增强学习（Reinforcement Learning and Control）

JerryLead@ISCAS
csxulijie@gmail.com
2011 年 5 月 13 日
来自 Andrew Ng Machine Learning 讲义

在之前的讨论中，我们总是给定一个样本 x，然后给或者不给 label y。之后对样本进行拟合、分类、聚类或者降维等操作。然而对于很多序列决策或者控制问题，很难有这么规则的样本。比如，四足机器人的控制问题，刚开始都不知道应该让其动那条腿，在移动过程中，也不知道怎么让机器人自动找到合适的前进方向。

另外如要设计一个下象棋的 AI，每走一步实际上也是一个决策过程，虽然对于简单的棋有 A*的启发式方法，但在局势复杂时，仍然要让机器向后面多考虑几步后才能决定走哪一步比较好，因此需要更好的决策方法。

对于这种控制决策问题，有这么一种解决思路。我们设计一个回报函数（reward function），如果 learning agent（如上面的四足机器人、象棋 AI 程序）在决定一步后，获得了较好的结果，那么我们给 agent 一些回报（比如回报函数结果为正），得到较差的结果，那么回报函数为负。比如，四足机器人，如果他向前走了一步（接近目标），那么回报函数为正，后退为负。如果我们能够对每一步进行评价，得到相应的回报函数，那么就好办了，我们只需要找到一条回报值最大的路径（每步的回报之和最大），就认为是最佳的路径。

增强学习在很多领域已经获得成功应用，比如自动直升机，机器人控制，手机网络路由，市场决策，工业控制，高效网页索引等。

接下来，先介绍一下马尔科夫决策过程（MDP，Markov decision processes）。

1. 马尔科夫决策过程

一个马尔科夫决策过程由一个五元组构成$(S, A, \{P_{sa}\}, \gamma, R)$

- S 表示状态集（states）。（比如，在自动直升机系统中，直升机当前位置坐标组成状态集）
- A 表示一组动作（actions）。（比如，使用控制杆操纵的直升机飞行方向，让其向前，向后等）
- P_{sa}是状态转移概率。S 中的一个状态到另一个状态的转变，需要 A 来参与。P_{sa}表示的是在当前$s \in S$状态下，经过$a \in A$作用后，会转移到的其他状态的概率分布情况（当前状态执行 a 后可能跳转到很多状态）。
- $\gamma \in [0,1)$是阻尼系数（discount factor）
- $R: S \times A \mapsto \mathbb{R}$，R 是回报函数（reward function），回报函数经常写作 S 的函数（只与 S 有关），这样的话，R 重新写作$R: S \mapsto \mathbb{R}$。

MDP 的动态过程如下：某个 agent 的初始状态为s_0，然后从 A 中挑选一个动作a_0执行，执行后，agent 按P_{sa}概率随机转移到了下一个s_1状态，$s_1 \in P_{s_0 a_0}$。然后再执行一个动作a_1，

就转移到了s_2，接下来再执行a_2...，我们可以用下面的图表示整个过程

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

如果对 HMM 有了解的话，理解起来比较轻松。

我们定义经过上面转移路径后，得到的回报函数之和如下

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots.$$

如果 R 只和 S 有关，那么上式可以写作

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots.$$

我们的目标是选择一组最佳的 action，使得全部的回报加权和期望最大。

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$$

从上式可以发现，在 t 时刻的回报值被打了γ^t的折扣，是一个逐步衰减的过程，越靠后的状态对回报和影响越小。最大化期望值也就是要将大的R(s_i)尽量放到前面，小的尽量放到后面。

已经处于某个状态 s 时，我们会以一定策略π来选择下一个动作 a 执行，然后转换到另一个状态 s'。我们将这个动作的选择过程称为策略（policy），每一个 policy 其实就是一个状态到动作的映射函数π：S ⟼ A。给定π也就给定了a = π(s)，也就是说，知道了π就知道了每个状态下一步应该执行的动作。

我们为了区分不同π的好坏，并定义在当前状态下，执行某个策略π后，出现的结果的好坏，需要定义值函数（value function）也叫折算累积回报（discounted cumulative reward）

$$V^\pi(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \middle| s_0 = s, \pi\right].$$

可以看到，在当前状态 s 下，选择好 policy 后，值函数是回报加权和期望。这个其实很容易理解，给定π也就给定了一条未来的行动方案，这个行动方案会经过一个个的状态，而到达每个状态都会有一定回报值，距离当前状态越近的其他状态对方案的影响越大，权重越高。这和下象棋差不多，在当前棋局s_0下，不同的走子方案是π，我们评价每个方案依靠对未来局势（R(s_1), R(s_2),...）的判断。一般情况下，我们会在头脑中多考虑几步，但是我们会更看重下一步的局势。

从递推的角度上考虑，当期状态 s 的值函数 V，其实可以看作是当前状态的回报 R(s)和下一状态的值函数 V'之和，也就是将上式变为：

$$V^\pi(s) = R(s_0) + \gamma(E[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + \cdots]) = R(s_0) + \gamma V^\pi(s')$$

然而，我们需要注意的是虽然给定π后，在给定状态 s 下，a 是唯一的，但A ⟼ S可能不是多到一的映射。比如你选择 a 为向前投掷一个骰子，那么下一个状态可能有 6 种。再由 Bellman 等式，从上式得到

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

s'表示下一个状态。

前面的 R(s)称为立即回报（immediate reward），就是 R(当前状态)。第二项也可以写作 $E_{s'\sim P_{s\pi(s)}}[V^{\pi}(s')]$，是下一状态值函数的期望值，下一状态 s'符合$P_{s\pi(s)}$分布。

可以想象，当状态个数有限时，我们可以通过上式来求出每一个 s 的 V（终结状态没有第二项 V(s')）。如果列出线性方程组的话，也就是|S|个方程，|S|个未知数，直接求解即可。

当然，我们求 V 的目的就是想找到一个当前状态 s 下，最优的行动策略π，定义最优的 V*如下：

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

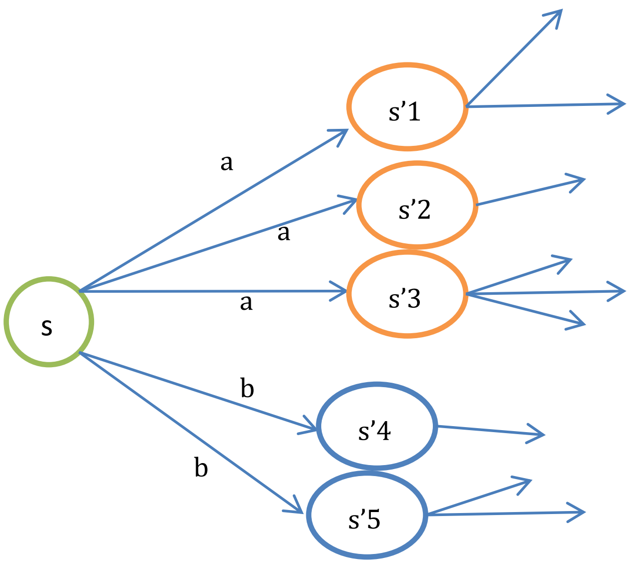
就是从可选的策略π中挑选一个最优的策略（discounted rewards 最大）。

上式的 Bellman 等式形式如下：

$$V^*(s) = R(s) + \max_{a\in A} \gamma \sum_{s'\in S} P_{sa}(s')V^*(s'). \qquad (2)$$

第一项与π无关，所以不变。第二项是一个π就决定了每个状态 s 的下一步动作 a，执行 a 后，s'按概率分布的回报概率和的期望。

如果上式还不好理解的话，可以参考下图：



定义了最优的 V*，我们再定义最优的策略$\pi^*: S \mapsto A$ 如下：

$$\pi^*(s) = \arg\max_{a\in A} \sum_{s'\in S} P_{sa}(s')V^*(s'). \qquad (3)$$

选择最优的π^*，也就确定了每个状态 s 的下一步最优动作 a。

根据以上式子，我们可以知道

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s).$$

解释一下就是当前状态的最优的值函数 V*，是由采用最优执行策略π^*的情况下得出的，

采用最优执行方案的回报显然要比采用其他的执行策略π要好。

这里需要注意的是，如果我们能够求得每个 s 下最优的 a，那么从全局来看，$S \mapsto A$的映射即可生成，而生成的这个映射是最优映射，称为π^*。π^*针对全局的 s，确定了每一个 s 的下一个行动 a，不会因为初始状态 s 选取的不同而不同。

2. 值迭代和策略迭代法

上节我们给出了迭代公式和优化目标，这节讨论两种求解有限状态 MDP 具体策略的有效算法。这里，我们只针对 MDP 是有限状态、有限动作的情况，$|S| < \infty, |A| < \infty$。

- 值迭代法

1、将每一个 s 的 V(s)初始化为 0
2、循环直到收敛 {
  对于每一个状态 s，对 V(s)做更新

$$V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$$

}

值迭代策略利用了上节中公式（2）

内循环的实现有两种策略：

1、同步迭代法

拿初始化后的第一次迭代来说吧，初始状态所有的 V(s)都为 0。然后对所有的 s 都计算新的 V(s)=R(s)+0=R(s)。在计算每一个状态时，得到新的 V(s)后，先存下来，不立即更新。待所有的 s 的新值 V(s)都计算完毕后，再统一更新。

这样，第一次迭代后，V(s)=R(s)。

2、异步迭代法

与同步迭代对应的就是异步迭代了，对每一个状态 s，得到新的 V(s)后，不存储，直接更新。这样，第一次迭代后，大部分 V(s)>R(s)。

不管使用这两种的哪一种，最终 V(s)会收敛到 V*(s)。知道了 V*后，我们再使用公式（3）来求出相应的最优策略π^*，当然π^*可以在求 V*的过程中求出。

- 策略迭代法

值迭代法使 V 值收敛到 V*，而策略迭代法关注π，使π收敛到π^*。

1、将随机指定一个 S 到 A 的映射π。
2、循环直到收敛 {
  （a） 令$V := V^{\pi}$
  （b） 对于每一个状态 s，对$\pi(s)$做更新

$$\pi(s) := arg \max_{a \in A} \sum_{s'} P_{sa}(s') V(s')$$

}

(a)步中的 V 可以通过之前的 Bellman 等式求得

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s').$$

这一步会求出所有状态 s 的V^{π}(s)。

(b)步实际上就是根据(a)步的结果挑选出当前状态 s 下，最优的 a，然后对π(s)做更新。

对于值迭代和策略迭代很难说哪种方法好，哪种不好。对于规模比较小的 MDP 来说，策略一般能够更快地收敛。但是对于规模很大（状态很多）的 MDP 来说，值迭代比较容易（不用求线性方程组）。

3. MDP 中的参数估计

在之前讨论的 MDP 中，我们是已知状态转移概率P_{sa}和回报函数 R(s)的。但在很多实际问题中，这些参数不能显式得到，我们需要从数据中估计出这些参数（通常 S、A 和γ是已知的）。

假设我们已知很多条状态转移路径如下：

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \dots$$
$$s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \dots$$
$$\dots$$

其中，$s_i^{(j)}$是 i 时刻，第 j 条转移路径对应的状态，$a_i^{(j)}$是$s_i^{(j)}$状态时要执行的动作。每个转移路径中状态数是有限的，在实际操作过程中，每个转移链要么进入终结状态，要么达到规定的步数就会终结。

如果我们获得了很多上面类似的转移链（相当于有了样本），那么我们就可以使用最大似然估计来估计状态转移概率。

$$P_{sa}(s') = \frac{\#\text{times took we action } a \text{ in state } s \text{ and got to } s'}{\#\text{times we took action a in state } s} \qquad (4)$$

分子是从 s 状态执行动作 a 后到达 s’的次数，分母是在状态 s 时，执行 a 的次数。两者相除就是在 s 状态下执行 a 后，会转移到 s’的概率。

为了避免分母为 0 的情况，我们需要做平滑。如果分母为 0，则令$P_{sa}(s') = 1/|S|$，也就是说当样本中没有出现过在 s 状态下执行 a 的样例时，我们认为转移概率均分。

上面这种估计方法是从历史数据中估计，这个公式同样适用于在线更新。比如我们新得到了一些转移路径，那么对上面的公式进行分子分母的修正（加上新得到的 count）即可。修正过后，转移概率有所改变，按照改变后的概率，可能出现更多的新的转移路径，这样P_{sa}会越来越准。

同样，如果回报函数未知，那么我们认为 R(s)为在 s 状态下已经观测到的回报均值。

当转移概率和回报函数估计出之后，我们可以使用值迭代或者策略迭代来解决 MDP 问题。比如，我们将参数估计和值迭代结合起来（在不知道状态转移概率情况下）的流程如下：

1、 随机初始化π
2、 循环直到收敛 {
   (a) 在样本上统计π中每个状态转移次数，用来更新P_{sa}和 R
   (b) 使用估计到的参数来更新 V（使用上节的值迭代方法）
   (c) 根据更新的 V 来重新得出π
   }

在(b)步中我们要做值更新，也是一个循环迭代的过程，在上节中，我们通过将 V 初始化为 0，然后进行迭代来求解 V。嵌套到上面的过程后，如果每次初始化 V 为 0，然后迭代更新，就会很慢。一个加快速度的方法是每次将 V 初始化为上一次大循环中得到的 V。也就是说 V 的初值衔接了上次的结果。

4. 总结

首先我们这里讨论的 MDP 是非确定的马尔科夫决策过程，也就是回报函数和动作转换函数是有概率的。在状态 s 下，采取动作 a 后的转移到的下一状态 s'也是有概率的。再次，在增强学习里有一个重要的概念是 Q 学习，本质是将与状态 s 有关的 V(s)转换为与 a 有关的 Q。强烈推荐 Tom Mitchell 的《机器学习》最后一章，里面介绍了 Q 学习和更多的内容。最后，里面提到了 Bellman 等式，在《算法导论》中有 Bellman-Ford 的动态规划算法，可以用来求解带负权重的图的最短路径，里面最值得探讨的是收敛性的证明，非常有价值。有学者仔细分析了增强学习和动态规划的关系。