的向量。这种情况下的一种特征是关于状态空间里的一个圆，如图8.2所示。如果一个状态在圆内，那么相应的特征值为1并且称之为出现；否则为0表示没出现。这种0-1值的特征叫做二值特征。给定一个状态，哪一个二进制特征出现指示了状态在哪一个圆内，因此可以粗糙的编码它的位置。用这种重叠的特征表示状态的方式叫做粗糙编码。

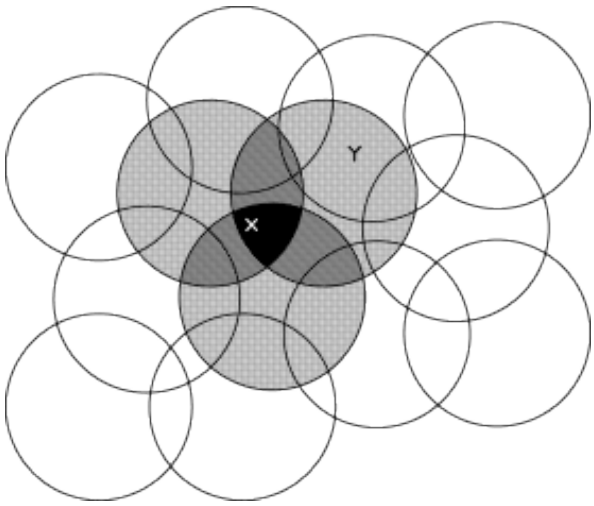


图 8.2: 粗糙编码

假设线性的梯度下降函数近似，考虑大小和密度的影响。与每一个圆关联的是单个参数（$\vec{\theta}_t$的一个分量），通过学习更新。如果我们训练了一个点X，然后所有和X有交集的参数都会受到影响。因此通过式8.8值函数在所有圆并集的点会受到影响，和X有更多相同圆的点受到更大的影响，如图8.2。如果圆很小，那么泛化会在小的距离上进行，如图8.3a所示，如果很大，将会影响到一个大的距离，如图8.3b所

示。更多的特征的形状决定了泛化的性质。比如如果那些圆不是严格的圆周，而是在一个方向有拉伸，泛化也会相似的被影响，如图8.3c所示。

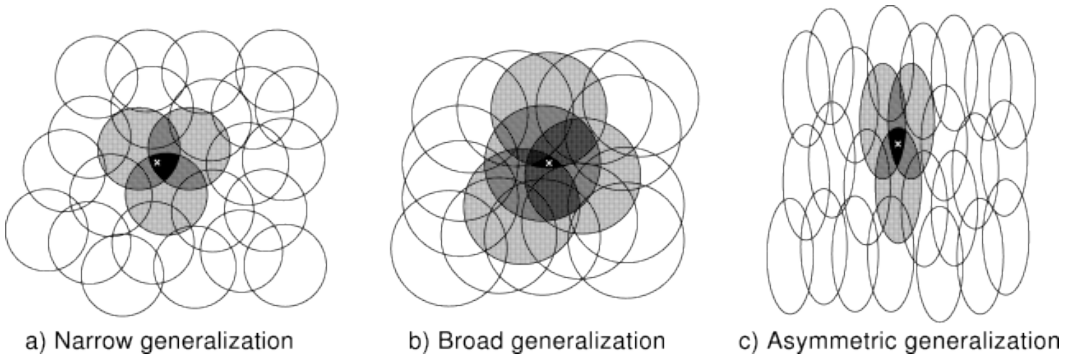


图 8.3: 大小和形状对泛化性能的影响

8.3.2 Tile Coding

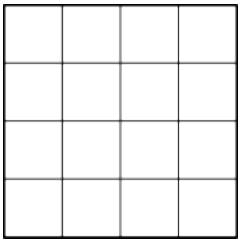
Tile Coding是粗糙编码的一种，尤其适合连续的数字电脑用于高效的在线学习。在Tile Coding中接纳的特征区域分组详细的覆盖了输入空间。每一个字划分叫做一个瓷片，每个划分的元素叫做一个tile。每一个tile 是一个二值特征的接纳区域。

Tile coding的一个直接的好处是在一个时间出现的所有的特征的数目被严格的控制并且输入状态之间相互独立。一个特征出现在一个瓷片中，因此出现的特征的总数总是和瓷片的数目相同。这允许我们以一种简单的直觉的思维设置步长参数α。比如选择$\alpha = \frac{1}{m}$，m表示瓷片的数目，这导致了精确的一次尝试学习。如果样本$s_t \mapsto v_t$被接收，然后无论先前的值是什么，新的值$V_{t+1} = v_t$。通常情况下用户希望变化的更慢一些，允许在目标输出中有泛化和随机变化。例如可以选择$\alpha = \frac{1}{10m}$，这种情况下一次更新会向目标前进十分之一。

因为tile编码值唯一的使用二值特征，权重和得到的近似值函数的计算是琐碎的。不是执行n次乘法和加法，只需要简单的计算$m \ll n$出现的特征的索引然后把m个相关的参数向量的分量加起来。资格迹的计算（8.7）也被简化，因为梯度（$\nabla_{\vec{\theta}} V_t(s_t)$）的分量通常是0，除了1。

如果使用了类似网格的瓷片，出现特征的索引值的计算尤其容易。假设我们解决一个具有两个连续状态变量的问题。那么最简单的空间瓷片化的方法是用均匀的二维网格：

给定空间中一个点的坐标x, y，计算上容易决定tile所在位置的索引。当用了多重瓷片，每一个都偏移不同的数，以便每一个能以不同的方式切分空间。在图8.4展示的例子，额外的增加了一行和一列，以便覆盖所有的点。两个突出显示的tile用×表示，代表了那些出现的状态。不同的瓷片（tilings）随机的偏移一个量，或者通过



精心设计的确定性策略。图8.3和图??表示对于泛化和渐进精确度的影响也适用于这里。tile的宽度和形状应该选择能够匹配用户期望的合适的泛化的广度。瓷片数的选择影响tile的密度。瓷片越稠密，近似函数的精度越高，但是计算代价也更大。

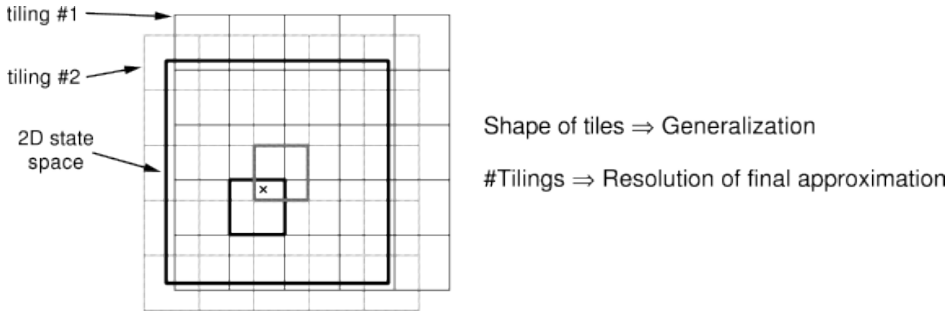


图 8.4: 多级的，重叠的gridtilings

注意到瓷片可以是任意的，没有必要是均匀的网格。tile不仅可以是奇怪的形状，如图8.5a，而且他们可以成为任意的形状和分布来实现特殊的泛化。比如8.5b的条形瓷片能够促进垂直维度的泛化，增强水平维度的辨别力，尤其是左边。三角形条纹的瓷片可以促进对角方向的泛化，如图8.5c。高维的情况，坐标轴对齐的条纹相应的忽视瓷片中的一些维度，也就是超平面切片。

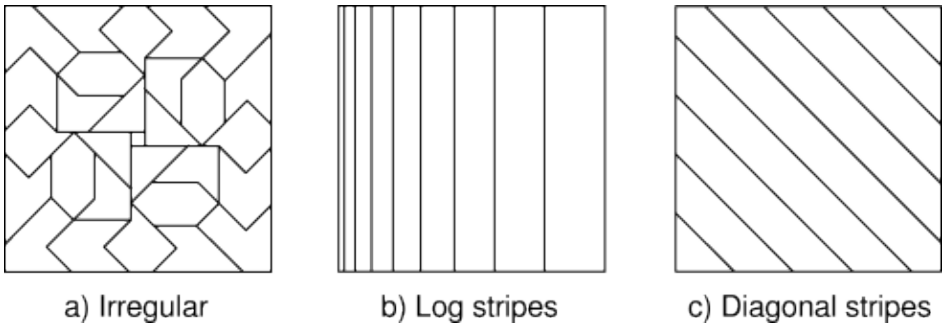
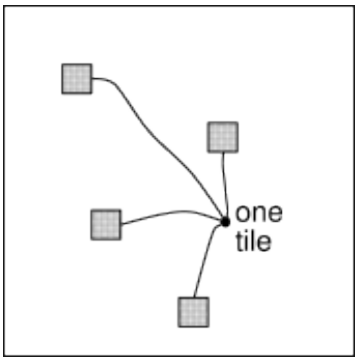


图 8.5: Tilings

另一个减小内存需求的技巧是哈希——a consistent pseudo-random collapsing of a large tiling into a much smaller set of tiles. Hashing produces tiles consisting of noncontiguous, disjoint regions randomly spread throughout the state space, but that

still form an exhaustive tiling. For example, one tile might consist of the four subtiles shown below: 通过哈希，内存需求按照大的因子减小而性能基本不丢失。这是因



为在状态空间只有一小部分需要高的分辨率。在内存需求不是维度数指数级的意义上，哈希解决了维度灾难，只是需要匹配实际任务的需求。好的公共领域的tile coding，包括哈希的实现都可以获得。

8.3.3 径向基函数

径向基函数是粗糙集编码到连续值特征的自然的泛化。不是每个特征必须是0或1，它可以是[0,1]区间的任何数，特征可以表现不同的度。一个典型的RBF特征i具有一个高斯响应$\phi_s(i)$，仅仅依赖于状态s和特征原型或者状态中心c_i的距离，和特征宽度σ_i的相对值：

$$\phi_s(i) = exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right)$$

过程的基准或者距离测度可以根据手头的任务以任何方式选择，只要是合适的。图8.6展示了一维的欧几里得测度的例子：

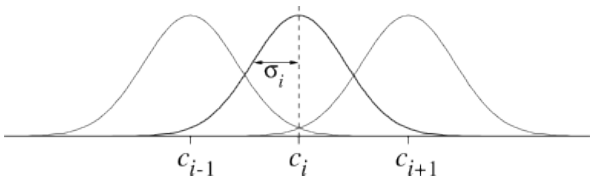


图 8.6: 一维的径向基函数

RBF网络是使用RBF作为他的特征的线性函数逼近器。学习有方程8.3和8.8定义，和其他的函数逼近器一样。RBF相对于二值特征的首要优点是它产生的函数逼近器是光滑的并且可微。除此之外，以写针对RBF网络的学习方法也改变中心和特征的宽度。这些非线性的方法可能会更加精确的拟合目标函数。RBF网络的缺点，

尤其是非线性的RBF网络，是巨大的计算复杂度，而且一般情况下需要人为的调整才能保证效率和鲁棒性。

8.3.4 Kanerva Coding

8.4 函数近似的控制

我们现在扩展使用函数近似预测值函数方法到控制方法，遵循GPI的模式。首先我们扩展状态值函数预测问题到行为值函数预测方法，然后结合策略改进和行为选择技术。像以前一样，保证探索的问题由在线策略或者离线策略方法解决。

行为值函数的扩展是直接的。在这里$Q_t \approx Q^\pi$用一个参数化的向量$\vec{\theta}$表示。不同于之前考虑的训练样本的形式$s_t \mapsto v_t$，现在我们考虑样本$s_t, a_t \mapsto v_t$。目标输出可以是任何$Q^\pi(s_t, a_t)$的近似，包括通常的备份值比如完整的蒙特卡罗回报R_t，或者一步Sarsa形式的回报$r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})$。通用的用于行为值预测梯度下降更新如下：

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\big[v_t - Q_t(s_t, a_t)\big]\nabla_{\vec{\theta}_t} Q_t(s_t, a_t)$$

比如，行为值方法的后向观点类比于$TD(\lambda)$是：

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\delta_t\vec{e}_t$$

这里的δ代表：

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \qquad \vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} Q_t(s_t, a_t)$$

$\vec{e}_t = \vec{0}$。我们称这种方法为梯度下降$Sarsa(\lambda)$。当策略固定时，这种方法以和$TD(\lambda)$同样的方式收敛，同样的误差限由8.9给出。

为了形成控制方法，我们需要把行为值预测方法和策略改进和行为选择技术耦合起来。适用于连续的行为，或者来自大规模的离散集的行为的应用技术是当前研究的一个主题并且还没有清晰的解决方案。另一方面，如果行为集合是离散的又不太大，那么我们可以使用前面章节已经开发的技术。也就是对于每一个可能的行为a，当前状态s，我们可以计算$Q_t(s, a)$，然后找到贪婪策略$a_t^* = \arg\max_a Q_t(s_t, a)$。策略改进通过改变策略估计成为贪婪策略（离线策略方法）或者是一个软的近似策略例如$\varepsilon - greedy$策略（在线策略方法）。在在线策略的

情况下，行为选择根据同一个策略，如果是离线策略方法，行为选择可是任意的策略。

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
    $\vec{e} = \vec{0}$
    $s, a \leftarrow$ initial state and action of episode
    $\mathcal{F}_a \leftarrow$ set of features present in s, a
    Repeat (for each step of episode):
        For all $i \in \mathcal{F}_a$:
            $e(i) \leftarrow e(i) + 1$ (accumulating traces)
            or $e(i) \leftarrow 1$ (replacing traces)
        Take action a, observe reward, r, and next state, s
        $\delta \leftarrow r - \sum_{i \in \mathcal{F}_a} \theta(i)$
        With probability $1 - \varepsilon$:
            For all $a \in \mathcal{A}(s)$:
                $\mathcal{F}_a \leftarrow$ set of features present in s, a
                $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
            $a \leftarrow \arg\max_a Q_a$
        else
            $a \leftarrow$ a random action $\in \mathcal{A}(s)$
            $\mathcal{F}_a \leftarrow$ set of features present in s, a
            $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
        $\delta \leftarrow \delta + \gamma Q_a$
        $\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$
        $\vec{e} \leftarrow \gamma \lambda \vec{e}$
    until s is terminal

图 8.7: 线性的，使用二值特征和$\varepsilon - greedy$策略的$Sarsa(\lambda)$梯度下降。具体阐述了累积迹和置换迹两种更新方式，包括清除未选择行为的迹

图8.7和图8.8展示了使用函数近似的在线策略($Sarsa(\lambda)$)和离线策略(Watkins's $Q(\lambda)$)控制方法。两种方法都使用了二值特征的梯度下降近似，比如tile coding和Kanerva coding。两种方法都使用$\varepsilon - greedy$做行为选择。都计算了出现特征及$\mathcal{F}_a$，对应于当前状态和所有可能的行为s。如果每一个行为的值函数都是相同特征的单独的线性函数，那么每一个行为的$\mathcal{F}_a$索引都是相同的，大大的简化了计算。

所有上面讨论的算法都使用了累积资格迹。尽管置换迹相对于表格式方法具有一些优势，置换迹没有直接扩展到值函数近似。回想置换迹是每次重新访问一个状态时就把他的迹重新设为1，而不是用1累加。但是对于这里的函数近似一个状态没有对应的一个迹，一个迹对应$\vec{\theta}_t$的一个分量，一个分量对应很多状态。一种对于线

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
    $\vec{e} = \vec{0}$
    $s, a \leftarrow$ initial state and action of episode
    $\mathcal{F}_a \leftarrow$ set of features present in s, a
    Repeat (for each step of episode):
        For all $i \in \mathcal{F}_a$: $e(i) \leftarrow e(i) + 1$
        Take action a, observe reward, r, and next state, s
        $\delta \leftarrow r - \sum_{i \in \mathcal{F}_a} \theta(i)$
        For all $a \in \mathcal{A}(s)$:
            $\mathcal{F}_a \leftarrow$ set of features present in s, a
            $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
        $\delta \leftarrow \delta + \gamma \max_a Q_a$
        $\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$
        With probability $1 - \varepsilon$:
            For all $a \in \mathcal{A}(s)$:
                $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
            $a \leftarrow \arg\max_a Q_a$
            $\vec{e} \leftarrow \gamma \lambda \vec{e}$
        else
            $a \leftarrow$ a random action $\in \mathcal{A}(s)$
            $\vec{e} \leftarrow 0$
    until s is terminal

图 8.8: 线性的，使用二值特征和$\varepsilon - greedy$策略的$Watkin's Q(\lambda)$梯度下降，使用累积迹

性的，使用二值特征的梯度下降函数近似方法有效的方法是把特征看待成状态，为了达成置换迹的目的。也就是每一次访问到一个有特征 i的状态时，对于特征 i的迹被设为1而不是增加1.

当处理状态-动作迹是，清楚哪些没有被选择的行为也是有益的。这个思想可以扩展到二值特征线性函数近似的例子。对于每一个访问的状态，我们首先清除那些没有选择的状态和行为的特征的迹，然后把选择的状态和行为的特征的迹设为1.正如在表格方法中说到的，这个可能不适合用置换迹的方式。两种迹，包括为选择动作的选择性清除的具体方法步骤见图8.7。

例8.2：过山车考虑任务一个动力不足的汽车开过一个陡峭的山路，如图8.9左上所示。困难之处在于重力大于小车的动力，即使开到最大马力小车也不能加速开过斜坡。唯一的解决办法是背离目标开到目标另一边的斜坡。然后，然后借助惯性越过山坡。这是一个简单的连续的控制任务，事情必须先变得更坏然后才能变好。很多控制方法都难以解决此类问题，除非人为的辅助设计。

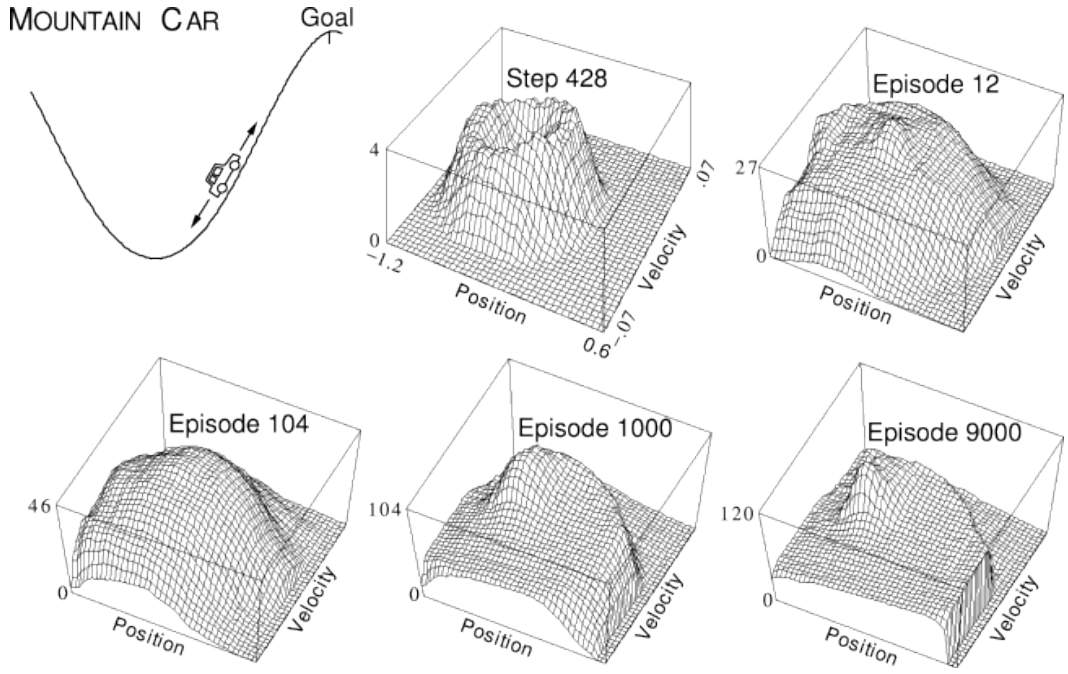


图 8.9: 过山车问题

这个问题中所有时间步长的回报都是-1，直到小车越过山峰顶部的目标位置，此时片段结束。有三个可能的行为：全速正向前进（+1），全速反向（-1）和0。小车移动根据简单的物理知识。它的位置x_t和速度$\dot{x}_t$通过下面规律更新：

$$x_{t+1} = bound[x_t + \dot{x}_{t+1}]$$

$$\dot{x}_{t+1} = bound[\dot{x}_t + 0.001a_t - 0.0025\cos(3x_t)]$$

*bound*函数表示强制限制范围，其中$-1.2 \leq x_{t+1} \leq 0.5, -0.07 \leq \dot{x}_{t+1} \leq 0.07$。当$x_{t+1}$到达了左边界，$\dot{x}_{t+1}$清0。当它到达了右边边界，目标达到，片段结束。每个片段从一个均匀的从这些范围内选择一个随机位置和随机速度。为了把这两个连续的状态变量变成二值特征，我们使用图8.4中的gridtilings。我们使用一个9×9的瓷片，每一个通过tile宽度的部分偏移。

图8.7中的Sarsa算法很容易的解决了这个问题，在100个片段内学习到了一个接近最优的策略。图8.9展示了一轮学习中值函数的相反数，使用参数$\lambda = 0.9, \varepsilon = 0, \alpha = 0.05(0.1/m)$。初始行为值为0，这是比较乐观的（该任务中真实值函数都是负的），因为做了大量的探索，即使探索参数$\varepsilon = 0$。这个可以通过“step 428”看出。此时一个片段还没有完成，但是小车开始在山谷中来回震荡，在状态空间表现出原型的轨迹。所有频繁探索的状态的值比未探索的更糟糕，这是因为实际的回报比期望的要差。这趋势小车离开他在的地方，去探索新的状态，直到找到一个解。图8.10详细的研究了参数α, λ的选择、迹种类的选择对学习速率的影响。

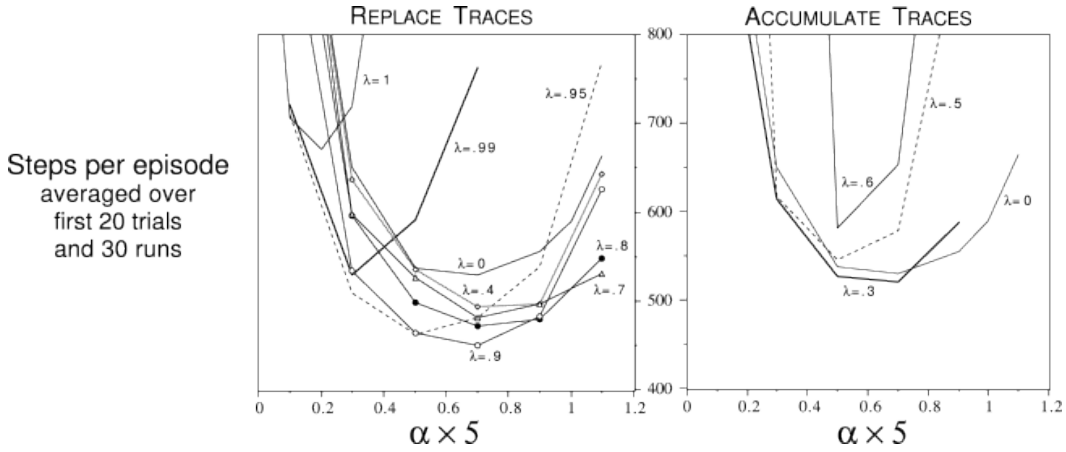


图 8.10: 不同种类迹以及参数α, λ对学习速率的影响

8.5 离线策略自举

现在我们返回预测问题仔细看看自举、函数近似、和在线策略分布的联系。自举的意思是一个状态值的更新借助于另一个估计值。TD方法涉及自举，像DP方法一样，然而MC方法不自举。$TD(\lambda)$是一个自举方法仅限于$\lambda < 1$，当$\lambda = 1$时我们不认为他是一个自举方法。尽管TD（1）在一个片段之内有自举，但是在一个完整的片段上它的效果和无自举的蒙特卡罗更新是一样的。

比起非自举的方法，自举的方法更难和函数近似结合。比如，考虑线性的，梯度下降值函数近似的例子。在这个例子中，对于任何的训练样本的分布P非自举的方法都能找到最小均方误差解（8.1）。然而自举的方法仅仅找到近似的最小误差解（8.9），而且仅仅是在线策略分布。更糟糕的是，MSE界限的性质会变得更糟当λ远离1，也就是说当方法偏离非自举的形式。

对于在线策略分布的自举方法收敛结果的限制得到了充分的考虑。这对于诸如Sarsa和actor-critic的在线策略方法来说不是问题，但是对于离线策略的Q-学习和DP方法是一个问题。离线策略控制中状态备份的分布和依据估计策略产生状态的分布完全一样。很多的DP方法，比如统一的备份所有状态。Q学习可以根据任意的分布备份状态，但是一般的还是根据与环境交互产生的分布遵循一个接近贪婪估计策略的软策略。我们使用离线策略自举这个词表示任何备份的分布和在线策略分布不同的自举方法。令人惊讶的是，离线策略自举结合函数近似会导致发散和无穷大的均值平方误差。

8.6 我们应该Bootstrap吗？

在这一点上你可能会疑惑我们为什么因为自举方法而烦恼。比起自举方法来，非自举的方法能够更可信的在一个更宽松的条件下应用于函数近似。非自举的方法获得一个更低的渐进误差，即使备份是根据在线策略分布的。通过让资格迹$\lambda = 1$，非自举方法的在线的、一步一步增量式的实施甚至是可能的。尽管有各种这样的原因，实际上通常我们还是会选择自举的方法。

在经验比较中，自举方法通常表现比非自举方法好。一个简单的比较这二者的方法是使用TD方法，然后从0（纯粹的自举）到1（纯粹的非自举）改变资格迹λ。图8.11总结了这些结果。在所有的情况下，当λ趋向于1（非自举）时，表现变得更糟糕。右上方的图表现的尤其明显。这是一个策略改善的任务，使用根号MSE作为性能测量参数。渐进的情况下，$\lambda = 1$是最好的，但是在非渐进的情况下，我们可以看到表现糟糕。

目前为止还不是十分清楚为什么涉及到一些自举的方法表现要比纯粹的非自举的方法好很多。可能是自举的方法学习的更快，或者可能是它们实际上学习的就比非自举的方法好。目前的存在的结果表明在减小与真实值函数误差的MSE指标上，非自举的方法表现的比自举方法好，但是减小MSE不是非得必要的最重要的目标。比如，如果你在任何状态的真实行为值函数上统一加1000，此时的MSE会很大，也就是很糟糕，但是你仍然会得到最优的策略。自举方法究竟发生了什么不是那么简单，但是他们似乎确实做了一些正确的事情。我们希望随着研究的继续会改善对这个问题的理解。

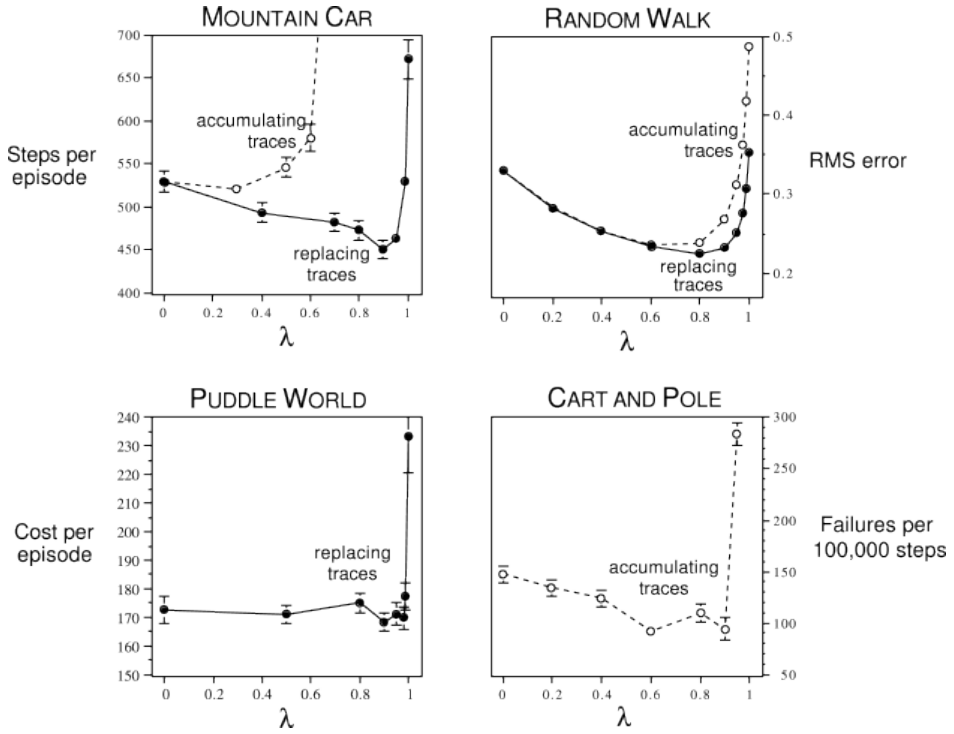


图 8.11: 增强学习中λ的作用，在所有情况下，性能越好，曲线越低。左边的两幅图是简单的连续状态控制任务，使用的是Sarsa(λ) 算法和tile coding，迹的选择可以是置换迹或者累积迹。上边右图是使用TD(λ)方法进行策略估计的随机行走问题。右边下图示倒立摆问题的无惩罚数据（例3.4）

8.7 总结

如果增强学习问题要用于人工智能或者大规模的工程应用，那么它必须具备泛化的能力。为了实现这个目标，任何当前广泛存在的用于监督学习值函数近似的方法都能被用于增强学习，你只需要简单把每一个备份看成是训练样本。特别的梯度下降方法可以很自然的扩展到前面章节阐述的函数近似的技术，包括资格迹。线性梯度下降法理论上尤其吸引人，并且当提供合适特征的情况下，实际中表现的也很好。选择特征是给增强学习系统施加先验知识的一个重要的手段。线性的方法包括径向基函数，tile coding和Kanerva coding。多层神经网络后向传播方法是用于非线性梯度下降函数近似的方法。

在大多情况下，增强学习预测和控制方法到梯度下降形式的扩展是很直接的。但是在函数近似、自举和在线离线策略特性上具有有趣的相互作用。自举方法，比如DP和TD(λ)($\lambda < 1$)方法，在于函数近似结合时尽在一定的条件下会比非自举方法表现的更可靠。因为控制的情况还没有经过理论上的分析，研究都集中于值预测问题。在这种情况下，使用线性梯度下降函数近似的在线策略自举方法会收敛到一个可靠的解，解的MSE误差限定在最小可能误差的$\frac{1-\gamma\lambda}{1-\gamma}$倍。另一方面，离线策略自举方法可能会偏离产生无穷大误差。已经探索了一些方法试图解决离线策略自举和函数近似结合的问题，但是这仍旧是一个开放的研究问题。尽管自举方法存在理论保证的局限，但是由于实际中他表现的比非自举方法好很多，所有它仍然是增强学习研究中持续的一个兴趣点。

Chapter 9

规划和学习

这一章中我们用一个统一的视角来阐述基于环境模型的方法（比如动态规划和启发式搜索）和不需要模型的方法（比如MC方法和TD方法）。我们把前者看待为规划方法，后者看待为学习方法。尽管这两类方法有着实际的区别，但是他们也有很多的相似性。尤其是两类方法的核心都是值函数的计算。此外，所有方法都是基于对未来时间的洞察，备份值的计算，然后用它更新近似值函数。在这本书的前一部分我们介绍了MC方法和TD方法作为不同不同的方法，然后我们有展示了如何使用资格迹使他们无缝的集成，比如TD（λ）方法。本章的目标也是一个相似的关于规划方法和学习方法的集成。前面章节中把他们作为不同的方法做了介绍，现在我们探索某种程度上这些方法的混合。

9.1 模型和规划

借助环境模型的意思是指一个智能体可以利用的用来预测环境如何对它的行为进行反应的任何事。给定一个状态和一个行为，一个模型产生下一个状态和回报的结果预测。如果模型时随机的，那么将会有几个可能的下一个状态和回报，每一个都以一定的概率发生。一些模型产生所有的可能性以及其概率的描述；这些我们称为分布模型。其他的模型只产生一个可能性，是根据概率抽样的；这些我们称为抽样模型。比如考虑一打骰子的和的模型。一个分布模型会产生所有可能的和以及他们发生的概率，然而一个抽样模型只是根据概率分布产生一个单独的和。在动态规划中的模型——估计所有的状态转移概率和期望回报$P^a_{ss'}$和$R^a_{ss'}$——是一个分布模型。在第5章我们使用的二十点的例子是一个抽样模型。分布模型强于抽样模型在于它能够用于产生抽样模型。但是令人惊讶的是在很多应用中获得抽样模型要比分布模型容易的多。

模型可以用来模仿或者仿真经验。给定一个起始状态和行为，一个抽样模型会

产生一个可能的转移，一个分布模型产生所有可能的转移。给定一个起始状态和一个策略，一个抽样模型能够产生整个片段，分布模型能够产生所有可能的片段和他们的概率。无论哪一种情况，我们都说模型被用于模拟环境并产生一个模拟的经验。

规划这个词在不同的领域以一些不同的方式使用。我们使用这个词表示任何计算过程，该过程以模型作为输入，产生一个与建模的环境进行交互的改善的策略：

$$model \xrightarrow{planning} policy$$

在人工智能领域，根据我们的定义有2种截然不同的规划方式。状态空间规划，包括我们这本书里面的方法，规划主要被看成是基于状态空间的最优策略或者目标路径的搜索。行为引起状态之间的转移，值函数基于状态被计算。在规划空间的规划，规划被看成是在规划空间的搜索。操作使得计划之间相互过渡，如果有值函数的话，值函数被定义在计划空间上。规划空间规划包括进化算法和偏序规划——在人工智能领域一个很流行的规划方法，在该方法中所有状态的规划中步骤的顺序不完全决定。规划空间方法难以有效的应用在增强学习关注的随机最优控制问题上，我们不进一步阐述(在11.6节中有一个规划空间的例子)。

我们本章提出的统一视角是所有的状态空间规划都遵循一个共同的结构，这个结构也在本书的学习方法中有介绍。剩余的章节中进一步完善这个思想，但是有两个基本的思想：1）所有的状态空间方法都涉及值函数的计算，并把它作为策略改善的一个关键中间步骤；2）值函数的计算借助于应用在仿真经验的操作的备份。这个公用架构可以图示为：

$$model \rightarrow simulated\, experience \xrightarrow{backups} values \rightarrow policy$$

动态规划方法显然满足这个结构：它遍历整个状态空间，产生每一个状态可能的转移的分布。每一个分布然后被用于计算备份值和更新状态的估计值。在本章中我们声称其他的各种各样的状态空间规划方法也满足这个结构，不同之处在于备份的种类，备份的次序以及备份信息保留时间的长短。

这种看待规划方法的方式强调了它和学习方法的联系。学习方法和规划方法的核心都是基于备份操作的值函数的估计。不同的地方是规划方法使用的是模型产生的模拟的经验，学习方法使用的环境产生的真实的经验。当然这个区别导致一些其他的差异，比如性能评估的方式和产生经验的灵活程度。但是这个相同的结构意味着很多思想和算法可以在规划和学习方法之间转换。尤其在很多情况下一个学习方法可以用规划方法的备份替代。学习方法需要经验的输入，很多情况下模拟的经验和实际的经验表现一样好。图9.1展示了一个基于一步表格式Q-规划和抽样模型随机

抽样的简单规划的例子。这个方法我们称之为随机抽样一步表格式Q规划，能和真实环境一样收敛到最优策略。

Do forever:
1. Select a state, $s \in \mathcal{S}$, and an action, $a \in \mathcal{A}(s)$, at random
2. Send s, a to a sample model, and obtain
   a sample next state, s', and a sample next reward, r
3. Apply one-step tabular Q-learning to s, a, s', r:
   $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$

图 9.1: 随机抽样的一步表格Q规划

除了对于规划和学习方法统一的框架外，本章的第二个主题是在小的增量的步骤中规划获得的效益。这使得在任何时候规划能够被打断或者被重定向，并且付出的计算代价很小，这将是有效的进行规划行为和模型学习融合的关键需求。更令人惊喜的是，在本章的后边部分我们将证明即使是纯粹的规划问题很小步骤的规划可能会是最有效的方式，一旦问题规模太大而没办法立刻解决。

9.2 规划、行为和学习的集成

当在线规划完成，随着环境的交互，产生了一些有趣的问题。新的从交互获取的信息可能会改变模型从而影响规划。对于状态和决策的某种个性化规划是值得考虑的，当前已经有了一些考虑，或者期望在不久的将来。如果决策和模型学习都是计算密集的过程，那么可用的计算资源可能要分开。在开始探讨这些问题前，我们介绍Dyna-Q，一个简单的集成了智能体在线规划主要功能的框架。在Dyna-Q框架中每个功能表现出简介琐碎的形式。在后续章节中我们建立获得每个功能的替代方式并且作一权衡。现在，我们仅仅探索阐明其思想并激发你的直觉。

在一个规划智能体中，真实经验至少有两个角色：他能用于改进模型(使之更加精确匹配真实环境)，其次借助于之前讨论的增强学习方法，实际经验能够用于改进值函数和策略。前者称为模型学习，后者称为直接增强学习。经验、模型、值函数和策略之间的可能关系总结如图9.2。每一个箭头表示影响关系和假定的改进。注意经验是如何借助模型来直接或者间接的改进值和策略。是后者，有时称之为间接增强学习会涉及到规划。

直接和间接方法都有优点和缺点。间接方法通常充分利用有限数量的经验，因而通过较少的环境交互获得更好的策略。令一方面，直接方法更简单，并且不会受到模型设计的偏差影响。一些人辩称间接方法总是优于直接方法，然而令一些人认为直接方法对于人和动物学习有更大的影响。心理学和人工智能的相关争论关系到

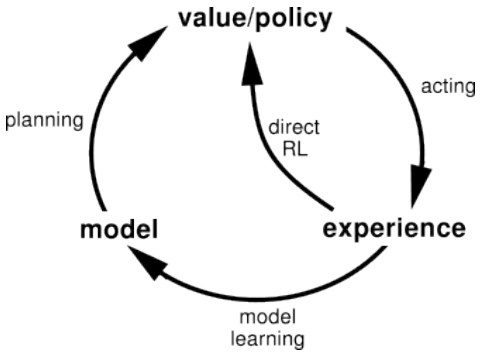


图 9.2: 学习、规划和行为的关系

认知对比试错学习以及慎重的规划对比反应式决策的相对重要性。我们的观点是在所有讨论中两种方法的对比被夸大了，可以通过二者相似性的认知而不是对立他们来获得更有洞察力的见解。比如在这本书中我们强调动态规划和时间差分方法的深层相似性，即使他们一种是为规划设计的，而另一个是无模型的学习。

Dyna-Q包括图??中的所有过程，即模型学习、规划、动作和直接增强学习，所有的都持续的发生。规划的方法就是图9.1所示的单步Q规划。直接的增强学习方法是单步表格式的Q学习。模型学习方法也是基于表格的并假设时节是确定的。在每一次转移$s_t, a_t \rightarrow s_{t+1}, a_{t+1}$，模型在表格条目里记录$s_t, a_t$并且确定的预测后续状态$s_{a+t}, a_{t+1}$。因此如果模型查询到一个已经经历的状态-行为对，它简单的返回最近一次观测到的下一个状态和回报作为它的预测。在规划中，Q规划算法随机的从已经经历过的状态-行为对中抽样，因此模型永远不会查询到一个没有信息的状态-行为对。

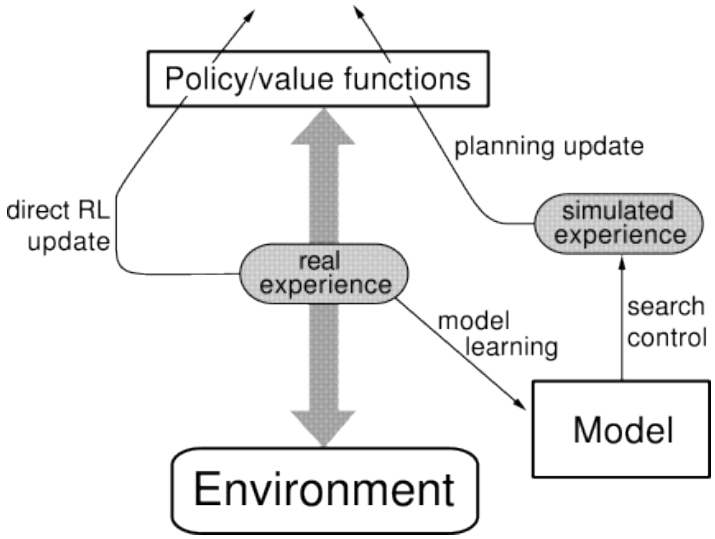


图 9.3: 通用的Dyna框架

图9.3展示了整体的Dyna智能体的框架，Dyna-Q是当中一个例子。中间的一列

代表基本的智能体和环境的交互，给出了真实经验的轨道。左边的箭头表示基于实际经验的直接增强学习，用于改进值函数和策略。右边是基于模型的过程。模型从真实经验中学习输出模拟的经验。我们用搜索控制(search control)表示从模型产生的模拟经验中选择起始状态和行为。最终，规划通过应用增强学习方法到模拟的经验实现，就好比是真实发生的经验一样。典型的，在Dyna-Q中，实际经验的学习方法和基于模拟经验的规划学习方法是一样的。因而增强学习方法最终对于学习和规划是相同的。在这个意义上学习和规划深度的集成，因为他们共享相同的机制，只是使用的经验源不同。

概念上规划、动作、模型学习和直接增强学习在Dyna智能体中同时并行的发生。然而在串行电脑的具体实施中，我们具体会有一个发生次序。在Dyna-Q中，动作、模型学习和直接增强学习需要的计算量小，我们假设他们只耗费一部分时间。每一步中剩余的时间用于规划过程，这个过程本质上计算量大。我们假设在动作、模型学习和直接增强学习之后有时间完成N次Q规划算法的迭代。图9.4给出完整的Dyna-Q算法。

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow \varepsilon$-greedy(s,Q)
(c) Execute action a; observe resultant state, s', and reward, r
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat N times:
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in s
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$

图 9.4: Dyna-Q算法。$Model(s,a)$表示对于状态-动作对s,a的模型(预测下一个状态和回报)内容。直接增强学习，模型学习和规划通过步骤(d)(e)(f)单独实现。如果步骤(e)(f)省略了，那么剩下的算法就是单步的表格式Q-学习

9.3 当模型错了

在前面章节介绍的迷宫例子中，模型的变化相对温和。模型开始是空的，然后完全被正确的信息填充。通常情况下，我们不能期望如此幸运。实际上因为环境的随机性和只有有限数量的观测例子，或者因为模型使用函数近似学习在泛化傻瓜是不完美的，或者简单的因为环境已经变化了但是新的行为还没有被观测到，从而导致了

模型的不正确。当模型不正确时，规划过程会计算得到一个次优的策略。

在某些情况下，规划计算的次优策略很快的导致模型错误的发现和修正。当模型是最优时会发生，所谓的最优模型的含义是能够比真实可能的预测更大的回报或者更好的状态转移。规划的策略试图利用这些机会并且通过这样做发现不存在的东西。

例9.2：阻塞的迷宫图9.5展示了迷宫例子中微小模型错误和恢复。初始，从起始点到障碍物的右边目标有一条短的路径，如左上图所示。在1000个时间步长后，短的路径被挡住了，在障碍物的左边开了一个更长的路径，如右上图所示。图表展示了Dyna-Q和两个其他Dyna智能体的平均累积回报。第一个部分图表表明所有三个Dyna智能体在1000步之内找到了短的路径。当环境变化了，图表变得扁平了，表示这个阶段智能体因为在障碍物附近游荡而没有获得回报。但是过了一会儿，他们会找到新的开口和新的最优行为。

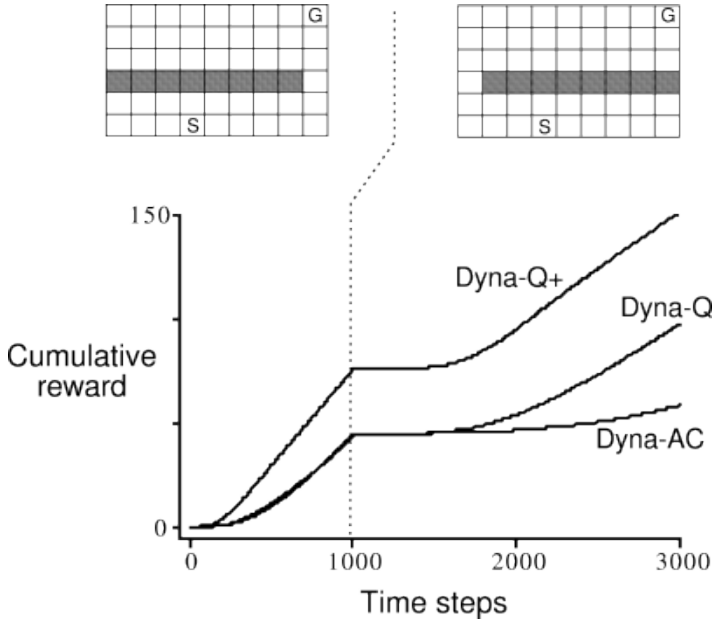


图 9.5: Dyna-Q智能体在阻塞任务上的平均表现。左边的环境用于前1000步，右边的环境是剩下的。Dyna-Q+是Dyna-Q的演化版本，更侧重探索获得的回报。Dyna-AC是一个Dyna智能体使用actor-critic学习方法而不是Q学习

当环境变得更好时产生了更大的困难，前面正确的策略不能揭示这些改善。在这些情况下模型错误长时间不会被检测到，我们看下一个例子。

例9.3：被切短的迷宫图9.6中迷宫的例子的环境表现了这种环境变化引起的问题。最开始，最优路径是走障碍物的左边（左上）。3000步之后，障碍物的右边开出了一条更短的路径，对最长路径没有影响。图表表示三个Dyna智能体中的2个自始至终没有转换到最短的路径。实际上，他们从不会意识到它的存在。他们的模型

告诉他们没有捷径，因此规划的越多，转向右边并发现捷径的可能性越小。即使采用$\varepsilon - greedy$策略，智能体也不大可能采取如此多的探索以致发现捷径。

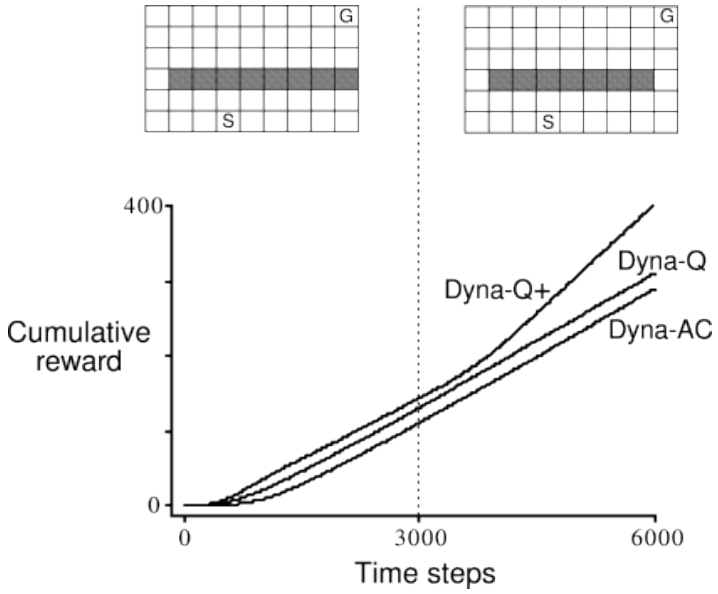


图 9.6: 在捷径问题上Dyna智能体的平均表现。左边的环境用于前3000步，剩下的是右边的环境

这个普遍的问题是探索和利用的另一个版本。在一个规划问题的语义环境中，探索意味着尝试一些行为来改善模型，然而利用意味着在当前模型下采取最优的行为。我们想要智能体探索来发现环境的变化，但是性能不会有大的分解。在初期的探索和利用冲突中，可能没有完美并且实用的解决方案，但是简单的启发式教学法通常有效。

Dyna-Q+智能体不能使用启发式思维解决有捷径的迷宫问题。智能体记录在实际环境交互中自从上一次访问一个状态-行为对后流失了多少时间步长。流失的时间越长，我们可能会假设状态对动力性发生变化和模型的不正确的可能性越大。为了鼓励探索长时间没尝试的行为，在模拟经验中涉及到这些行为时给予一个特殊的“津贴回报”。尤其如果一个转移的模型回报是r，这个转移已经有n个时间步长没有发生了，那么规划中对于该转移的回报备份值为$r + k\sqrt{n}$，k是一个很小的量。这将鼓励智能体持续测试这些易获得的状态转移甚至为了实施这些测试规划很长的行为序列。当然所有这些测试都有代价，但是很多情况下，比如在捷径迷宫的例子，这种计算价值值得多余的探索。

9.4 优先遍历

在前面部分介绍的Dyna智能体中，模拟转移开始的状态-行为对通过在所有之前经历

的对中随机均匀的选择。但是均匀选择通常不是最好的；如果模拟转移和备份专注于特殊的状态-行为对规划可能更有效率。比如，考虑在第一个迷宫任务中第二幕里发生了什么（图??）。在第二幕的开始，只有直接进入目标的状态-行为对有一个正值；其他所有对的值还是0。这意味着备份几乎所有的转移是没有意义的，因为他们把智能体从一个0 值状态带到另一个0值状态，因此备份没有影响。只有进入目标的前一个状态的值会变化。如果模拟转移均匀的产生，在进入有用的状态前需要很多浪费的备份。随着规划进行，有用备份的区域开始增长，但是和专注于最好的状态相比规划仍然效率低下。在我们实际目标的大规模问题中，状态的数量如此之大以至于一个不专注的搜索会相当的低效。

本例建议集中在从目标返回搜索可能是有益的。当然，我们实际不想使用的方法都有具体的“目标状态”的思想。我们想要能够适用于通用回报函数的方法。目标状态只是一个特殊的例子，方便激发直觉。一般的，我们想要回溯不仅仅是目标状态而是任何值发生改变的状态。假设在给定模型下肢开始是正确的，正如在迷宫例子中发现目标之前。假设现在智能体发现环境变化，而且一个状态的估计值也变了。典型的，这意味着很多其他状态的值也应该被改变，但是真正有用的单步备份是那些导致进入状态值已经变化的状态。如果这些行为的值被更新了，那么先前的状态反过来也会变化。如果这样，那么导致进入这些状态的状态也要被备份，然后他们先前状态改变了。以这种方式用户可以从任何值发生变化的状态向后工作，要么执行有益的备份要么终止传播。

作为有用备份反向传播的边界，通常增长很快，产生很多能够有益备份的状态对。但是不是所有的都一样有用。一些状态的值可能变化很大，然而其他的变化很小。后续状态变化很大的状态变化很可能也很大。在一个随机环境中，估计转移概率的变化也会导致变化幅度和需要备份的对的重要性的不同。根据状态的急迫性程度来优先备份和执行他们是自然的做法。这是优先遍历潜在的思想。内存会为每一个估计值变化不平凡的状态动作对维护一个队列，优先级根据变化的大小。当队列顶部的状态动作对被备份，它对每一个先前对影响被计算。如果这个影响大于某个小的阈值，然后这个对被插入队列中获得一个新的优先级(如果队列中之前已经有一个记录，插入会保留具有更高优先级的条目)。以这种方式变化的效果有效的往后传播知道静止。对于确定环境的情况，整个算法如图9.7所示。

迷宫的优先遍历算法研究发现在迷宫任务中优先遍历算法能够极大地增加得到最优解的速度，通常5到10的因子。图9.8展示了一个典型的例子。这些迷宫任务的数据序列和图??的完全一样，除了在网格分辨率的变化。优先遍历算法相对于无优先级的Dyna-Q具有决定性的优势。两个系统每次环境交互中至少做5次备份。

例9.5：杆操纵这个任务的目标是以最小的步数操纵一个附近有很多糟糕的障碍物的杆到一个目标位置(图9.9)。杆可以沿着轴线或者垂直轴线平移，或者可以

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow policy(s,Q)$
(c) Execute action a; observe resultant state, s', and reward, r
(d) $Model(s,a) \leftarrow s', r$
(e) $p \leftarrow |r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$.
(f) if $p > \theta$, then insert s,a into $PQueue$ with priority p
(g) Repeat N times, while $PQueue$ is not empty:
$s,a \leftarrow first(PQueue)$
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
Repeat, for all $\bar{s},\bar{a}$ predicted to lead to s:
$\bar{r} \leftarrow$ predicted reward
$p \leftarrow |\bar{r} + \gamma \max_a Q(s,a) - Q(\bar{s},\bar{a})|$.
if $p > \theta$ then insert $\bar{s},\bar{a}$ into $PQueue$ with priority p

图 9.7: 确定环境下的优先遍历算法

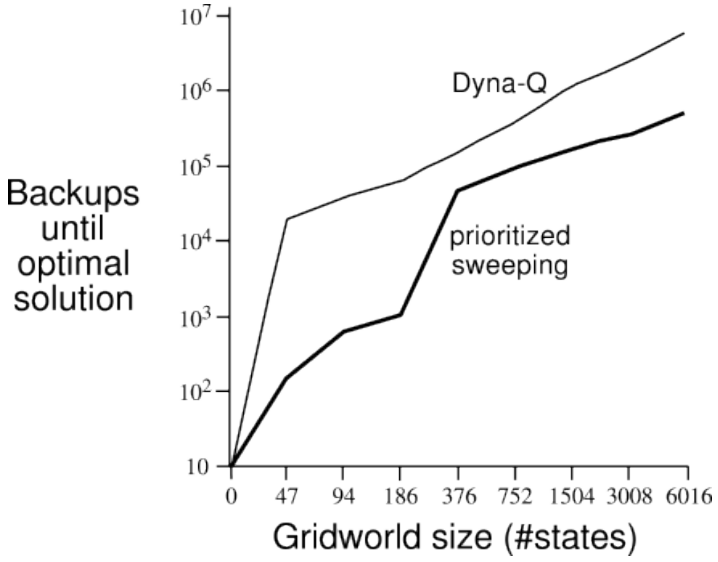


图 9.8: 优先遍历在一个大范围的网格分辨率上极大地缩短了Dyna迷宫任务的学习时间，

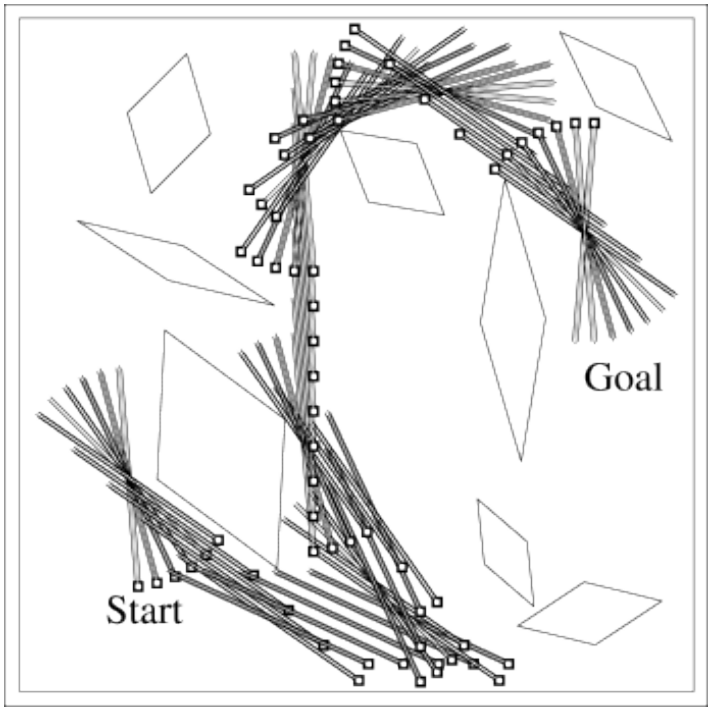


图 9.9: 优先遍历算法用于操纵杆的解

围绕轴心以任意方向旋转。每一次移动的距离大约是工作空间的1/20，旋转增量是10度。移动是确定性的量化为20×20位置中的一个。图形描绘了障碍物和使用优先遍历获得的从开始到目标的最短的方案。这个问题依旧是确定性的，但是有4个行为14400个潜在的状态(其中的一些由于障碍物无法到达)。这个问题太大而不能使用非优先方法解决。

优先遍历显然是一个强大的想法，但是目前开发的算法似乎并不容易扩展到有趣的例子中，最大的问题是算法依赖于离散状态的假设。当一个状态发生变化，这个方法对所有先前可能被影响的状态进行计算。如果值函数近似被用于模型学习或者学习值函数，那么单个备份会影响很多的其他状态。这些状态如何被鉴定或者有效的处理还不明显。另一方面，专注于那些认为是值变化的状态搜索，然后是他们先前的状态，直觉上似乎是合理的。进一步的研究可能会产生优先遍历更一般的版本。

优先遍历扩展到随机环境的情况相对直接。模型会记录每一个状态动作对已经经历的次数和下一个状态是什么。然后用全备份备份每一个对而不是一个抽样备份是很自然的，同时考虑所有可能下一个状态和发生的概率。

9.5 全备份和抽样备份比较

前面章节的例子中给出了一些集合学习和规划方法的可能的想法。本章的后面部分，我们分析一些涉及的部分思想，首先看看全备份和抽样备份的相对优势。

这本书的大部分是关于不同种类的备份，我们考虑了大量的变种。现在专注于单步备份，他们主要在3个二元维度上不同。前两个维度是他们是否备份状态值还是行为值，以及他们估计最优策略的行为值还是任意给定的一个策略。这给出了四类近似值函数的备份Q^*, V^*, Q^π, V^π。其他二元维度是备份是否是全备份，考虑所有可能发生的事件，或者抽样的备份－－只考虑单个可能发生的样本。这三种二元维度产生八类情况，当中的七种涉及到具体的算法，如图9.10所示。这些单步备份中的任何一种都能用于规划方法。Dyna-Q使用Q^*抽样备份，但是也可以是Q^*全备份，或者是Q^π的全备份或者抽样备份。Dyna-AC使用V^π抽样备份和一个学习策略结构。对于随机问题，优先遍历总是使用当中的一种全备份。

当我们在第6章介绍单步抽样备份时，我们把它作为全备份的一个替代。在没有分布模型时，全备份是不可能的实现的，但是可以用来自环境的或者抽样模型的抽样转移实现抽样备份。暗示着以那个观点来看，如果全备份可能的话将会优于抽样备份，但是事实真是如此吗？当然全备份会产生一个更好的估计因为他不会受抽样误差影响，但是它需要更多的计算，计算通常在规划中是有限的资源。为了更合适的评估规划中全备份和抽样备份的相对优点，我们必须控制它们不同的计算需求。

具体的，考虑近似Q^*的全备份和抽样备份，一个特殊的例子，离散的状态和行为，一个查找表表示近似值函数Q，一个模型估计状态转移概率为$\hat{P}^a_{ss'}$和期望回报$\hat{R}^a_{ss'}$。一个状态动作对(s,a)的全备份是：

$$Q(s,a) \leftarrow \sum_{s'} \hat{P}^a_{ss'} \big[\hat{R}^a_{ss'} + \gamma \max_{a'} Q(s',a')\big] \tag{9.1}$$

相应的给定一个抽样的下一个状态s'，对于(s,a)的抽样备份是：

$$Q(s,a) \leftarrow Q(s,a) + \alpha \big[\hat{R}^a_{ss'} + \gamma \max_{a'} Q(s',a') - Q(s,a)\big] \tag{9.2}$$

这里面α是惯例的正的步长参数，$\hat{R}^a_{ss'}$用来替换在应用无模型的Q学习中的抽样回报。

在环境是随机的含义下，全备份和抽样备份之间的差别是巨大的，一个随机环境具体是指给定一个状态和行为，有很多可能的下一个状态以不同的概率出现。如果只有一个可能的下一个状态，那么上面的全备份和抽样备份就是一样的$(\alpha = 1)$。如果有很多可能的下一个状态，那么差别可能是巨大的。支持全备份在于他是一个精确的计算，导致一个新的$Q(s,a)$，这个Q值的正确性仅受到后继状态$Q(s',a')$的限

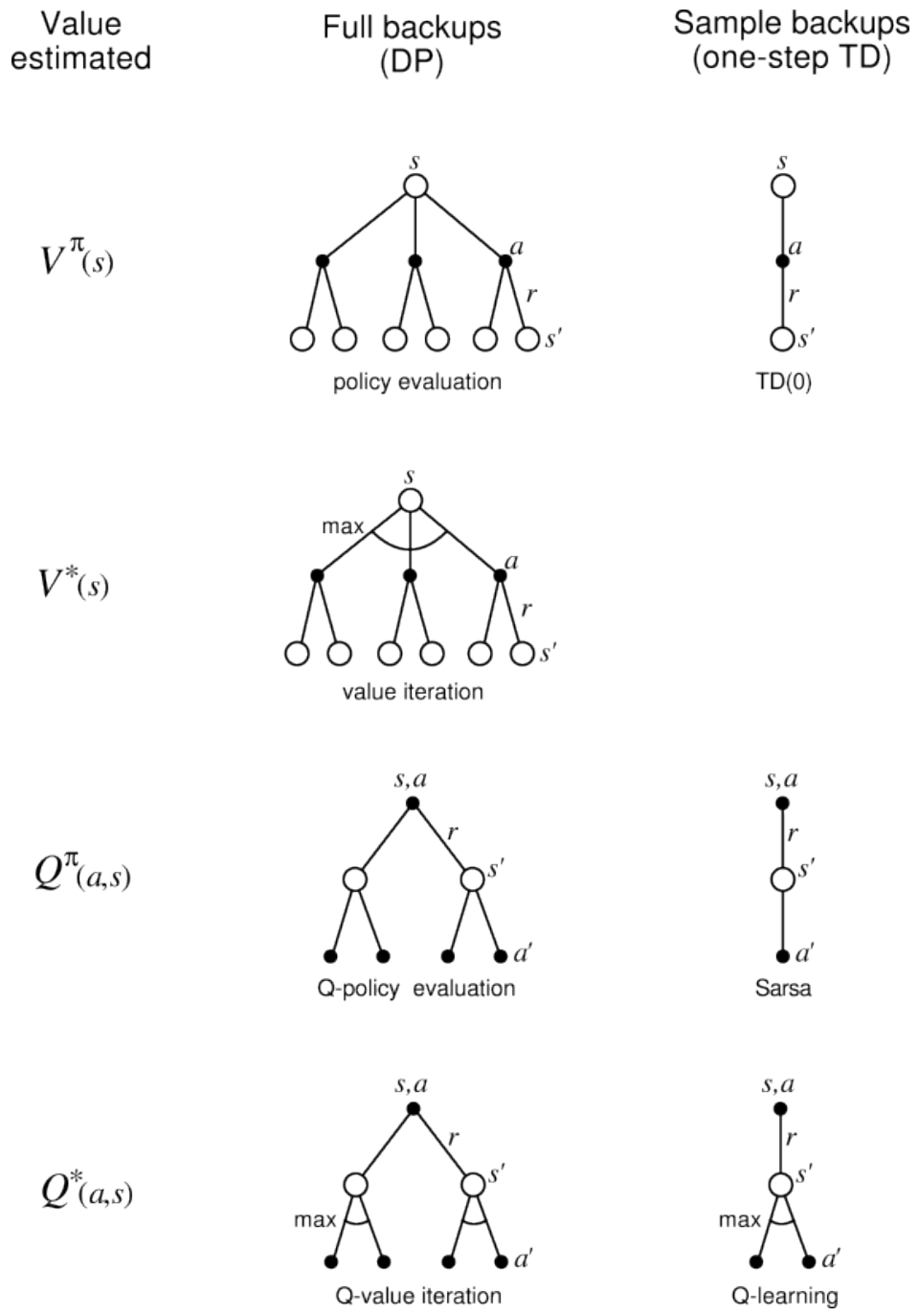

Value estimated
Full backups (DP)
Sample backups (one-step TD)
$V^{\pi}(s)$
policy evaluation
TD(0)
$V^{*}(s)$
max
value iteration
$Q^{\pi}(a,s)$
Q-policy evaluation
Sarsa
$Q^{*}(a,s)$
max
Q-value iteration
max
Q-learning

图 9.10: 单步备份

制。抽样备份除此之外受到抽样误差的影响。另一方面，抽样备份由于只考虑一个下一个状态，不是所有可能的下一个状态，所以计算代价小。实际上，备份操作需要的计算量通常有估计值Q对应的状态行为对数量决定。对于一个具体的起始对(s, a)，用b表示分支因子，指下一个可能状态s'的个数。那么对的全备份大概的计算量是抽样备份的b倍。

如果有足够的时间完成全备份，那么结果的估计通常要好于b次抽样备份，因为没有抽样误差。但是如果没有足够的时间完成一个全备份，那么抽样备份通常是可取的，因为它至少在少于b次备份中改善了值估计。在一个具有很多状态动作对的大问题中，我们通常处于后面的情况。有这么多的状态动作对，所有的都要全备份是很耗时的。在这之前基于很多状态对的少量抽样备份可能比少量状态对的全备份好的多。那么给定单位计算资源，是执行一些全备份还是执行b倍的抽样备份好？

图9.11展示的一个结果分析提供了一个这个问题的答案。它表示对于全备份和具有多种分支因子b的抽样备份估计误差作为计算时间的函数。这个实例考虑的是所有b个后继状态具有相同的可能性，并且初始估计误差是1。假设下一个状态的值是正确的，那么全备份完成之后误差减小为0。在这个实例中，抽样备份根据$\frac{1}{\sqrt{t}}\frac{b-1}{b}$减小误差，$t$表示已经执行的抽样备份数(假设平均抽样，也就是$\alpha = 1/t$)。关键的发现是对于适当大的$b$，在部分$b$次备份后误差急剧减小。对于这些情况，很多状态动作对可以显著的改善他们的值。

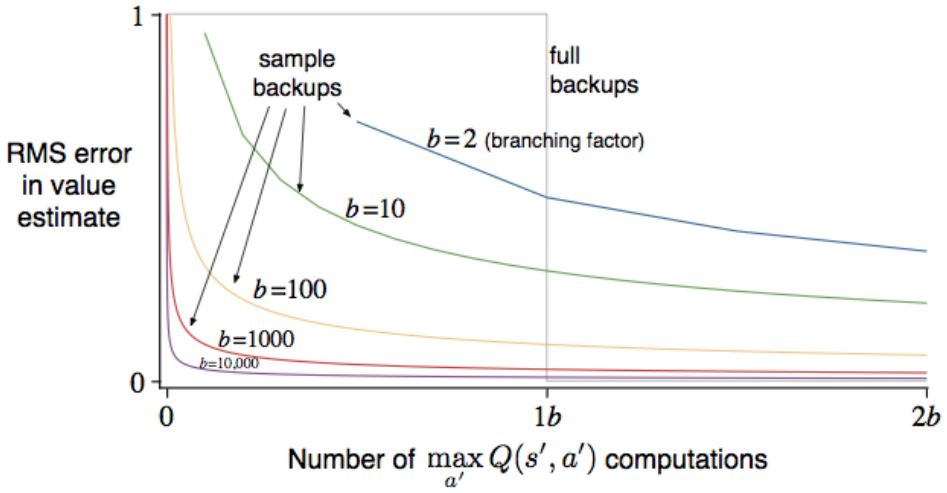


图 9.11: 全备份和抽样备份的效率对比

图9.11中表示的抽样备份的优势可能在实际效果中被低估了。在实际问题中，后继状态值本身也是通过备份估计。为了是估计更快的变得更精确，抽样备份具有第二个优势是来自后继状态的备份更精确。这些结果表情在具有大量随机分支因子和大量需要精确求解状态的问题中抽样备份可能优于全备份。

9.6 轨迹抽样

本章比较两种两种分布备份的方式。经典的来自动态规划的方式是遍历整个状态空间集合，每一次遍历都备份每一个状态。这对于规模大的任务是有问题的，因为可能连完成一次备份的时间都没有。很多状态下大量的状态都是不相关的，因为他们只有在很糟糕的策略下才会被访问或者被访问的概率很低。详尽的遍历暗示着对于每部分状态空间花费相等的时间，而不是专注在需要的地方。如第4章讨论的，详尽的遍历和言下之意平等的对待所有状态不是动态规划必须的性质。原则上，备份可以是任何你喜欢的分布方式(为了假设收敛，所有状态和状态时间对要在有限的时间内访问无限次)，但是实际中常用详尽的遍历。

第二种方法是根据一些分布从状态或者状态行为空间中抽样。可以使用均匀抽样，如Dyna-Q智能体，但是它会面临和详尽备份相同的一些问题。更有吸引力的是根据在线策略分布分配备份，也就是根据当前策略下观测到的分布。使用这个分布的优势是它容易产生；只需要与模型作用，遵照当前的策略。在片段式任务中，用户从起始状态(根据起始状态分布)模拟直到终止状态。在连续任务中，可以从任何位置开始，一直不停的仿真。无论哪一种情况，抽样状态转移和回报都由模型给定，抽样行为由当前的策略给定。换句话说，用户仿真了一个清楚的单独的轨迹，在轨迹中出现的状态或者状态行为对被执行备份。我们称这种产生经验和备份的方式叫做轨迹抽样(*trajectory sampling*)。

除了轨迹抽样外很难再想象任何有效的根据在线策略分布分配备份的方式。如果有在线策略分布的显式表示，用户可以遍历所有状态，根据在线策略分布权衡每一个备份，但是这再一次把我们带入了详尽遍历带来的高代价计算问题。用户或许可以抽样和更新来自分布的单个状态行为对，但是即使这样很高效，这会在仿真轨迹上产生多少收益？即使知道显式的在线策略分布是可能的。分布随着策略的改变而改变，计算分布需要的计算量可以相比于一个完整的策略评估。考虑其他的可能性使得轨迹抽样似乎高效并且简洁。

在线策略备份是一个好的选择吗？直觉上它似乎是，至少比均匀分布好。比如，如果你在学习下象棋，你研究实际游戏中可能出现的位置，而不是棋盘上的随机位置。后者可能是一个有效的状态，但是要精确的给他们值在现实游戏中是一个困难的技术。在第8章我们还知道在线策略分布在使用值函数近似时具有明显的优势。这是目前为止使用通用线性函数近似我们能保证收敛的唯一一个分布。无论是否使用函数近似，在线策略都会极大的提高规划速度。

专注于在线策略分布是有益的，因为它忽视了大量的无趣的状态，或者因为它导致同一部分空间重复备份变得有害。我们做了一个小实验来评估它的影响。为了隔离备份分布的影响，我们使用整个单步全表备份，如式9.1定义。在均匀的情况

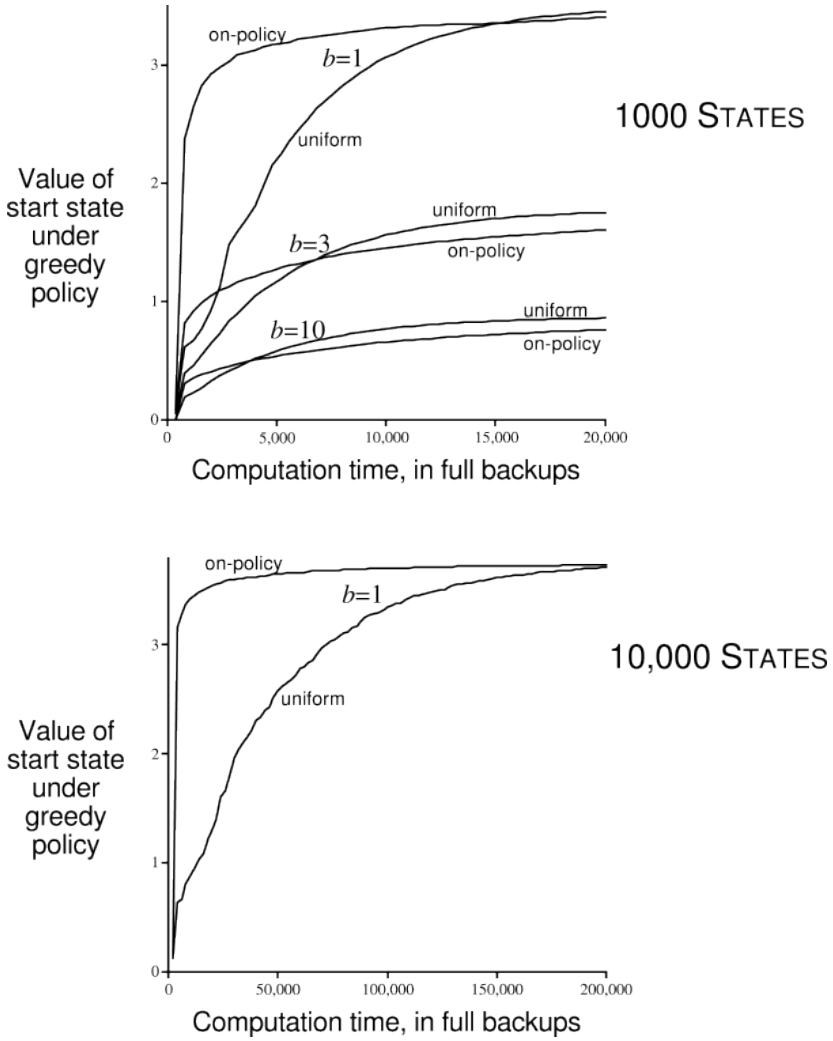


图 9.12: 状态空间上的均分分布和仿真在线策略轨迹备份的相对效率。图示了随机产生的两种大小的任务和不同分支因子b下的结果

下，我们循环的在线备份每一个状态行为对，在在线策略情况下我们仿真片段，备份那些在当前$\varepsilon - greedy$策略下出现的每一个状态行为对。任务时无折扣的片段任务，如下随机产生。每一个$|\mathcal{S}|$状态，有两个可能的行为，每一个会导致b个中的一个下一个状态发生，相等的可能性从b个状态中随机选择一个。分支因子b对于所有的状态行为对都相同。除此之外，在所有的转移中有0.1的概率进入到终止状态，结束这一幕。我们使用片段任务来清晰的衡量当前策略的品质。规划过程中的任何点上用户可以停下来计算起始状态的真实值$V^{\bar{\pi}}(s_0)$，根据值函数Q获得的策略$\bar{\pi}$。

图9.12表示了200个抽样任务、1000个状态和分支数为1,3,10时的结果。得到的策略的品质作为完成全备份的次数的函数用图描述。在所有情况下，根据在线策略抽样会导致开始的更快的规划，而后期规划变得缓慢。分支数越少，效果越强，初始阶段的快速规划越长。其他的试验中，我们发现随着状态的数量增加效果也变得越强。比如，图9.12的下图表示了分支为1，具有10000个状态的结果。在这种情况下在线策略的优势是巨大而又持久的。

所有这些结果是有意义的。短期来看，根据在线策略采样有益于集中在接近初始状态后面的状态。如果状态很多，分支数少的话，效果明显而且持久。长远来看，集中在在线策略分布可能会有害，因为经常出现的状态已经获得了正确的值。对它们的抽样是无用的，但是对其他状态的抽样可能会是有益的工作。这大概是为什么详尽的，不集中的方法在长远来看要比前面好的原因，至少是小的问题上效果要好。这些结果不是确定的，因为问题是以特殊的随机的方式产生的，但是他们确实暗示着根据在线策略分布的抽样对于大规模问题很有优势，尤其是状态行为空间中只有一个小的子集被访问的问题。

9.7 启发式搜索

在人工智能里主流的状态空间规划方法叫做启发式搜索。尽管字面上和我们目前所讲的规划方法不同，启发式搜索和它的一些分支思想能够以有益方式和这些方法结合。不像这些方法，启发式搜索不考虑改变近似值函数的值，只是基于当前的值改善行为选择。换句话说，启发式搜索作为策略计算的部分规划。

在启发式搜索里，对于每一个面临的状态，考虑一个可能的延续树。对叶子节点使用近似值函数然后想当前状态的根节点备份。搜索树里的备份和本书讨论的最大备份一样。备份在当前节点停止。一旦这些节点的备份值被计算，他们当中最好的被选为当前的行为，然后所有的备份被抛弃。

在传统的启发式搜索中不通过改变近似值函数的值来保存备份值。实际上，值函数通常认为设定并且在搜索中从不改变。但是，使用启发式搜索计算备份值或者本书中任何其他方法来考虑让值函数随着时间改善是自然的。在某种意义上我们采

取了这种方法。我们的贪婪算法或者$\varepsilon-$贪婪算法和启发式搜索是一样的，虽然只是小规模的。比如，为了计算给定模型和状态值函数下的贪婪行为，我们必须从当前可能行为往前看一步到下一个所有可能的状态，备份所有回报和估计值，然后挑选最好的行为。就像是传统的启发式搜索，这个过程计算可能行为的备份值，但是不试图保存他们。因此，启发式可以被看做是超过一步的贪婪策略思想的延伸。

搜索更深而不是单步的想法是获得更好的行为选择。如果有一个完美的模型和一个不完美的行为值函数，那么实际上深层的搜索会产生更好的策略。一般的，当搜索知道片段结束时，不完美值函数的影响会被消除，并且以这种方式决定的行为是最优的。如果搜索到足够的深度k使得γ^k非常小，那么行为会相应的接近最优。另一方面，搜索的越深，需要的计算越多，通常会导致慢的响应时间。Tesauro高水平的西洋双陆棋选手TD-Gammon 提供了一个好的例子。系统使用$TD(\lambda)$通过自我对弈学习一个afterstate值函数，使用启发式搜索的形式移动。作为一个模型，TD-Gammon使用了掷骰子概率的先验知识和假设对手总是选择它认为最好的行为。Tesauro发现启发式搜索的越深，TD-Gammon下的越好，但是每一步花的时间越长。西洋双陆棋有大量的分支因子，然而必须在几秒之内做出决策。通常可取的是只往前搜索几步，但是即使如此行为选择的结果也很好。

目前为止我们强调启发式搜索是一个行为选择技术，但是这可能它最重要的一面。启发式搜索也暗示一些选择性的分配备份的方法，这些方法可能会导致更好更快的最优值函数近似。大量的启发式搜索的研究致力于使得搜索尽可能的高效。搜索树选择性的生长，在一些线上更深，其他的更浅。比如，对于可能是最好行为的搜索树通常更深，对于那些智能体可能不想采取的行为搜索的浅。我们能够在备份分布上使用相似的思想吗？或许可以通过优先的更新那些值函数更接近最大化的状态来实现这个思想。根据我们现在的了解，这种或者其他的基于启发式搜索思想来分配备份的可能性还没有被探索。

我们不应该忽略启发式搜索集中备份的最明显的方式：集中在当前状态。启发式搜索的高效性大部分是因为搜索树密切的关注那些当前状态之后可能出现的状态和行为。在你的一生中你可能花费在象棋上的时间比西洋棋多，但是当你开始下西洋棋，考虑西洋棋和特殊棋子的位置，你最可能的下一步的动作以及后续的位置是值得的。无论你如何选择行为，这些状态和行为无疑具有最高的备份优先级，也是你希望能够精确近似的行为值。你不但需要优先考虑即将来临的事件的计算，也应该考虑有限的内存资源。比如在象棋中，远没有如此多的可能位置来存储每一个不同的值估计，但是基于启发式搜索的象棋程序可以很轻松的存储上百万的从当前位置之后面临的位置。这种大量的内存和计算资源也许是启发式搜索如此有效的原因。

备份分布也可以以相似的方式改变成集中于当前的状态和最优可能的后续状

态。在有限的情况下我们可能会利用启发式搜索来构造一颗搜索树，然后从低到上各自执行单步备份，如图9.13所示。如果备份以这种方式排序并且用一个查找表表示，那么同样的备份可以通过启发式搜索实现。

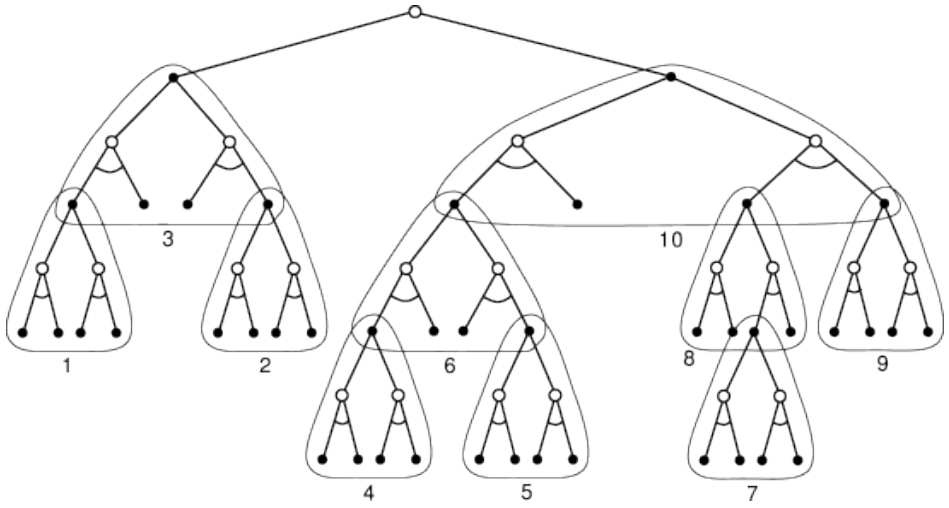


图 9.13: 启发式搜索可以通过一个单步备份序列实现。顺序为深度优先搜索

9.8 总结

我们概括的展望了规划最优行为和学习最优行为之间密切的关系。都涉及到同一值函数的估计，两种情况下增量式的更新估计，以小的备份操作是自然的做法。这使得通过学习和规划过程的集成变得简单，只需要二者同时更新一个值函数。除此之外，任何学习方法都可以转化成规划方法，通过把学习方法应用在仿真经验而非实际经验。这个意义下学习和规划变得更相似；他们可能是在不同经验源上操作的相同算法。

把增量是规划和行为、模型学习集成是直接的。规划，行为，和模型学习相互以圆的形式作用(图9.2)，每一个产生其他需要改善的；他们之间不需要其他的作用。所有过程最自然的处理方式是异步并行的。如果过程必须共享计算资源，那么可以任意的分解——根据适合并且能高效处理手头任务的任何机构。

本章中我们接触了一些不同规模的状态空间规划方法。当中最重要的一个是备份的分布，也就是搜素的专注点。优先遍历把焦点放在那些最近值发生变化的先前状态。应用于增强学习的启发式搜索尤其专注于当前状态的继承状态。轨迹抽样是集中在在线策略分布的方便的方式。所有这些方法可以极大的增加规划的速度，也是搜索里的当前话题。

另一个有趣的维度是备份的大小。备份越小，规划方法增值越快。所有备份中

最小的是单步备份。我们展示了单步备份在大问题上可能是可取的。相关的一个话题是备份深度。很多情况下深度备份可以通过一系列浅备份完成。

Chapter 10

增强学习的维度

本书中我们试图把增强学习看做跨越方法连贯的思想集，而不是单个方法的集合。每一个思想可以看做是不同方法的一个维度。浙这些维度集合跨越一个大的方法空间。通过在维度层面探索这个空间我们希望获得最广泛的和持久的理解。本章中我们使用方法空间中维度的概念来概括本书中完善的增强学习观点，也用它来辨别我们谈论领域的一些重要的差别。

10.1 统一的观点

本书中截止目前探讨的所有增强学习方法有三个关键的共同思想。首先，他们的目标都是值函数估计。第二，所有操作都是通过在实际的或者可能的状态轨迹上备份值。第三，所有的都遵循通用广义策略迭代框架(GPI)，意味着它们都为何一个近似值函数和一个近似策略，它们连续不断的依靠彼此相互改进。这3个所有方法的共同思想限制了本书涉及的主题。我们提出值函数，备份和GPI是和所有潜在的智能模型相关的强有力的组织原则。

图10.1展示了两个最重要的区分不同方法的维度。这些维度和用于改进值函数的备份种类有关。垂直维度上是他们是抽样备份(基于一个抽样轨迹)还是全备份(基于可能轨迹的分布)。过程的全备份需要一个模型，然而抽样备份可以有模型，也可以没有模型(另一个维度)。水平方向的维度对应于备份的深度，也就是自举的度。在四个角落空间中的三个是主要的值估计的三个方法：DP,TD和蒙特卡罗方法。在空间的下边缘是抽样备份方法，从单步TD备份到全回报的MC备份。二者之间的范围包括基于n步备份的方法和混合n步备份和比如通过资格迹实现的$\lambda-$备份方法。

DP方法在图片的最左上角是因为它涉及单步的全备份。右上角是全备份的情况，它的备份深度如此之深直到终止状态(或者在连续任务中，直到折扣使得进一步的回报达到一个可以忽略的程度)。这就是详尽的搜索。这两个维度之间包括启发式

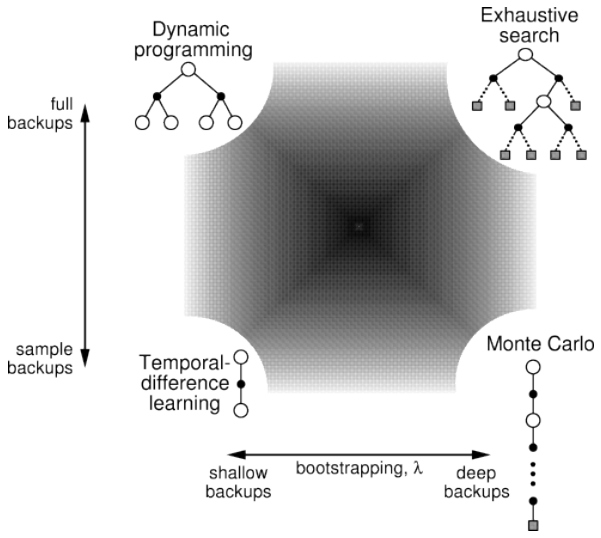


图 10.1: 增强学习的切片空间

搜索算法和相关的搜索和备份到一个有限深度的方法。也有处于垂直维度之间的方法。这些方法混合全备份和抽样备份，以及混合单步备份内的抽样和分布的可能方法。方形内部填充的区域代表所有这些方法之间的空间。

第三个重要的维度是函数近似。函数近似可以看做是一个可能性的正交谱，涉及状态聚合的极限的表格方法，大量的线性方法和各种各种的非线性方法。第三个维度可以可视化为垂直于图10.1平面。

另一个我们在这本书中着重强调的是在线策略和离线策略方法的二元区别。前者智能体通过它当前遵循的策略学习值函数，然而后者学习值函数的策略是当前它认为最好的策略。因为探索的需求这两个策略通常不同。

除了上面讨论的四个维度，我们在本书中还定义了一些其他的：

- **回报的定义**
  任务时片段的还是连续的，有折扣还是无折扣的？

- **行为值和状态值以及afterstates值**
  应该估计哪一种值？如果只有状态估计，那么还需要一个模型或者单独的策略来选择行为。

- **行为选择还是探索**
  如何选择行为来合理的权衡探索和利用？我们只考虑了最简单的方式：$\varepsilon-greedy$和softmax行为选择，和乐观的初始值。

- **同步的还是异步的**
  所有状态的备份是同时还是以一定顺序一个接一个？

- **置换迹还是累积迹**
  如果使用了迹，哪一种最合适？

- **真实的和模拟的**
  应该备份真实经验还是模拟的经验？如果都用，每一个用多少？

- **备份的位置**
  那个状态和状态行为应该被备份？无模型的方法只能选择实际面临的状态或者状态行为对，但是基于模型的方法可以任意选择。这里有很多潜在可能性。

- **备份的时间**
  备份应该作为行为选择的一部分，还是在它之后？

- **备份的记忆**
  备份值应该保留多长时间？应该永久保存，还是仅仅在行为选择计算时保存，如启发式搜索？

当然这些维度既不是详尽的也不是唯一的。单个算个在很多方面都不同，很多算法处于几个维度的几个位置。比如Dyna使用真实经验和模拟经验来影响同一个值函数。以不同的计算方法在不同的状态和行为表示上维护多重值函数是明智的。但是这些维度构成了一个凝聚性的思想集，这些思想可以用来描述或者探索广泛的可能的方法空间。