增强学习[1]

简介

我们之前已讨论过许多的模型和算法，如果给定有标签样本集，我们就可以对数据进行回归、分类，预测新样本的值或者标签；如果给定无标签样本集，我们就可以对数据进行聚类、降维。然而对序列决策或者控制问题，我们很难给出一个规则化的数据集。比如四足机器人的行走问题，我们并不知道该如何令机器人采取『正确』的动作来行走，因此也无法给学习算法提供具有明确指导（supervision）的数据进行拟合。

在接下来我们将要讨论的增强学习（reinforcement learning）中，我们为算法提供一个奖励函数（reward function），它能够给出learning agent的动作是有利还是不利的判断。具体到四足机器人行走问题中，如果机器人向前走了一步，奖励函数会给出正奖励；如果机器人向后移动或者摔倒，奖励函数给出负奖励。学习算法的任务就是找到一系列动作以获得最大的奖励。

增强学习已经在许多领域成功应用，比如直升机自动飞行、四足机器人行走、手机网络路由、市场决策、工厂控制和高效网页索引等。接下来，让我们首先来学习马尔科夫决策过程（Markov decision process, MDP）。

1 马尔科夫决策过程

MDP可以用一个包含五个参数的元组$(S, A\{P_{sa}\}, \gamma, R)$表示，其中

- S是一系列状态的集合，例如位置坐标构成的集合。
- A是一系列动作的集合，例如朝向各个方向的运动构成的集合。
- P_{sa}是状态转换概率。对每个状态$s \in S$和动作$a \in A$而言，P_{sa}给出了在状态s下，我们执行动作a后所得新状态的概率分布。
- $\gamma \in [0, 1)$称为折扣因子（discount factor）。
- $R : S \times A \mapsto \mathbb{R}$称为奖励函数，有时候也认为奖励函数是只关于状态$S$的函数，即$R : S \mapsto \mathbb{R}$。

MDP的动态过程是这样的：我们从某状态s_0开始，并执行一个动作$a_0 \in A$，执行后，agent以状态转换概率随机转到下一个状态s_1，其中$s_1 \sim P_{s_0a_0}$。接着，我们再执行一个动作a_1转到状态s_2，其中$s_2 \sim p_{s_1a_1}$。接着……这整个过程我们可以表示为

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

我们定义上述过程的总收益（total payoff）为

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

或者

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

在大多数时候，我们会选择更简单的第二种形式。我们在增强学习中的目标是选择一系列动作以最大化总收益的期望

$$E\left[R\left(s_0\right)+\gamma R\left(s_1\right)+\gamma^2 R\left(s_2\right)+\ldots\right]$$

注意到第t步的奖励值会乘以折扣因子γ^t，所以为了使期望最大化，我们应该将奖励值大的动作放在前面，奖励值小的动作放在后面[2]。定义策略（policy）$\pi : S \mapsto A$是一个状态到动作的映射，执行该策略意味着，当我们处于状态s时，我们选择下一步的动作是$a=\pi(s)$。此外，我们定义策略π的值函数（value function）为

$$V^\pi(s)=E\left[R\left(s_0\right)+\gamma R\left(s_1\right)+\gamma^2 R\left(s_2\right)+\cdots \mid s_0=s, \pi\right]$$

值函数表达的是初始状态为s，并以策略π选择动作的总收益期望。如果我们使用确定的策略π，那么值函数满足Bellman等式

$$V^\pi(s)=R(s)+\gamma \sum_{s^{\prime} \in S} P_{s \pi(s)}\left(s^{\prime}\right) V^\pi\left(s^{\prime}\right) \qquad \text{(eq-1)}$$

也就是说值函数有两部分组成，第一部分$R(s)$是立即奖励（immediate reward），它就是我们在状态s可直接得到的奖励；第二部分是所有下一状态值函数的期望，它也可以表示成$E_{s^{\prime} \sim P_{s \pi(s)}}\left[V^\pi\left(s^{\prime}\right)\right]$，$s^{\prime}$表示的是下一状态。

在确定策略π下，假定每个状态的转换概率$P_{s \pi(s)}$已知，我们可利用Bellman等式有效地求出所有的状态的值函数$V^\pi(s)$。对于一个有限状态的MDP（$|S|<\infty$），我们对每个状态都可以得到一个Bellman等式，综合起来我们就可得到一个包含$|S|$个未知数，$|S|$个等式的线性方程组，直接求解即可。

定义最优值函数如下

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

也就是说，最优值函数会找到最优策略，从初始状态开始，我们可以由最优策略得到下一步的动作，构成一系列最优决策。最优值函数的Bellman等式形式如下

$$V^*(s)=R(s)+\max_{a \in A} \gamma \sum_{s^{\prime} \in S} P_{s a}\left(s^{\prime}\right) V^*\left(s^{\prime}\right)$$

上式第一项是立即奖励，第二项是下一状态最优值函数的期望在所有动作上的最大化。接下来定义最优策略$\pi^* : S \mapsto A$为

$$\pi^*(s)=\arg \max_{a \in A} \sum_{s^{\prime} \in S} P_{s a}\left(s^{\prime}\right) V^*\left(s^{\prime}\right) \qquad \text{(eq-2)}$$

那么对每个状态s及策略π，我们可得到

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

需要注意的是，最优策略π^*是针对所有状态s而言的，并不会因为初始状态不同就会得到不同的最优策略。

2 值迭代和策略迭代

对于有限状态、有限动作的MDP（$|S| < \infty, |A| < \infty$），我们有两种算法可以得到每个状态的最优值函数——值迭代和策略迭代。

值迭代的算法如下

1. 对每个状态s，初始化值函数为$V(s) := 0$
2. 重复以下过程直至收敛
    - 对每个状态，更新值函数为$V(s) := R(s) + \max_{a\in A} \gamma \sum_{s'} P_{sa}(s')V(s')$

算法内部的循环有两种更新方法。第一种称为同步更新，我们每次计算完一个状态的值函数后并不立即更新，而是等待所有的状态都完成了计算，最后再一起更新；第二种称为异步更新，我们每次计算完一个状态的值函数后便直接更新。无论是同步更新还是异步更新都能使V收敛到V^*，得到V^*后由(eq-2)可得π^*。

策略迭代的算法如下

1. 随机初始化π
2. 重复以下过程直至收敛
    - (a)令$V := V^\pi$
    - (b)对每个状态s，令$\pi(s) := \arg\max_{a\in A} \sum_{s'} P_{sa}(s')V(s')$

在内部循环中，步骤(a)可由(eq-1)计算出所有状态的值函数（解线性方程组）；步骤(b)会在当前状态下找出最优动作，然后对策略进行更新。在有限次数的迭代后，V会收敛到V^*，π会收敛到π^*。

值迭代与策略迭代都是解决MDP问题的标准算法，而且并没有一个标准来判断哪种是更好的算法。对于小规模的MDP，策略迭代收敛更快；对于状态空间很大的MDP，如果采用策略迭代，我们就要解一个参数、等式很多的线性方程组，这时候值迭代是一个更好的选择。

3 MDP中的参数估计

到目前为止，我们讨论的内容都假定转换概率及奖励函数已知，但在实际问题中，这些参数是未知的，我们需要去估计它们[3]。

假设我们我们尝试了状态转换的各种可能性，得到多条状态转换路径如下所示

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

其中，每一条路径代表一次试验[4]，$s_i^{(j)}$表示第j次试验中MDP的第i个状态，$a_i^{(j)}$表示第j次试验我们在状态s_i执行的动作。得到这些路径后我们可以给出状态转换概率的最大似然估计

$$P_{sa}(s') = \frac{\text{\#times we took action } a \text{ in state } s \text{ and got to } s'}{\text{\#times we took action } a \text{ in state } s} \qquad \text{(eq-3)}$$

为避免除零错误的出现，我们需要做一下拉普拉斯平滑，此时$P_{sa}(s') = 1/|S|$，这时意味着在状态s上执行动作a后会等可能性地转移到任一状态，即$P_{sa} \sim \text{Uniform}$。注意到即便进行了

新的试验，我们仍可以很容易地更新转换概率的估计，具体到(eq-3)中，我们只要累加分子分母即可。

~~用相似的办法，我们可以用在状态s的观察到的奖励的平均值作为其立即奖励$R(s)$。~~[5]

得到转换概率和奖励函数的估计后，接下来就是利用值迭代或者策略迭代解出最优值和最优策略。把参数估计和值迭代的过程结合起来，我们就可得到如下在转换概率未知情况下MDP的算法

1. 随机初始化策略π
2. 重复以下过程
    - (a)在MDP中以策略π执行某确定次数的试验
    - (b)利用(eq-3)估计转换概率P_{sa}（和R）
    - (c)利用步骤(b)中得到的参数执行值迭代过程
    - (d)由最优值V^*得到最优策略π^*

在内循环的步骤(C)中，值迭代也是一个循环的过程，如果我们每次都将V初始化为0，这样程序会做许多在之前循环中已做过的计算，十分耗时。如果我们每次将V初始化为上一步的计算结果就可以有效加速程序。

---

1. Written by Jimmy on 2016/03/18.↩
2. 这个过程跟我们下棋时的思考过程很相似，我们在下棋的时候，通常会往后推算几步，但是最看重的还是下一步局势。↩
3. 在通常情况下，$S, A\ \&\ \gamma$是已知的。↩
4. 试验的结束条件是MDP到达终结状态或者状态转移次数达到规定的上限。↩
5. 这个过程具体怎么操作讲义上并没有讲，我也没有想通，因为我感觉奖励函数是一种人为的评判。well，先打上删除线等知道的时候再来详细解释。↩