# 第4章 动态规划

本章开始介绍常见的两种免模型预测方法，**蒙特卡洛方法（ Monte Carlo，MC ）**和**时序差分方法（ Temporal-Difference，TD ）**。在讲解这两个方法之前，我们需要铺垫一些重要的概念，有模型（ Model base ）与免模型（ Model free ），预测（ Predicton ）与控制（ Control ）。

## 4.1 有模型与免模型

在前面的章节中，我们其实默认了一个事实，即状态转移概率是已知的，这种情况下使用的算法称之为**有模型算法（ Model based RL ）**，例如动态规划算法。但大部分情况下对于智能体来说，环境是未知的，这种情况下的算法就称之为**免模型算法（ Model free RL ）**，目前很多经典的强化学习算法都是免模型的。当然近年来出现了一些新的强化学习算法，例如 、 和 等。这些算法利用了神经网络和其他机器学习方法建立一个近似的环境模型，并使用规划和强化学习的方法进行决策，这些算法也都称为有模型算法。

## 4.2 预测与控制

前面提到很多经典的强化学习算法都是免模型的，换句话说在这种情况下环境的状态转移概率是未知的，这种情况下会去近似环境的状态价值函数，这其实跟状态转移概率是等价的，我们把这个过程称为**预测（ Prediction ）**。换句话说，预测是在马尔可夫决策过程 中，输入策略 ，然后输出状态价值函数 。相对的，**控制（ Control ）** 则需要我们找到一个最优策略 ，并且同时输出对应的最优状态价值函数 。之所以提到这两个概念，是因为很多时候我们不能一蹴而就解决好控制问题，而需要先解决预测问题，进而解决控制问题。

## 4.3 蒙特卡洛估计

蒙特卡洛估计方法在强化学习中是免模型预测价值函数的方式之一，本质是一种统计模拟方法，它的发展得益于电子计算机的发明。假设我们需要计算一个不规则图形的面积，这种情况下是很难通过规则或者积分的方式得到结果的。而蒙特卡洛基于这样的想法：比如我们有一袋豆子，把豆子均匀地在一定范围内朝这个图形上撒，撒到足够多的数量时数一下这个图形中有多少颗豆子，这个豆子的数目就是图形的面积。当豆子越小撒的越多的时候，结果就越精确。此时我们借助计算机程序可以生成大量均匀分布坐标点，然后统计出图形内的点数，通过它们占总点数的比例和坐标点生成范围的面积就可以求出图形面积。

那么在强化学习中蒙特卡洛方法是怎么预测状态价值函数 的呢？我们回顾 的定义公式，如式 :

由于这里的奖励函数 是环境给到智能体的， 则是设置的参数，因此其实 是可以手动计算出来的。假设机器人需要在一个 的网格中解决最短路径的问题，即从起点到终点走哪条路最短。在这种情况下，我们如果想要知道某个网格即某个状态代表的价值，那么我们可以从这个网格或状态出发，搜集到其他所有网格之间可能的轨迹。出于简化计算的考虑，我们考虑一个 的网格，如图 4.1 所示，我们用机器人的位置表示不同的状态，即，规定机器人智能体向右或者向下走，分别用 和 。起点为坐标，即初始状态为，终点为右下角的网格 ，我们设置机器人每走一步接收到的奖励为 ， 折扣因子 。

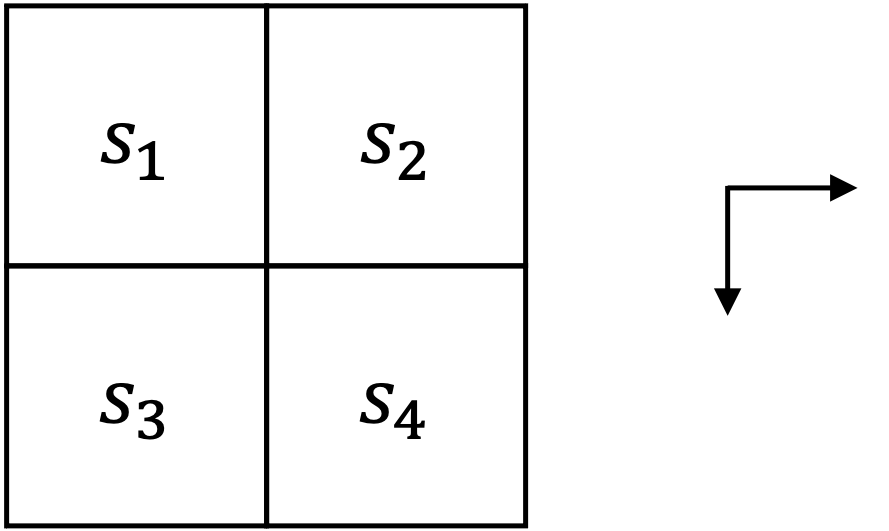


图4.1迷你网格示例

那么如果要计算 的价值 ，我们先搜集到其他状态的所有轨迹，比如到 的轨迹是 ，即从 开始，执行动作即向右之后到达 。这条轨迹记为 ，对应的回报为 ，**注意这里的下标不代表时步**。同理，到 的轨迹 为 ，对应的回报为 。到状态 的轨迹有两条，分别是 和 ，对应的回报分别为 , 。这样我们就能得到式 ：

同样的可以计算出其他状态对应的价值函数，， 由于是终止状态，因此 ，这样就得到了所有状态的价值函数分布，如图 所示。

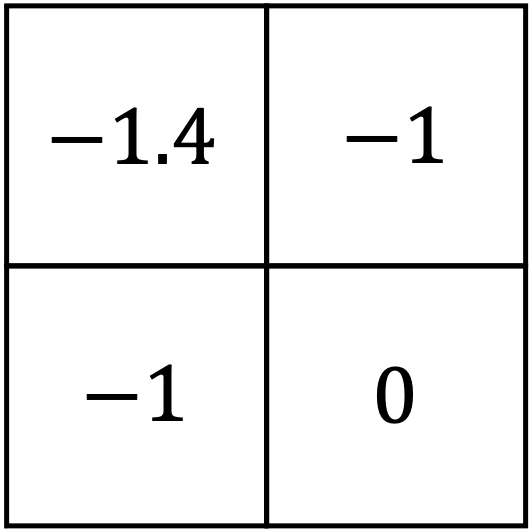


图4.2价值函数网格分布

对于状态数量更多的情况，直接按照上述的方式去计算价值函数是不现实的。蒙特卡洛方法的思路是我们可以采样大量的轨迹，对于每个轨迹计算对应状态的回报然后取平均近似，称之为经验平均回报（empirical mean return）。根据大数定律，只要采样的轨迹数量足够多，计算出的经验平均回报就能趋近于实际的状态价值函数。当然，蒙特卡洛方法有一定的局限性，即只适用于有终止状态的马尔可夫决策过程。

蒙特卡洛方法主要分成两种算法，一种是首次访问蒙特卡洛（）方法，另外一种是每次访问蒙特卡洛（）方法。 方法主要包含两个步骤，首先是产生一个回合的完整轨迹，然后遍历轨迹计算每个状态的回报。注意，只在第一次遍历到某个状态时会记录并计算对应的回报，对应伪代码如图所示。而在 方法中不会忽略统一状态的多个回报，在前面的示例中，我们计算价值函数的方式就是 ，比如对于状态 ，我们考虑了所有轨迹即 和 的回报，而在 我们只会记录首次遍历的回报，即 和 其中的一个，具体取决于遍历到 时对应的轨迹是哪一条。

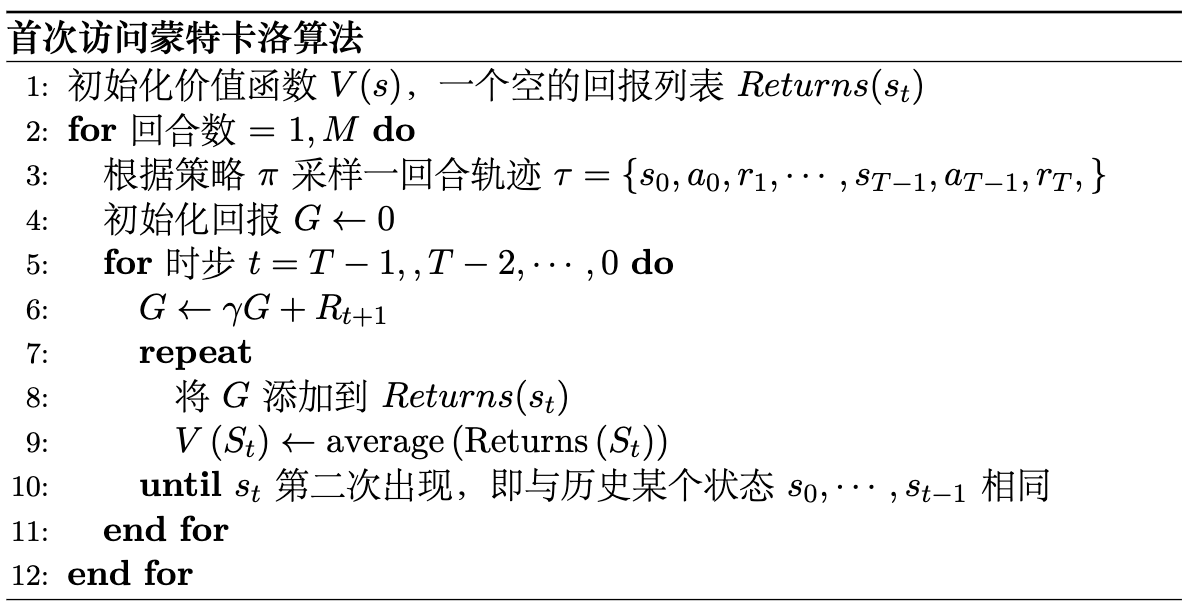


图4.3首次访问蒙特卡洛算法伪代码

实际上无论是 还是 在实际更新价值函数的时候是不会像伪代码中体现的那样 ，每次计算到新的回报 直接就赋值到已有的价值函数中，而是以一种递进更新的方式进行的，如式 所示。

这样的好处就是不会因为个别不好的样本而导致更新的急剧变化，从而导致学习得不稳定，这种模式在今天的深度学习中普遍可见，这里的步长就是深度学习中的学习率。

对应到蒙特卡洛方法中，更新公式可表示为式 ：

其中 表示学习率，为目标值与估计值之间的误差（error）。

此外， 是一种基于回合的增量式方法，具有无偏性和收敛快的优点，但是在状态空间较大的情况下，依然需要训练很多个回合才能达到稳定的结果。而 则是更为精确的预测方法，但是计算的成本相对也更高。

## 4.4 时序差分估计

时序差分估计方法是一种基于经验的动态规划方法，它结合了蒙特卡洛和动态规划的思想。最简单的时序差分可以表示为式 ：

这种算法一般称为**单步时序差分（one-step TD）**，即 。可以看到，在这个更新过程中使用了当前奖励和后继状态的估计，这是类似于蒙特卡罗方法的；但同时也利用了贝尔曼方程的思想，将下一状态的值函数作为现有状态值函数的一部分估计来更新现有状态的值函数。此外，时序差分还结合了自举（ ）的思想，即未来状态的价值是通过现有的估计 （也叫做**时序差分目标**）进行计算的，即使用一个状态的估计值来更新该状态的估计值，没有再利用后续状态信息的计算方法。这种方法的好处在于可以将问题分解成只涉及一步的预测，从而简化计算。此外， 被定义为**时序差分误差（ ）**。

但有一点需要注意的是，由于基于时步的学习方式，并且终止状态没有下一步，比如当 是终止状态时， 是没有意义的。因此时序差分方法在实践过程中会把终止状态单独做一个判断，即将对应未来状态的估计值设置为 ，然后更新当前状态的估计值，这个过程也被称作**回溯**，如式 所示，后面所有基于时序差分的方法都会有这样的一个判断。

## 4.5 时序差分和蒙特卡洛的比较

结合图 总结一下时序差分方法和蒙特卡洛方法之间的差异。

时序差分方法可以在线学习（ ），每走一步就可以更新，效率高。蒙特卡洛方法必须等游戏结束时才可以学习。

时序差分方法可以从不完整序列上进行学习。蒙特卡洛方法只能从完整的序列上进行学习。

时序差分方法可以在连续的环境下（没有终止）进行学习。蒙特卡洛方法只能在有终止的情况下学习。

时序差分方法利用了马尔可夫性质，在马尔可夫环境下有更高的学习效率。蒙特卡洛方法没有假设环境具有马尔可夫性质，利用采样的价值来估计某个状态的价值，在不是马尔可夫的环境下更加有效。

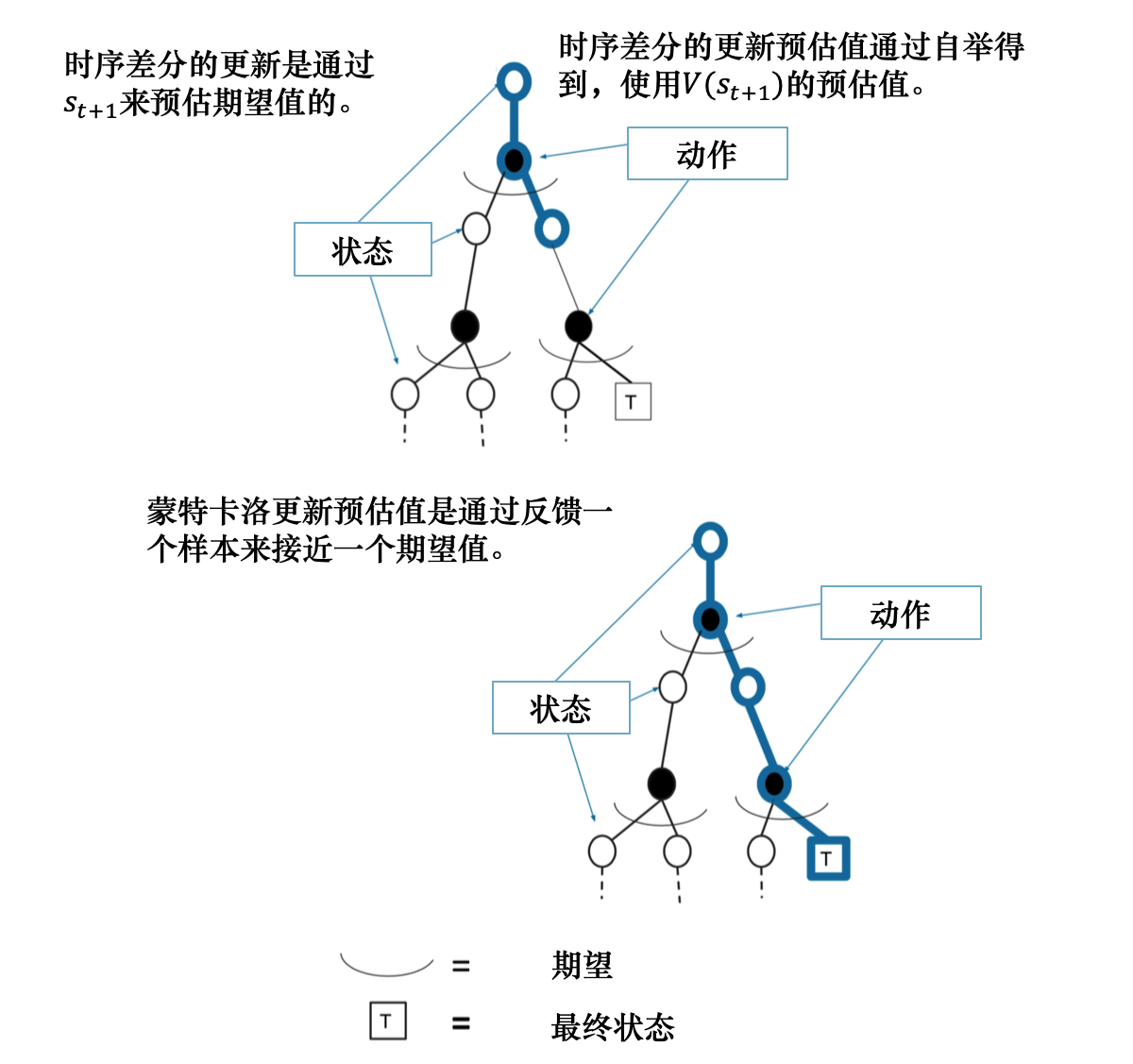


图4.4时序差分方法和蒙特卡洛方法的差异

## 4.6 n 步时序差分

把时序差分方法进一步拓展，之前只是向前自举了一步，即 ，我们可以调整为两步，利用两步得到的回报来更新状态的价值，调整 步就是 步时序差分（ ），如式 所示：

我们会发现当 趋近于无穷大时，就变成了蒙特卡洛方法，因此可以通过调整自举的步数，来实现蒙特卡洛方法和时序差分方法之间的权衡。

以下是一些常见的方法来选择合适的 :

网格搜索（ ）：对于给定的一组 值，可以通过网格搜索方法在这些值中进行遍历，并评估每个值对应的算法性能。选择在验证集上表现最好的 值作为最终的选择。

随机搜索（ ）：随机选择一组 值，在验证集上评估每个值对应的算法性能。通过多次随机搜索，可以得到更好的 值。

自适应选择：在训练过程中逐渐适应地调整 的取值。例如，可以在训练的早期使用较小的 值，以更多地依赖单步TD误差来减小偏差；在训练的后期逐渐增大 值，以更多地依赖多步回报来减小方差。

交叉验证（ ）：将数据集划分为多个子集，交叉验证不同的 值，并平均它们的性能评估结果。这样可以更好地估计不同 值的泛化性能。

经验取值：在某些情况下，根据先前的经验或已知的任务特性，可以选择一些常用的 取值作为初始值，并进一步微调。

无论使用哪种方法，重要的是要在合适的验证集上评估不同 值对应的算法性能。找到使得算法在验证集上表现最佳的 值，以确保在测试集或真实环境中获得良好的性能。

需要注意的是， 的最佳取值可能因任务、环境和算法的不同而异。因此，选择合适的 值是一个实验过程，需要根据具体问题进行调整。在实际应用中，可以结合多种方法来找到最佳的 值，以获得更好的性能。

## 4.7 本章小结

本章重点讲解了两种常见的免模型预测方法，蒙特卡洛和时序差分方法。另外涉及到了一些关键的概念，有模型与免模型，预测与控制，建议读者熟练掌握。

## 4.8 面试题

1. 有模型与免模型算法的区别？举一些相关的算法？

2. 蒙特卡洛方法和时序差分方法的优劣势