**第三章**

# Return

在一次实验中，时间步t之后的奖励记做 Rt+1, Rt+2, Rt+3, . . .,

**片段奖励**Gt:= Rt+1+ Rt+2+ Rt+3+· · ·+ RT, 式子(3.1)：

这个地方最后一个状态是否一定是终结状态呢？这个我现在也不知道，但是当面向具体的问题时，我想这不是一个问题。总之此处定义了一个片段(epsode)的奖励值。

**非终结状态**：S

**终结状态**： S+

加上折扣因子后是：

在这里插入图片描述

# MDP

满足马尔科夫属性的强化学习任务叫做**马尔科夫决策过程**markov decision process:。特别的，当状态空间和行为空间有限时，这被叫做finite markov process process，有限MDP是重要的强化学习理论，理解了finite MDP就可以解决90%的RL问题。

在Finite MDP任务中，基于当前的S，采取action a，下一个时间步的states s’和及时奖励记做式子3.6：

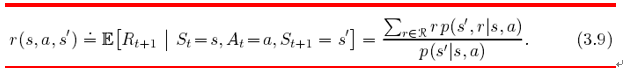
在这里插入图片描述

基于3.6式，我们可以计算更多的指标，例如及时奖励的期望值，下面的式子之所以有点奇怪，是因为考虑了转移到**相同的state，可能奖励值不一样**，如果相同是state，奖励值也一样的话，公式可以简化。

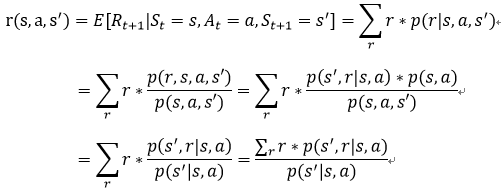
状态转移概率：

在这里插入图片描述

基于State-action-next state三元组的即时奖励期望值：



具体过程如下,：



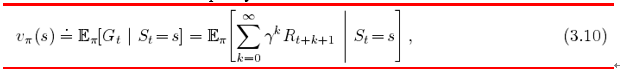
# Value function

**Policy**:是从state到action的映射。

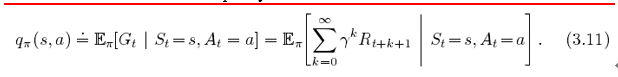
在value function的命名空间下，奖励指的是**奖励的期望值**:。

Value function是和特定的policy绑定的，只有在一定的policy控制下，才能计算相应的value function。目的是为了评估某个state或者state-action有多好，本质上是通过奖励值来评估有多好。特别的:，终结状态的奖励值为0.

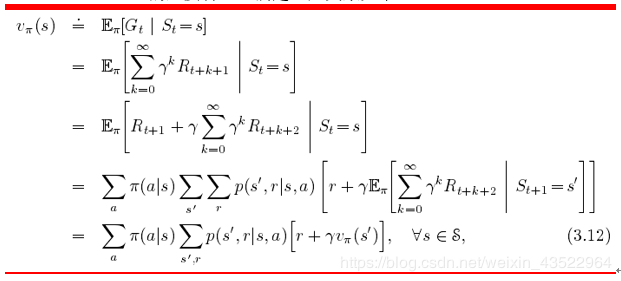
State value function for policy π。



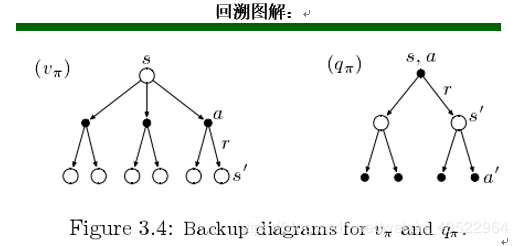
Action value function for policy π

对于Vπ和qπ可以用经验值进行估计：例如我们可以通过计算实验数据中每个状态的奖励期望，来计算state value function，假如我们独立的统计出不同action后经历的期望值，就可以估计action value function，这种就是后面会学的**蒙特卡洛**方法:。如果state非常多的话，再利用这种方法就不太实际，取而代之的是**基于参数**:的估计方法，会在本书的第二部分介绍。

Value function的重要特点是满足**递归关系**:如下：



**解释:**基于当前的状态，可以采取一个action，基于state-action pair 转移到s’并得到及时奖励。对所以可能的action奖励值加权求和就是state value function.但是根据这个公式，可以看出：book中考虑的是比较通用全面的情况：基于state可以采取不同的action，同一个action可能跳转到不同的状态，同一个状态可能获得的奖励值不同。应用到具体情况时可以简化：例如同样的action进入到相同的state，相同的state reward相同。



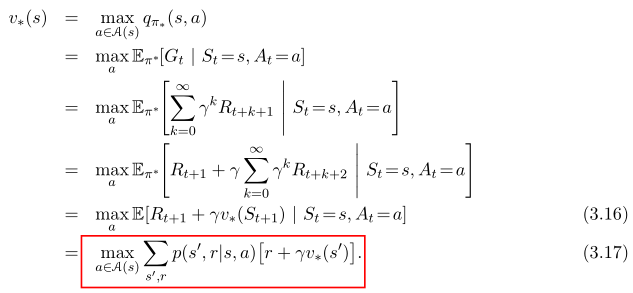
# Optimal state value function

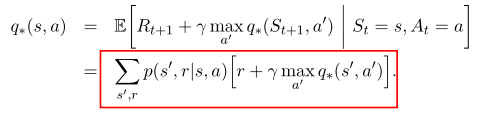
所谓最优，必然是在所有的状态可以获得的奖励值最大，最大的比较对象是其他的policy对应的value function。Policy最优→奖励最大→value function最大









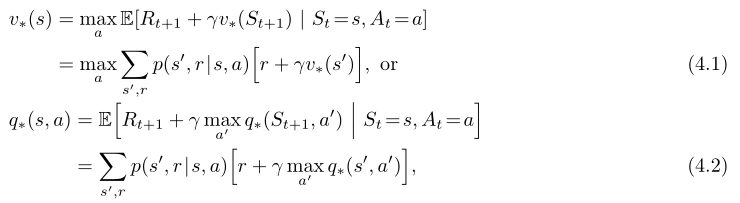


以上两个公式是optimal情况下的bellman equation，是一种特殊情况，因为不用对所有的action加权求和，而只要选择其最大的。

简单的理解就是：当前状态的最大收益，等于下一状态的奖励+及时奖励的最大值。

**第四章 动规**

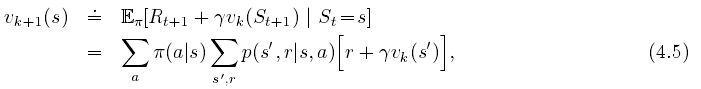
动规方法由于其本身巨大的计算消耗和过于理想的假设，而并不实用，但是对于理解其他模型仍然重要。



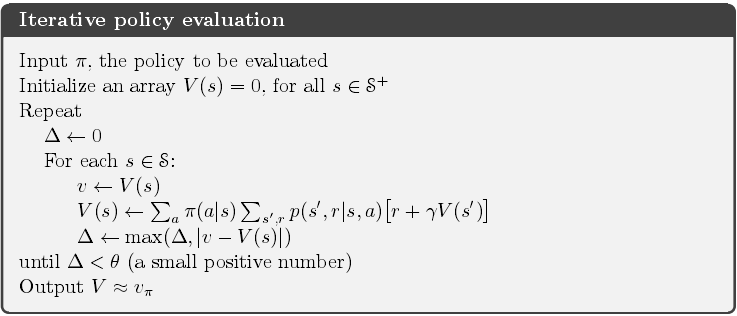
直接求最大值不就行了吗，为什么还要加期望呢？因为采取某一个action后，其状态、奖励等仍不是唯一固定的，所以求了均值。

## 4.1 policy evaluation-策略评估

随机初始化之后，根据下面的公式进行迭代。这种方法被称作full back up，因为考虑所有的后继状态，而不是对后继状态采样。



具体实现的时候有两种思路：用两个矩阵交替更新、一个矩阵原地操作。后一种方法效率更高，一般也会收敛到相同的结果。对于停止策略，可以设置一个阈值。

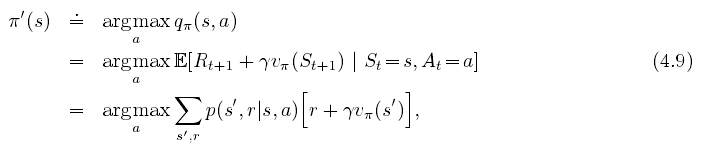


对于每一轮迭代：初始化最大步进值为0， 对每个状态先备份再更新，并尝试更新最大步进值，假如在议论迭代后最大的步进值已经非常小，就停止迭代。

## 4.2 policy improvement –策略优化

我们在前一节对某个策略的评估就是为了优化policy。一个好的policy的定义如下：





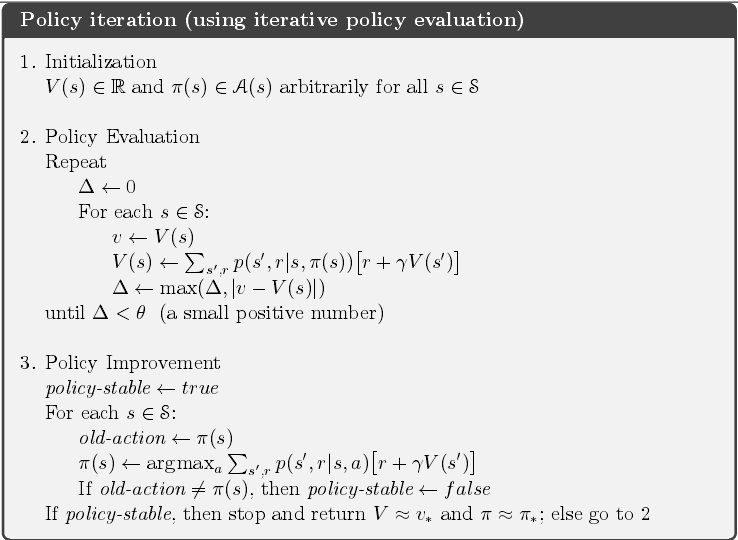
## 4.3 policy Iteration-策略迭代

通过策略优化，我们能得到一个更好的policy，基于此就可以更新value function，这是一次循环，对这个过程迭代就可以获得ptimal policy，这叫做policy iteration。

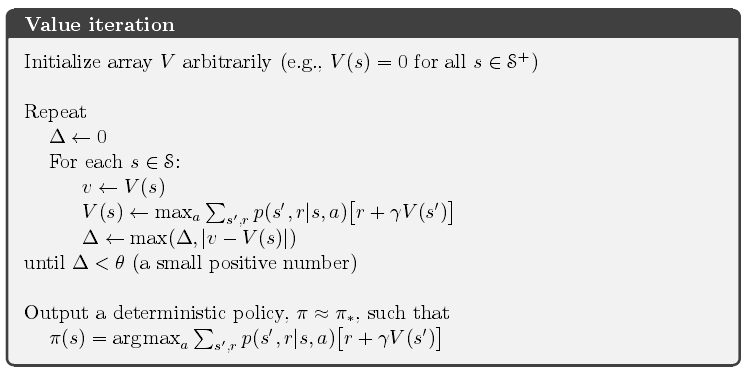


E:evaluation

I:improvement



## 4.4 value iteration



和策略迭代的区别在于：基于策略的方法会有一个π(a|s)，然后基于此再求一个期望，而直接迭代的方法不依赖policy，直接在action空间中贪心选择。

类比：value iteration就是遍历所有的可能，然后选择汇报最大的，policy可以学到一个s到a的映射。