【强化学习入门 1】从零开始认识强化学习

张楚珩
清华大学 交叉信息院博士在读

目录

引言

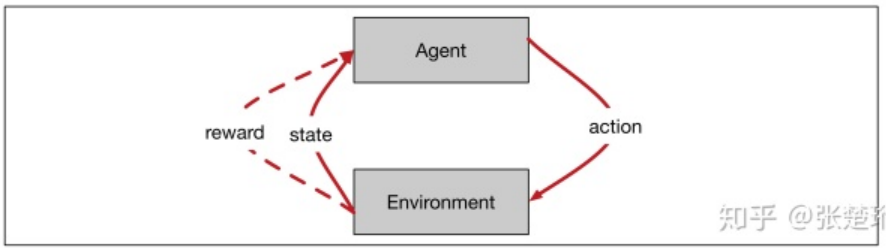
强化学习学习的是一个策略，目前主要有三大类学习的框架，它们分别是***策略迭代方法***（policy iteration method）、***策略梯度方法***（policy gradient method）和***无导数优化方法***（derivative-free optimization method）。我们在这一讲里面主要介绍强化学习的基本概念，在接下来的几讲中，我们将分别介绍这几大类方法以及其中具有代表性的算法。

强化学习基本概念

强化学习是机器学习领域的一类学习问题，它与常见的有监督学习、无监督学习等的最大不同之处在于，它是通过与环境之间的交互和反馈来学习的。正如一个新生的婴儿一样，它通过哭闹、吮吸、爬走等来对环境进行探索，并且慢慢地积累起对于环境的感知，从而一步步学习到环境的特性使得自己的行动能够尽快达成自己的愿望。再比如，这也同我们学习下围棋的模式类似，我们通过和不同的对手一盘一盘得对弈，慢慢积累起来我们对于每一步落子的判断，从而慢慢地提高自身的围棋水平。由DeepMind研发的AlphaGo围棋程序在训练学习的过程中就用到了强化学习的技术。

下面让我们来正式地定义一下强化学习问题。强化学习的基本模型就是个体-环境的交互。***个体/智能体***（agent）就是能够采取一系列行动并且期望获得较高收益或者达到某一目标的部分，比如我们前面例子中的新生婴儿或者在学习下围棋的玩家。而与此相关的另外的部分我们都统一称作***环境***（environment），比如前面例子中的婴儿的环境（比如包括其周围的房间以及婴儿的父母等）或者是你面前的棋盘以及对手。整个过程将其离散化为不同的时刻（time step）。在每个时刻环境和个体都会产生相应的交互。个体可以采取一定的***行动***（action），这样的行动是施加在环境中的。环境在接受到个体的行动之后，会反馈给个体环境目前的***状态***（state）以及由于上一个行动而产生的***奖励***（reward）。其中值得注意的一点是，这样个体-环境的划分并不一定是按照实体的临近关系划分，比如在动物行为学上，动物获得的奖励其实可能来自于其自身大脑中的化学物质的分泌，因此这时动物大脑中实现这一奖励机制的部分，也应该被划分为环境；而个体就仅仅只包括接受信号并且做出决定的部分。

上面所描述的个体-环境相互作用可以使用图一中的示意图表示。存在一连串的时刻 $t=1,2,3\cdots$ ，在每一个时刻中，个体都会接受到环境的一个状态信号 $S_t \in \mathcal{S}$ ，在每一步中个体会从该状态允许的行动集中挑选一个来采取行动 $A_t \in \mathcal{A}(S_t)$ ，环境接受到这个行动信号之后，在下一个时刻环境会反馈给个体相应的状态信号 $S_{t+1} \in \mathcal{S}^+$ 和即时奖励 $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ 。其中，我们把所有的状态记做 $\mathcal{S}^+$ ，把非终止状态记做 $\mathcal{S}$ 。



图一：个体环境相互作用

强化学习的目标是希望个体从环境中获得的总奖励最大，即我们的目标不是短期的某一步行动之后获得最大的奖励，而是希望长期地获得更多的奖励。比如一个婴儿可能短期的偷吃了零食，获得了身体上的愉悦（即，获取了较大的短期奖励），但是这一行为可能再某一段时间之后会导致父母的批评，从而降低了长期来的总奖励。在很多常见的任务中，比如下围棋，在一局棋未结束的时候奖励常常都是为零的，而仅当棋局结束的那一个时刻才会根据个体的输赢产生一个奖励值；而在另外一些任务中，环境给予奖励可能分布在几乎每个时刻中。对于像下围棋这样存在一个终止状态，并且所有的奖励会在这个终止状态及其之前结算清的任务我们称之为***回合制任务***（episodic task）；还存在另外的一类任务，它们并不存在一个终止状态，即原则上它们可以永久地运行下去，这类任务的奖励是分散地分布在这个连续的一连串的时刻中的，我们称这一类任务为***连续任务***（continuing task）。由于我们的目标是希望获得的总奖励最大，因此我们希望量化地定义这个总奖励，这里我们称之为***收益***（return）。对于回合制任务而言，我们可以很直接定义收益为

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

其中 T 为回合结束的时刻，即 S_T 属于终止状态。对于连续任务而言，不存在一个这样的终止状态，因此，这样的定义可能会在连续任务中发散。因此我们引入另外一种收益的计算方式，称之为衰减收益（discounted return）。

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

其中衰减率（discount factor） γ 满足 $0 \leq \gamma \leq 1$ 。这样的定义也很好理解，相比于更远的收益，我们会更加偏好临近的收益，因此对于离得较近的收益权重更高。

强化学习的结果是就是这样一个行动决策，我们称***策略***（policy）。策略给出了在每个状态下我们应该采取的行动，我们可以把这个策略记做 $\pi(a|s)$ ，它表示在状态 s 下采取行动 a 的概率。

如果我们可以计算出每个状态或者采取某个行动之后收益，那么我们每次行动就只需要采取收益较大的行动或者采取能够到达收益较大状态的行动。这也是策略迭代方法中的一个主要思路。在策略迭代方法中，通过对于和环境的交互，迭代地估计出到达某个状态或者采取某个行动对应的收益，同时更新策略与此匹配；通过多次迭代，当期望收益估计的越来越准的时候，策略也慢慢变好。那么自然就产生一些问题，通过迭代，是否能把对应的期望收益估计准确？当期望收益估计准确的时候相应的策略是否为最优？这些问题将在下面引入更为正式的表述之后给出解答。

马可夫决策过程

我们从大家熟悉的马可夫链说起，对于一个离散有限的状态空间 S^+ 中的每一个状态 s ，它转移到其他状态的概率只取决于该状态本身，即所说的马可夫性，即

$$Pr[S_{t+1}|S_0, S_1, \cdots, S_t] = Pr[S_{t+1}|S_t]$$

马可夫决策过程（Markov decision process, MDP）与马可夫链的区别就在于，在每个状态下，可以由个体从可能的行动空间 $A(s)$ 中选择一个行动 a ，紧接着的状态转移概率随着所选择行动的不同而不同。另外，我们再加入即时奖励，就可以得到MDP的动力学特征

$$p(s',r|s,a) = Pr[S_{t+1}=s', R_{t+1}=r|S_t=s, A_t=a] = Pr[S_{t+1}=s', R_{t+1}=r|S_0, A_0, S_1, A_1, \cdots, S_t, A_t\}$$

我们注意到，马可夫性使得我们可以仅仅利用当前状态来估计接下来的收益，即仅仅使用当前状态来估计的策略并不比使用所有历史的策略差。可以说马可夫性带给我们了极大的计算上的便利，我们不用每一步都去处理所有的历史步骤，而是只需要面对当前的状态来进行处理。同时注意到，可能有些信息并没有被完整的包含到模型的状态信号 S_t 中，这使得模型并不满足马可夫性。不过我们一般还是认为它们是***近似地***满足马可夫性的，因此仅仅使用当前状态的信息来做出对于未来收益的预测和行动的选择并不是很差的策略。同时，如果选取不仅仅是当前时刻的状态状态，而是之前的多个时刻的状态叠加在一起作为当前状态，一般可以增强马可夫性。

关于MDP更为形式化的定义是一个五元组 $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ ，而提到的MDP动力学特征 $p(s',r|s,a)$ 反映了这个MDP问题中的所有环境因素。

- $\mathcal{S}$ 表示状态空间，在***有限状态的马可夫决策过程***（finite MDP）中，该集合元素格式是有限的；
- $\mathcal{A}(s)$ 表示处于状态 s 下时，可能的行动空间，在有限状态的马可夫决策过程中，每一个集合元素个数都是有限的；
- $P(s'|s,a)$ 表示在状态 s 下，采取行动 $a \in \mathcal{A}(s)$ 时跳转到状态 s' 的概率，有关系$ $P(s'|s,a) = Pr[S_{t+1}=s'|S_t=s, A_t=a] = \sum_{r \in R} p(s',r|s,a)$ ；
- $R(s',s,a)$ 表示在状态 s 下，采取行动 $a \in \mathcal{A}(s)$ 并跳转到状态 s' 时取得的即时奖励，有关系 $R(s',s,a) = \mathbb{E}[R_{t+1}|S_t=s, A_t=a, S_{t+1}=s'] = \frac{\sum_{r \in R} r\, p(s',r|s,a)}{p(s'|s,a)}$ ；类似地，我们还可以得到在状态 s 下，采取行动 $a \in \mathcal{A}(s)$ 后的期望即时奖励， $R(s,a) = \mathbb{E}[R_{t+1}|S_t=s, A_t=a] = \sum_{r \in R} r \sum_{s' \in S} p(s',r|s,a)$ ；
- $\gamma \in [0,1]$ 表示衰减率，它指征了该MDP的目标为 $\max_{\pi} \mathbb{E}_{\pi,P,R}[\sum_{t=1}^{T} \gamma^t R_t]$ ，其中 $R_{t+1} = R(S_{t+1}, S_t, A_t)$

在下面的几个部分中，我们讨论的MDP都是有限状态的马可夫决策过程。

价值函数

接下来我们介绍强化学习领域非常重要的两个函数***状态价值函数***（state value function） $V_\pi(s)$ 和***行动价值函数***（action value function） $Q_\pi(s,a)$ ，其中函数 $V_\pi(s)$ 表示的是当到达某个状态 s 之后，如果接下来一直按着策略 π 来行动，能够获得的期望收益；函数 $Q_\pi(s,a)$ 表示的是当达到某个状态 s 之后，如果采取行动 a ，接下来再按照策略 π 来行动，能获得的期望收益。它们的具体定义如下。

$$V_\pi(s) = \mathbb{E}_\pi[G_0|S_0=s] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1}|S_0=s]$$

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_0|S_0=s, A_0=a] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1}|S_0=s, A_0=a]$$

递归地展开上面这两个公式即可得到相应的两个***Bellman方程***（Bellman equation），比如对于上述关于 $V_\pi(s)$ 的式子，有

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_{t+1}|S_0=s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{t=0}^{\infty} \gamma^t R_{t+2}|S_1=s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)[r + \gamma \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_{t+2}|S_1=s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')] \\ &\forall s \in \mathcal{S} \end{aligned}$$

这就是关于状态价值函数的Bellman方程，它描述的是当前状态的价值函数和它后续状态价值函数之间的递归关系。同理可以得到关于行动价值函数的Bellman方程。

$$Q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s',a')]$$

那么对于一个有限状态的马可夫决策过程来说，是否存在一个最优的策略呢？如果一个策略 π' 在任何状态 $s \in \mathcal{S}$ 下都有 $V_\pi(s) \le V_{\pi'}(s)$ ，那么我们称策略 π' 优于 π 。对于所有的策略排序，我们总可以找到最优策略（有可能不止一个最优策略），我们记最优策略为 π^* ，最优策略具有如下性质

$$\begin{aligned} V^*(s) &= \max_\pi V_\pi(s) \\ Q^*(s,a) &= \max_\pi Q_\pi(s,a) \end{aligned}$$

使用上述类似的方法，我们可以得到***最优情况下的Bellman方程***（Bellman optimality equation）

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)[r + \gamma V^*(s')]$$
$$Q^*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} Q^*(s', a')]$$

显然地，状态价值函数和行动价值函数之间有关系 $V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s, a)$ ， $V^*(s) = \max_a Q^*(s, a)$ 。

Bellman算子

在前一节中我们定义了两种价值函数，如果能够求得在某策略下的价值函数，我们就可以对该策略的好坏进行评估；如果能够求得最优情形下的价值函数，我们就可以自然得出最优策略。但我们还不知道要如何计算得到相应的价值函数数值，我们这里引入的***Bellman算子***（Bellman operator/Bellman backup operator）及其相关的性质，可以为相关算法的收敛性提供一些保证。

考虑有限的状态空间 $\mathcal{S} = [n]$ 和一个定义在状态集上的实值函数 V ，状态空间给定的时候该实值函数也可以写成向量的形式 $V \in \mathbb{R}^n$ 。定义Bellman算子 $\mathcal{T} : \mathbb{R}^n \to \mathbb{R}^n$ ，

$$(\mathcal{T}V)_i = \max_{a \in \mathcal{A}(i)} \sum_{j \in [n]} P(j|i, a)(R(j, i, a) + \gamma V_j)$$

其中下标都表示向量的某个元素。对于一个策略 π 还可以定义Bellman算子 $\mathcal{T}_\pi : \mathbb{R}^n \to \mathbb{R}^n$ ，

$$(\mathcal{T}_\pi V)_i = \sum_{j \in [n]} P(j|i, \pi)(R(j, i, \pi) + \gamma V_j)$$

对于确定性的策略来说里面的 π 就是 $\pi(i)$ ；对于随机策略来说，还需要在外面对策略给出的行动求期望，这里将其省略。

Bellman算子具有以下性质：

- Bellman算子 $\mathcal{T}$ 和 $\mathcal{T}_\pi$ 在max norm下都是按系数（modulus） γ 收缩的（contraction），即 $\|\mathcal{T}V - \mathcal{T}U\|_\infty \le \gamma \|V - U\|_\infty$ 。
- Bellman算子 $\mathcal{T}$ 具有唯一不动点 V^* ，即 $\mathcal{T}V^* = V^*$ ；Bellman算子 $\mathcal{T}_\pi$ 也具有唯一不动点 V_π ，即 $\mathcal{T}_\pi V_\pi = V_\pi$ ， V^* 和 $\mathcal{T}_\pi$ 的定义与前一节相同。前一个性质加上Banach fixed point theorem可以推导到此性质。
- Bellman算子 $\mathcal{T}$ 和 $\mathcal{T}_\pi$ 单调（monotonic），即 $V \le \mathcal{T}V \Rightarrow V \le \mathcal{T}V \le \mathcal{T}^2 V \le \cdots \le V^*$ ，其中对于向量来说 $\le$ 表示每一个元素都小于等于。
- 当 $\mathcal{T}V = V$ 时， $V = V^*$ ；当 $\mathcal{T}_\pi V = V$ 时， $V = V_\pi$ ；当 $\mathcal{T}V = \mathcal{T}_\pi V = V$ 时， π 是最优策略。
- 对任意的 $V \in \mathbb{R}^n$ ， $\lim_{k \to \infty} \mathcal{T}^k V = V^*$ 。

Value Iteration和Policy Iteration

给出两种找到最优值函数 V^* 的算法，分别是***值迭代***（Value Iteration，VI）和***策略迭代***（Policy Iteration，PI），最后给出***改进的策略迭代***（Modified Policy Iteration）是这两者的统一。

算法一：**Value Iteration**

1 **initialise** $V^{(0)}$
2 **for** $t = 0, 1, \cdots, T$ **do**
3 $\quad V^{(t+1)} \leftarrow \mathcal{T}V^{(t)}$

值迭代有如下性质：

- 如果初始化的时候 $V^{(0)}$ 的每一个元素都置0，那么有 $V^{(0)} \le V^{(1)} \le \ldots$ 。
- $\|V^{(n)} - V^*\|_\infty \le \gamma^n \|R\|_\infty / (1 - \gamma)$ 。
- 在 $T \to \infty$ 的时候，能够得到 $V = V^*$ 。

算法二：**Policy Iteration**

1 **initialise** $V^{(0)}$
2 **for** $t = 0, 1, \cdots, T$ **do**
3 $\quad$ **find** π_t, **s.t.** $\mathcal{T}_{\pi_t} V^{(t)} = \mathcal{T}V^{(t)}$ **(i.e greedy policy)**
4 $\quad V^{(t+1)} \leftarrow \lim_{k \to \infty} \mathcal{T}_{\pi_t}^k V^{(t)}$

策略迭代分为两个步骤交替进行：一步称作***策略改进***（policy improvement），即固定价值函数，并且找到在该价值函数下的最优策略（第3行）；另一步称作***策略估计***（policy evaluation），即固定策略，找出该策略下的价值函数（第4行）。这样的更新方式同样具有类似值循环的收敛和界的性质。这里出现的 $\lim_{k \to \infty}$ 在实际操作中通常是循环直到 $\|V\|_\infty$ 小于某个设定的小量。

算法三：**Modified Policy Iteration**

1 initialise $V^{(0)}$
2 for $t = 0, 1, \cdots, T$ do
3 find π_t, s.t. $\mathcal{T}_{\pi_t} V^{(t)} = \mathcal{T} V^{(t)}$ (i.e greedy policy)
4 $V^{(t+1)} \leftarrow \mathcal{T}_{\pi_t}^{m_t} V^{(t)}$

改进的策略迭代是对于上面两种算法的统一：当 $m_t = 1$ 时，该算法就是值循环；当 $m_t = \infty$ 时，该算法就是策略迭代。该算法的收敛性能够由Bellman算子的性质保证。值迭代也可以看做策略迭代的策略评价步只进行了一次迭代，还没有完全收敛到该策略的价值函数的时候，又进行了下一步的策略改进步。实际上，这并不影响算法的收敛，并且常常会更快；更进一步，在第3、4行的循环中，甚至不需要每次都更新所有的状态（即 V 的所有位置），该算法同样有效。这类交替甚至异步进行估计 V_π 、改进 π 的算法思想，我们称之为***广义策略迭代***（generalized policy iteration）。

动态规划方法

动态规划（dynamic programming, DP）算法其实就是直接从前面的值循环得到的，得到最优情形下的状态价值函数之后，输出相应的最优策略。下面给出动态规划算法。

算法四：**Dynamic Programming**

1 initialise array V arbitrarily (eg. $V(s) = 0, \forall s \in \mathcal{S}^+$)
2 repeat
3 $\Delta \leftarrow 0$
4 for each $s \in \mathcal{S}$
5 $v \leftarrow V(s)$
6 $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
7 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
8 until $\Delta < \theta$ (a small positive number)
9 output a deterministic policy, $\pi \approx \pi_*$, such that
10 $\pi(s) = arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

动态规划算法也可以看做把前面推导的Bellman方程化为了增量的形式进行计算，当算法收敛到不动点的时候，这时得到的状态价值函数就满足最优情形下的Bellman方程，对应的贪心策略就是最优策略。其中，核心的算法在第6行，它迭代地将状态价值函数 $V(s)$ 逼近最优状态价值函数 $V^*(s)$ 。算法的循环中没有显式地定义每一步对应的策略，我们可以对于每一步迭代的价值函数 $V(s)$ 都定义相应的贪心策略。利用以下定理，我们可以知道，经历每一步的迭代，之后策略都不比之前的策略差；从而当算法收敛的时候，我们可以得到最优策略。

Policy Improvement Theorem
如果存在一对确定性的策略π和π'，使得$q_\pi(s, \pi') \geq v_\pi(s), \forall s \in \mathcal{S}$，那么$\pi'$不比$\pi$差。
如果存在一对确定性的策略π和π'，使得$v'_\pi(s) \geq v_\pi(s), \forall s \in \mathcal{S}$，那么$\pi'$不比$\pi$差。

上述算法的第6行中可以看到， $V(s)$ 是单调不减的，因此，其对应的策略也应该是单调提升的。

从第5行-第7行可以看出，在每一轮迭代中，都需要对于状态空间中所有的状态访问一次。当状态空间庞大的时候，这样的循环会十分消耗时间。可以采用异步DP的算法，每轮迭代仅更新一部分的状态，只要保证在多轮迭代中，每个状态都能被访问到，这样的异步DP算法一样能收敛到最优策略上。

DP算法对于每一个状态价值函数的估算都依赖于前一时刻各状态价值函数的数值，这种特性我们称之为***bootstrap***。这个单词本意是拔靴带，一个比较好的翻译是“自举”，参看下面这张图片来更好的理解这个词的含义。同时我们注意到，在DP算法中，我们认为反映环境的所有信息 $p(s', r|s, a)$ 是已知的，即我们已经拥有了对于环境的建模，这种利用反映环境信息的模型来进行计算的特性我们称之为***model-based***。从某种意义上来说，DP方法本质上还并没有涉及到对于环境的“学习”过程，因为DP没有通过与环境的交互来获取关于环境的信息。

图：Bootstrap

蒙特卡洛方法

如果说动态规划类方法主要从迭代的角度出发，通过反复把Bellman算子作用到价值函数上期望收敛到最优情形下的价值函数；那么***蒙特卡洛方法***（Monte Carlo methods）则是直接从最优价值函数的定义出发，通过采样来直接对于最优价值函数进行无偏估计。

具体来说就是先进行采样，即随机地从一些状态出发，然后按照行动策略（behavior policy） μ 采取行动，直到达到终止状态；然后进行最有价值函数的估计，即回溯地利用探索到的这部分信息来更新目标策略（target policy） π 下的价值函数。当每一轮的行动策略与目标策略一致的时候，我们称这样的方法为on-policy的；反之，我们称之为off-policy的。

我们先来学习on-policy的情形。

算法五：Monte Carlo Method (on-policy)

1 initialise $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
2 $Q(s,a) \leftarrow$ arbitrary
3 $Returns(s,a) \leftarrow$ empty list
4 $\pi(a|s) \leftarrow$ arbitrary ϵ-soft policy
5 repeat forever:
6 generate an episode using π
7 for each pair s, a in the episode:
8 $G \leftarrow$ return following the first occurance of s, a
9 append G to $Returns(s,a)$
10 $Q(s,a) \leftarrow$ average($Returns(s,a)$)
11 for each s in the episode:
12 $A^* \leftarrow \arg\max_a Q(s,a)$
13 for all $a \in \mathcal{A}(s)$
14 $\pi(a|s) \leftarrow \begin{cases} 1-\epsilon+\epsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \epsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$

对于MC方法，只有满足如下两个条件的时候才能够保证算法收敛到最优策略

- 主循环被执行无穷多次
- 当循环次数趋向无穷的时候，每个状态被遍历的次数也趋向无穷

对于第一点，我们只能在资源允许的情况下尽可能多地运行主循环。对于第二点来说，我们的解决方法主要有两个思路；要么在每次主循环开头时，保证选择任意起始状态的概率都大于零，这种方法称之为exploring start；要么在算法运行过程中保证我们行动策略 μ 具有足够的随机性，即能始终以一个大于零的几率访问任意可能的行动。特别地， $\forall a \in \mathcal{S}$ ，如果某个策略 μ 能够保证对于每个行动的访问概率不低于 $\epsilon/|\mathcal{A}(s)|$ ，则称这样的策略是 ϵ -soft的。

对于算法五第14行中定义的策略，我们称之为 ϵ -greedy的，它是一种 ϵ -soft策略。算法五中第10行是对于当前策略对应价值函数的估计，第14行是基于当前价值函数对于策略进行改进，由于探索使用的策略（第6行）和目标输出的策略（第14行）是同样的策略，因此这种方法是on-policy的。

行动策略也可以和目标策略不相同，即得到下面的off-policy的蒙特卡洛算法。

算法六：Monte Carlo Method (off-policy)

1 initialise $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
2 $Q(s,a) \leftarrow$ arbitrary
3 $C(s,a) \leftarrow 0$
4 $\pi(s) \leftarrow$ a deterministic policy that is greedy w.r.t.Q
5 repeat forever:
6 generate an episode using any soft policy μ
7 $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T, S_T$
8 $G \leftarrow 0$
9 $W \leftarrow 1$
10 for $t = T-1, T-2, \cdots,$ downto 0:
11 $G \leftarrow \gamma G + R_{t+1}$
12 $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
13 $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}(G - Q(S_t, A_t))$
14 $\pi(S_t) \leftarrow \underset{a}{arg\,max}\, Q(S_t, a)$
15 if $A_t \neq \pi(S_t)$ then exit forloop
16 $W \leftarrow W / \mu(A_t|S_t)$

而算法三中，探索所使用的策略 μ 是任意固定的 ϵ-soft策略（第6行），比如它可以是在行动集中等概率选取行动的随机策略。而目标输出的策略 π 是一个关于价值函数的贪心策略，因此这种方法是off-policy的。由于行动策略与目标策略不一致，因此在更新价值函数的时候我们需要把行动策略对应的收益投影到目标函数对应的收益上。直观地来说，如果行动策略选择了某个当前目标策略大概率会选择的路径，那么对应的价值函数的改变会更多一些；反之，应该尽可能少一些地改变价值函数。这里我们使用了重要性采样（importance sampling）的技术。即目标策略对应的价值函数 $V(s)$ 与行动策略得到的收益 G_t 应该满足如下关系。

$$V(s) = \frac{\sum_t \rho_t^T G_t}{\sum_t \rho_t^T}$$

其中重要性采样率（importance-sampling ratio）

$$\rho_t^T = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$$

其中， T 表示终止状态的时刻。算法三中的第12、13、16行所描述的就是这样的重要性采样的增量实现。

总体来说，蒙特卡洛方法能够通过与环境的交互来学习一个好的策略，它不需要对于环境的建模，即它是***model-free***的。不同于之前的动态规划算法，每一步中价值函数的更新并不依赖于之前的价值函数，而是直接更新到采样得到收益的平均值上，因此它不是bootstrap的。由于蒙特卡洛方法要求一直采样到最末尾，因此它只能用于回合制的任务。

总结

这一讲我们学习了强化学习的一些基本概念，包括

- 强化学习的基础模型是马可夫决策模型
- 强化学习的目标是找到最大化收益的策略
- 找寻策略的一个重要途径是找到该马可夫决策模型上的价值函数
  - 直接使用采样的方法估计这个被定义出来的价值函数形成了蒙特卡洛方法
  - 使用Bellman算子迭代计算这个价值函数形成了动态规划方法
  - 在此两者的基础上还会发展出我们后面会讲到的***时间差分方法***（temporal-difference，TD）

强化学习的不同算法有不同的维度，我们可以总结出来下面这张表，在之后的学习之中，我们可以加入更多的行（算法）和列（维度）

Method	Boostrap	On/Off-policy	Model-based/free
DP	Yes	--	Model-based
MC	No	Both available	Model-free

参考资料

1. Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998. 一本经典的强化学习入门教材

2. Puterman, Martin L. Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, 2014. 一本讲马可夫决策过程的书，涉及更为坚实的数学基础
3. Berkeley CS294 课件可以在 这里 找到
4. 我导师开设的 高等理论计算机课程 的最后两讲

编辑于 2019-03-31

强化学习 (Reinforcement Learning)