# 策略是状态到动作的映射

# 有模型（马尔可夫决策过程）

## 有模型预测：给定策略，对所有状态的价值进行估计

将贝尔曼期望备份反复迭代

（加入了t算是动态规划）

文本

低可信度描述已自动生成

## 有模型控制：给定马尔可夫决策过程，求出最优的策略

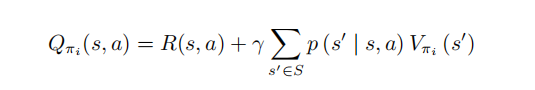
策略迭代：

第一个步骤是策略评估，当前我们在优化策略 π，在优化过程中得到一个最新的策略。我们先保证这个策略不变，然后估计它的价值，即给定当前的策略函数来估计状态价值函数。

第二个步骤是策略改进，得到状态价值函数V后，我们可以进一步推算出它的 Q 函数。得到 Q 函数后，我们直接对 Q 函数进行最大化，通过在Q 函数做一个贪心的搜索来进一步改进策略。

图示

描述已自动生成（这里评估只评一次）



图片包含 徽标

描述已自动生成

第一次的策略迭代很有意义，初始状态V=0，所以1.1时V=R，1.2时选择的a都会选择自身有R的S。这样就会导致这些与有R的S相邻的S’在2.1时也具有V。 实际上这个过程就实现了奖励（价值）的传播。

问题1：若的在此刻的policy improvement中改变了，但是我的其它与相邻的仍使用了V（，）来计算自身的Q函数，这有什么影响吗？

其实没有影响，V（，）哪怕变为了V（，），其仍然不能否认原本的（，）是具有价值的，既然有价值那当然可以用于计算。

况且V（，）必然是≥V（，）的，所以如果因为V（，）的存在导致采取了指向的动作，当然是不亏的！因为V（）更大了，这个指向

的选择显得更加明智！

价值迭代：

价值迭代直接使用贝尔曼最优方程进行迭代，从而寻找最佳的价值函数。找到最佳价值函数后，我们再提取最佳策略。（简单来说就是每步都走当前最优的a，也就是Q函数值最大的结果，然后根据走出的结果更新V值，再根据V值改进策略，不断迭代）

文本

中度可信度描述已自动生成

文本, 信件

描述已自动生成

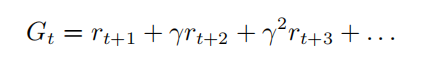
一个先评估、一个先移动

# 免模型（不知道奖励函数以及状态转移函数）

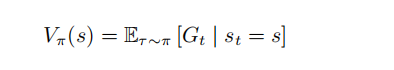
## 免模型预测（策略评估，给定了策略）

蒙特卡罗模拟（纯采样）：

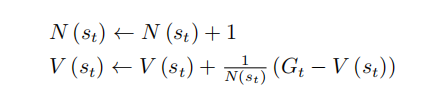
不断使用该策略采样，每次采样都可以得到一个回报G，

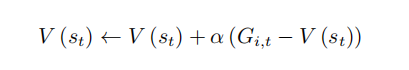


无数了G求均值即可当作该状态下的价值估计V。



对于每次采样后的V更新，有如下式子：





指定episode，不然无法停止。

时序差分（自举+采样）：

以一阶为例，每当我们走了某一步后可以得到的即时奖励 rt+1；

我们利用了自举的方法，通过之前估计的V的结果来估计 V (st+1) （初始的话这些V都是0），再加了折扣因子，即 γV (st+1)。那么对于V的估计就有：



图示

描述已自动生成

那么对于n阶时序差分，就有：

图示

描述已自动生成

墙上的钟表

低可信度描述已自动生成

其中 被称为时序差分目标，也就是我们想要逼近的值。

时序差分可以边走边更新，每走一步就更新一步。

初始状态仍然是有r的s先获得V，然后这个s会去影响周围的s’的V，逐渐的

这个α的选取很讲究，当α=1时，此时每次的更新都忘记了上一次的V，当α很小时，当下每一步都不会对整体的V产生太大的影响，最终反映出的更像是期望的结果。

通过足够多的样本数据，可以模拟出状态转移的概率分布结果。

## 免模型控制：

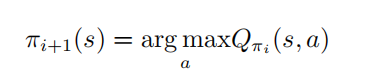
免模型控制是策略迭代的推广

1. 我们根据给定的当前策略 π 来估计价值函数V（估计方法有以上两种）

（初始的时候这一步还是V=R）；

第二，得到估计的价值函数V后，通过贪心的方法来改进策略，当有模型时：

（这时每个a都尝试！）

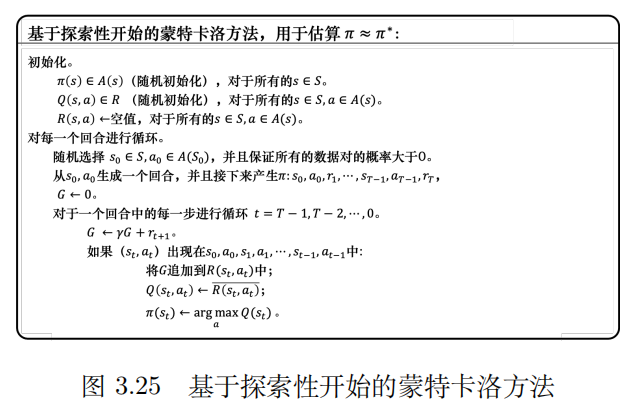


这两步不断迭代，最终得到最优的策略。

-------------------------------------------------------------------------------------------------------

现在的问题是：式中的R（s下采取a导向不同s’的概率不知道，所以奖励函数的期望也没法求）和P在免模型中均是未知的，所以哪怕评估出了V也没法算出Q函数的具体值，算不出Q函数的具体值也就无法argmax来改进策略。

1. 使用蒙特卡洛模拟对Q函数进行估计。



表达的意思就是，对所有(s，a)进行采样，基于当前的策略，产生一段轨迹（重复无数多次，把所有可能都试出来），对每条轨迹都求一个回报G，然后对所得的轨迹回报求均值来近似Q（s，a），最终从众多Q（s，）中选择最大的a作为此时改进的策略。 **是吗，没看懂其实**

为了确保蒙特卡洛能够有足够的探索，我们使用了ε-贪心（ε-greedy）探索：ε-贪心是指我们有1 − ε 的概率会按照 Q 函数来决定动作，通常 ε 就设一个很小的值，1 − ε 可能是 0.9，也就是 0.9 的概率会按照 Q 函数来决定动作，但是我们有 0.1 的概率是随机的（探索出不同的可能路径）。通常在实现上，ε 的值会随着时间递减。

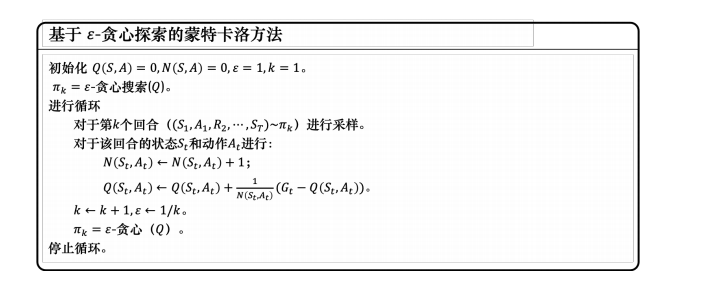
1. 探索 vs. 利用平衡。

在刚开始的时候，我们其实很难确定哪个动作是更好的，对于Q值的估计是极为不准确的，所以我们尝试不相信Q的结果，再随机走一个探索探索。

如果恰好下次评估时，这个新的尝试带来了更大收益，那么就达到了ε-贪心的目的。

（但是事实上如果评估策略与估计Q都使用的蒙特卡洛的话，引入ε-贪心并没有什么意义吧，因为哪怕我尝试了一个随机动作，这个动作无非就是我在评估Q（s，a）时的一个轨迹而已，我下使用来评估此处的价值时，得到的结果应该是和Q（s，）一致的）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | s | 1 |  |
|  |  |  |  |  |  |  |  |
|  | 100 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |



举个例子来理解ε-贪心：

首先初始化，部分地方有R值，初始策略是完全随机走；

1. 价值评估，使用免模型预测手段，评估出每个格子的价值V。
2. Q函数估计，此时策略是随即走，所以ε-贪心并未发挥作用，最后根据argmax，选出了第一次的改进策略。
3. 根据该策略重新进行价值评估。
4. 得出价值后，ε-贪心发挥作用，即此时在蒙特卡洛随机走的时候，1 – ε的概率走Q最大的方向，剩下ε的概率走其它方向

# 基于值的方法（不断更新Q表的值，最后根据Q表选取策略）

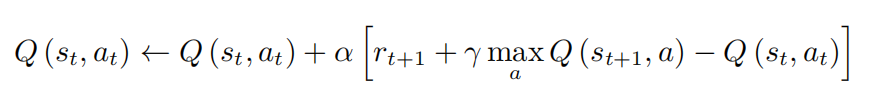
1. **Q--learning：异策略时序差分控制**

假设现在的Q表已经是：

|  |  |  |
| --- | --- | --- |
|  | A1 | A2 |
| S1 | 10 | 0 |
| S2 | 0 | 0 |
| S3 | 10 | 0 |
| S4 | 0 | 10 |
| S5 | 10 | 0 |

我们先初始随机选择一个S1，然后根据ε-贪心选择了A1，观察此时获得了奖励R=10，转移到了状态S3，那么选择s3中最大的值，为A1，那么根据：

（假设α为0.1，y为0.9） Q（S1，A1）=10 + 0.1（（10+0.9\*10）- 10）=10.9



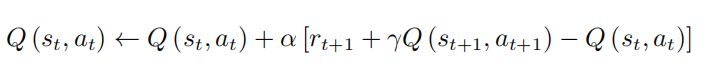
转移到状态S3，根据ε-贪心选择了，碰到了ε概率选择了值更小的A2，做A2，观察获得了奖励5，转移到了S5，那么

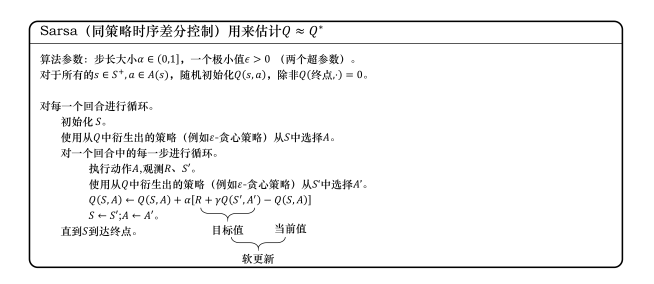
Q（S3，A2）=0 + 0.1（（5+0.9\*10）- 10）=0.3

转移到状态S5，根据ε-贪心选择了…

这样迭代下去，直至收敛。

1. **Sarsa：同策略时序差分控制**





Sarsa与Q-learning唯一的不同点在于，选择a的时候均使用了ε-贪心，即使用了下一步s’贪心所得到的a’的回报来更新此刻自己在s处采用动作a的回报。

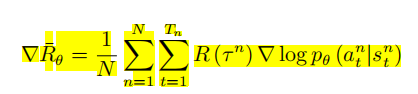
并且下一步一定会执行动作a’。

1. **深度Q网络（DQN）**

**问题：**强化学习任务所面临的状态空间往往是连续的，存在无穷多个状态，传统的Q-learning方法会遇到维度灾难问题。

# 基于策略的方法（不断更新策略，直至达到最大的）

* 1. 普通的策略梯度算法（最大化期望奖励）

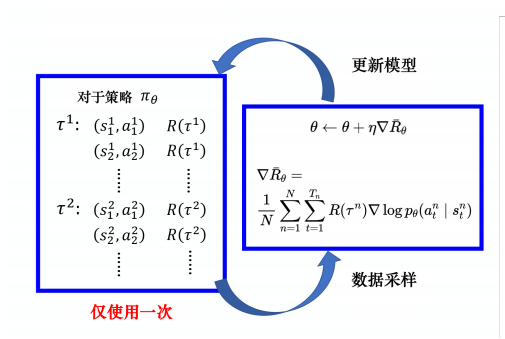


智能体拥有自己的策略网络，该网络输入一个状态s，输出执行各个动作的概率*pθ* (*a|s*1)。 （可以把策略网络理解为此时的策略）

然后演员在每个状态下，都根据*pθ* (*a|s*)的概率分布情况进行采样，采集出一条轨迹τ，并得到轨迹τ的累计奖励R (τ)，不断采集大量策略下的样本。来模拟出现在策略下期望奖励。

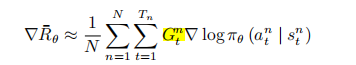
根据足够多的下的样本，来对策略网络中的参数*θ*进行一次梯度上升更新。（由于样本很多，有很多很多的(s, a)，所以更新的效力也比较大）

其结果就是那些累计奖励权重大的动作在之后的输出的概率会变得更大。



（可以配套baseline以及G(t)来优化Policy Gradient方法）

* 1. **REINFORCE：蒙特卡洛策略梯度（交叉熵方法）**



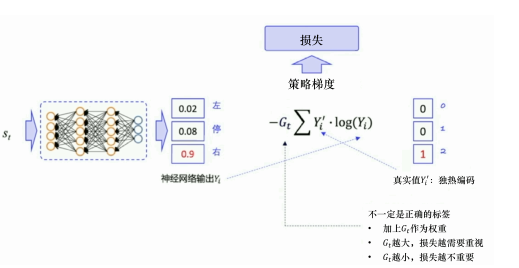
REINFORCE算法相较于一般的Policy Gradient作出了优化：

首先它使用了回报G(t)来代替了R (τ)，从宏观的整体轨迹τ转变到更加微观的动作a。这样我们不需要来采集大量下的样本，仅用一条下的轨迹就可以对那些回报高的状态-动作对完成更新（即回合更新）

文本, 信件

描述已自动生成

其次，REINFORCE算法采用了交叉熵损失函数，然而实际的动作 at 只是我们输出的真实的动作，它不一定是正确的动作，它不能像手写数字识别一样作为一个正确的标签来指导神经网络朝着正确的方向更新，所以我们需要乘一个奖励回报 Gt（该动作下的回报）。Gt 相当于对真实动作的评价。如果 Gt 越大，未来总奖励越大，那就说明当前输出的真实的动作就越好，损失就越需要重视。



图示

描述已自动生成

对每个动作都计算损失，最后各个动作的损失做加和，一起更新θ。再重新采样。

* 1. **异策略重要性采样（行为策略和目标策略）**

**问题：**策略梯度实际上是同策略的方法，因为它的采样和更新使用的是同一组θ，这样的问题是，每次更新过后θ就会改变，那么上次采样的结果对于新的θ来说是没有意义的（因为即代表了策略网络更新了策略，那么此时再采样得出的结果（结果包含(s,a)对，Gt等等都是不一样的））。

所以策略梯度的数据利用率很低，每次采样的数据只能用于θ更新一次，非常消耗时间。

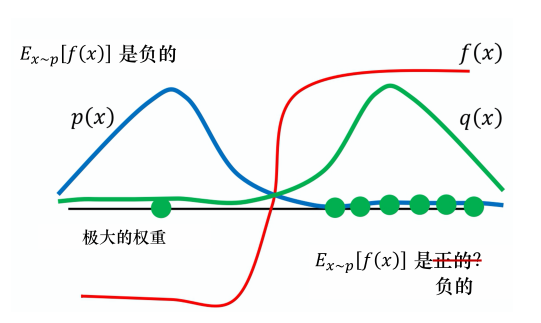
**问题补充：**一般的神经网络可以连续更新是因为，训练样本x->y，参数w的更新不会改变x的概率分布状态（假如是y=x²，如果改变了x的概率分布导致取到x=2的概率很大，那么神经网络就只会得到x=2，y=4，算不出x=3时y=9），只会不断修正Loss，所以可以反复更新。

**解决办法：重要性采样**

**核心思想：**

**1.** 使用另一个进行采样，采样的结果对θ进行更新，由于采样的结果是和θ无关的，即θ的更新并不会影响到的概率分布状态，所以此时采样的数据能对θ进行多次更新。

**2.** 加入了pθ(τ)/p(τ)这个重要性权重。因为实际上θ与之间的概率分布是不同的，所以pθ(τ)/p(τ)来修正两个分布的差异。



**举个例子来说明pθ(τ)/p(τ)项的作用：**

假设蓝线是 p(x) 的分布，绿线是q(x) 的分布，红线是 f(x)。如果我们要计算 f(x) 的期望值，从分布 p(x) 做采样，显然 Ex∼p[f(x)] 是负的。这是因为左边区域 p(x) 的概率很高，所以采样会到这个区域，而 f(x) 在这个区域是负的，所以理论上这一项算出来会是负的。

接下来我们改成从 q(x) 采样，因为 q(x) 在右边区域的概率比较高，所以如果我们采样的点不够多，可能只会采样到右侧。如果我们只采样到右侧，可能Ex∼q[f(x)]是正的。我们这边采样到这些点，去计算它们的f(x) 都是正的。我们采样到这些点都是正的，取期望值以后也都是正的，这是因为采样的次数不够多。

假设我们采样次数够多，在某一次中采样到了左边的点，因为左边的点的 p(x) 和 q(x) 是差很多的，这边 p(x) 很大，q(x) 很小。f(x) 好不容易终于采样到一个负的，这个负的就会**被乘上一个非常大的权重**，这样就可以**平衡**刚才那边一直采样到正的值的情况。

**化简后的期望奖励公式如下所示：**

图形用户界面

低可信度描述已自动生成

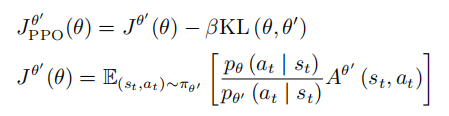
* 1. **近端策略优化PPO**

**问题：**虽然异策略的思路很好，哪怕加入了重要性采样，但是问题是*pθ* (*at |st*) 与 *pθ ′* (*at |st*)的分布依旧不能差太多，不然效果还是会很差。

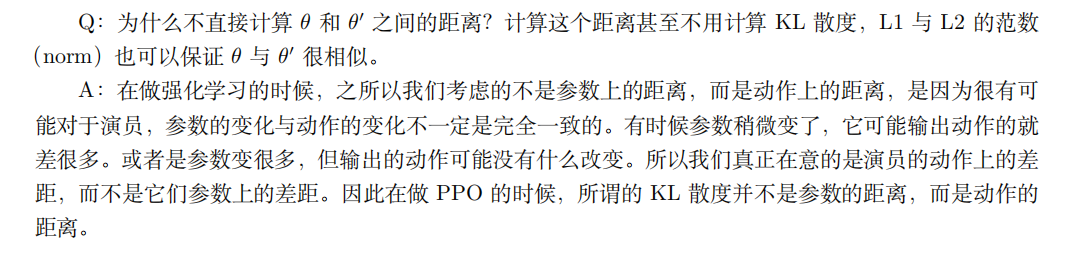
**思路：**我们希望在训练的过程中，学习出的*θ*与*θ′* 越相似越好。那么我们是否可以多加一个约束（constrain）。这个约束用于衡量*θ*与*θ′* 的相似程度。

**这就引出了*θ*与*θ′* 输出的动作的 KL 散度（KL divergence）**（用于计算两个概率分布间的差距，这里用于反映输入s，输出a的距离）

**注意1：**虽然 PPO 的优化目标涉及到了重要性采样，但其只用到了上一轮策略θ′ 的数据。PPO 目标函数中加入了 KL 散度的约束，行为策略 θ′ 和目标策略 θ 非常接近，PPO 的行为策略和目标策略可认为是同一个策略，因此 PPO 是同策略算法。



**注意2：**（关键词是 参数稍微变了，输出的动作就可能差非常多）



* + 1. **近端策略优化惩罚（PPO-penalty）**

**问题：**β 要设置为多少比较合适呢？我们需要有一个动态调整 β 的方法。

**核心思想：自适应 KL 惩罚 （adaptive KL penalty）**。

我们先设一个可以接受的 KL 散度的最大值。假设优化完以后，KL 散度的值太大，这就代表后面惩罚的项βKL没有发挥作用，我们就把 β 增大。另外，我们设一个 KL 散度的最小值。如果优化完式 (5.17)以后，KL 散度比最小值还要小，就代表后面这一项的效果太强了，我们怕他只优化后一项，使 θ 与 θk一样，而减弱了对于E(R)的优化力度，这不是我们想要的，所以我们要减小 β。

**即 如果 KL(*θ, θk* ) *>* KLmax，增大 *β*；如果 KL(*θ, θk* ) *<* KLmin，减小 *β*。**

* + 1. **近端策略优化裁剪（PPO-clip）**

**流程：**裁剪不使用KL散度来约束*θ*与*θ′*的相似程度，而是使用了min与clip裁剪函数来实现这一功能。公式如下：

文本, 信件

描述已自动生成

clip裁剪函数是指，在括号里面有 3 项，如果第一项小于第二项，那就输出 1 *− ε*；第一项如果大于第三项，那就输出 1 + *ε*。

图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成

**核心思想：**通过裁剪函数，实时控制着*θ*与*θ′*的差距，当A>0(s，a是正回报时)，我们希望扩大*pθ(at |st)*，但这个扩大的比值不能令*pθ*(*at |st*)/ *pθ′*(*at |st*)大于1+，不然两个的分布差距就太大了；A<0的时候同理。