强化学习经典入门书的读书笔记系列--第五篇（下）

原创 2017-06-15 小吕 神经网络与强化学习

原文参考：《Reinforcement Learning: An Introduction 》Second Edition Draft
公众号内回复“**强化学习**”，可以下载本系列文章的配套书籍。
文章原创，欢迎转发，禁止转载！

接之前第五篇（上）

5.5 Evaluating One Policy While Following Another (Off-policy Policy Evaluation)

这一节的问题很有意思。之前我们在estimate v_π或者q_π时，需要根据目标policy--v_π，生成一系列episodes，然后再进行policy的evaluation和improvement。现在问题来了，假如我们仍然想estimate v_π或者q_π，然而给定的episodes缺不是依据v_π或者q_π生成的，而是依据一个不同的policy $\mu \neq \pi$,怎么办呢？这里先给出几个名词：π叫做目标策略（target policy），μ叫做行为策略（behavior policy），而从behavior policy生成的episodes中学习target policy的过程，叫做off-policy learning。

（我想提醒各位读者的是，这里引入的off-policy的思想非常重要，甚至在后面几章都还有提及，是重头戏，希望大家可以重视。同时，既然很重要，其难度也不小，所以也要细细品味其中的内涵）

首先，我们要问个问题：好端端的，我们已经在前几节基本解决了使用蒙特卡罗算法来estimate q_π 并获得optimal policy，为什么这里还要来谈off-policy的问题。这就要从上一节的蛛丝马迹中发现端倪。在上一节末尾，我们已经可以从$\varepsilon - soft$的一类policies中找到最优的policy $\varepsilon - greedy$了，但是请注意限定词“$\varepsilon - soft$”。其实，我们最终得到的policy，是在妥协了“exploring start”这个假设下的最优policy。为什么？因为我们必须需要这个假设，没有这个假设，一切蒙特卡洛算法的根基就无从谈起，这些内容读者们可以看上一篇，有详细的介绍。

于是，为了满足这个“exploring start”假设，on-policy方法就必须要一定概率地去选择一些次优actions。这也就提出了一个问题：能不能不去妥协“exploring start”假设，还是用之前的greedy策略，每次就选最优的action呢？这就是为什么要讨论off-policy的原因。

在off-policy方法中，我们有两种policy：一种用于产生episodes，是之前说的behavior policy，姑且叫做μ；另一种，就是target policy，它是依据episodes得到的value function，得到的greedy policy。behavior policy是已知的，target policy是未知的。为了能够从μ生成的episodes中估计π,μ必然是随机的，也就是$\mu(a|s) > 0$，因为必须确保π中的所有actions在μ中都出现。π却可以是确定性的，因为我们想得到完全的最优解。也就是说，把exploration的任务交给μ，让它彻底地exploration，而从它产生的q_π中，通过某种方式，使得π可以保持彻底的greedy。

（高能预警，下面的这部分自我感觉比较难懂，但是尽量说明白。）

首先我们给出两个概率表达式，如下：

$$p_i(S_t) = \prod_{k=t}^{T_i(S_t)-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)$$
$$p_i^{'}(s_t) = \prod_{k=t}^{T_i(s_t)-1} \mu(a_k|s_k)p(s_{k+1}|s_k, a_k)$$

逐一分析：i表示第i个episode序列；大写的S和小写的s为了区分在策略π和策略μ下的序列；这两个式子分别表示在策略π和策略μ下，第一次访问到状态s之后的两个相同序列的发生概率，注意是相同序列。假设把从state$s$开始观测到的return记作$G_i(s)$,也就是说这个$G_i(s)$是在behavior policy下观测到的，如果我们想要根据这个return来estimate$v_\pi(s)$,就需要满足下列式子：

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i^{'}(s)} G_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i^{'}(s)}}$$

这里需要特比说明的有三点：第一，为什么求和？这个好解释，因为蒙特卡洛方法就是要作多次returns之后的平均；第二，为什么要求$\frac{p_i(s)}{p_i^{'}(s)}$？这个原因要回顾一下上面的两个概率表达式。我们这么来想：假设有两个事件A和B，已知A发生的概率为$p(A)$,B发生的概率为$p(B)$,现在又已知A的某个相关变量$f(A)$,那么怎么求B的那个相关变量$f(B)$呢？其实这里A的相关变量就可以看成是$G_i(s)$，B的那个相关变量就是要求的$V(s)$，两者概率的相对比例和两个相关变量的相对比例是相同的。第三点，为什么要除以$\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i^{'}(s)}$，这个也好解释，为了归一化。

好了，还有一個问题，怎么得到$\frac{p_i(s)}{p_i^{'}(s)}$？这个相对好解决：

$$\frac{p_i(S_t)}{p_i'(S_t)} = \frac{\prod_{k=t}^{T_i(S_t)-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T_i(S_t)-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T_i(S_t)-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}.$$

要注意这里为什么最后可以约掉$p(S_{k+1}|S_k, A_k)$,是因为前面强调的相同序列。只有在相同序列中，两种策略下的S_k和A_k才均相同，这样$p(S_{k+1}|S_k, A_k)$才相同。从另一个角度看，如果不直接求两者的相对比例，而是去分别求出$p_i(S_t)$和$p_i^{'}(S_t)$,再试图相除，就做不到。因为蒙特卡洛算法面对的问题往往是不知道任何环境动态性质的，也正是因为我们没有这么做，才恰好利用了相同序列$p(S_{k+1}|S_k, A_k)$可以消掉的性质，解决了求$\frac{p_i(s)}{p_i^{'}(s)}$这个问题。

5.6 Off-Policy Monte Carlo Control

有了上一节的铺垫，这一节我们就可以看一下真正的off-policy 蒙特卡洛算法了。重新回顾一下Off-Policy的思想：使用$\varepsilon - soft$类型的behavior policy来生成episodes，这些episodes经过了充分的exploration；从这些episodes中按照蒙特卡洛的“every visit”或者“first visit”方法计算$G_i(s)$，然后根据

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i^{'}(s)} G_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i^{'}(s)}}$$

来计算target policy的$v_\pi(s)$。off-policy的优点是分离了exploration和greedy action ‘s choice这两个过程，而之前在on-policy算法中，它们必须相互妥协，使得最终效果不是很好。

程序应该怎么写呢？以下给出了伪代码：

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$N(s,a) \leftarrow 0$; Numerator and
$D(s,a) \leftarrow 0$; Denominator of $Q(s,a)$
$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:
(a) Select a policy μ and use it to generate an episode:
$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
(b) $\tau \leftarrow$ latest time at which $A_\tau \neq \pi(S_\tau)$
(c) For each pair s, a appearing in the episode at time τ or later:
$t \leftarrow$ the time of first occurrence of s, a such that $t \geq \tau$
$W \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\mu(A_k|S_k)}$
$N(s,a) \leftarrow N(s,a) + WG_t$
$D(s,a) \leftarrow D(s,a) + W$
$Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$
(d) For each $s \in \mathcal{S}$:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

解释一下伪代码中的几个重点：首先是小括号（b）那一句，latest time的含义，指的是，在这个时间点τ之后，所有的$A_\tau = \pi(S_\tau)$,也就是说只有这个时间点之后的序列，才能被用于计算target policy的$Q(s, a)$,为什么？请读者结合前面的内容思考。然后是（c）那部分，为什么$W \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\mu(A_k|S_k)}$？因为在时间点$\tau$之后的序列，两种策略的动作选择都是相同的，也就是说$\pi(A_k|S_k) = 1$,因此分子是1。

这种方法是完美的吗？并不尽然，一个潜在的问题就是每个episode可利用的部分只有尾部，也就是只有在时间点τ之后的序列，即最后一个non-greedy action之后，才可以被用来更新$Q(s, a)$。那么如果non-greedy actions特别多，会使得学习过程很缓慢。这个问题的严重性如何评估，需要进一步商榷。

5.7 Incremental Implementation

这一节主要讲的就是一个小的trick，回想一下第二节中讨论的多臂赌博机问题，那里提到了把return写成递增的形式。这一节也是讨论如何通过某种方式，把off-policy方法中的下式：

$$V(s)=\frac{\sum_{i=1}^{n_s}\frac{p_i(s)}{p_i'(s)}G_i(s)}{\sum_{i=1}^{n_s}\frac{p_i(s)}{p_i'(s)}}$$

写成递增的形式。这样写有个好处就是可以省内存和计算时间，因为随着episodes数量的增加，如果按照原写法计算，会不断增加内存的负担，也会不断增加计算量。而写成递增的形式，则无论episodes数量如何增大，我们始终只需要保存两个变量：原$V(s)$和增量，就可以不断更新return了。

以下是数学推导，有兴趣的可以一看：

> 令$W_k=\frac{p_k(s)}{p_k'(s)}$,
>
> 那么上式的$V(s)$可以写成以下形式：
>
> $$V_n=\frac{\sum_{k=1}^{n-1}W_kG_k}{\sum_{k=1}^{n-1}W_k}$$
> 于是
> $$V_{n+1}=\frac{\sum_{k=1}^{n}W_kG_k}{\sum_{k=1}^{n}W_k}$$
> 令
> $$\sum_{k=1}^{n}W_k=C_n$$
> 那么
> $$V_{n+1}=\frac{C_{n-1}V_n+W_nG_n}{C_n}$$
> $$=\frac{(C_n-W_n)V_n+W_nG_n}{C_n}$$
> 最终可得：
> $$V_{n+1}=V_n+\frac{W_n}{C_n}[G_n-V_n]$$
> $$C_{n+1}=C_n+W_{n+1}$$
> $$C_0=0$$

5.8 Summary

按惯例，需要总结一下。蒙特卡洛方法的主要思想就是从过往的实际经验中学习策略。这些过往经验以若干独立的样本片段给出。这种独特的思想使蒙特卡洛方法拥有DP所不具备的三大优势：

(1)不需要环境的动态model。实际环境的model有时很复杂，很难准确建立。

(2)可以通过模拟样本片段来使用。很多情况下，精确的转移概率模型很难确立，但是往往很容易模拟样本的走势。

(3)可以有选择地专注于真正重要的一部分状态集合。

还有一個优势在后几章将会讲到，当环境的马尔科夫性质不能被严格满足时，使用蒙特卡罗方法比使用DP受到的影响小（这一点我暂时很难给出确切的说明，在后几章或许可以说的更明白一些）。这是因为蒙特卡洛方法不进行bootstrap，而DP需要bootstrap。

这里要说明一下bootstrap的含义：在统计学中，bootstrap常常表示在有些的样本中进行多次有放回抽样，以获得足够估计总体的样本数量。在这里指的是有没有通过估计的方法来引导计算，也就是自举。DP算法中，每次value的更新都需要之前的value估计值，而蒙特卡洛算法中就不需要这些。

蒙特卡洛算法的基本模式依然是GPI,先policy evaluation，然后policy improvement。其核心思想则提供了另一种policy evaluation的方法：不用model来递归计算所有可能性，而是对真实样本片段经验过程的returns平均化。我们通常估计$q_\pi(s,a)$而不再是$v_\pi(s)$,因为前者可以保证在没有model的情况下也能进行policy improvement。另外，蒙特卡洛算法也可以写成增量形式（前一节内容）。

蒙特卡洛算法中，保证足够的exploration是最需要注意的关键点。因为如果只是选择当前对应$q_\pi(s,a)$的最优action，那么将会导致很多其他的actions永远没机会选到。一个解决方案是“exploring start”，这个方案在模拟产生episodes也许可行，但是在从真实经验中学习时就不可行了，因为我们无法控制start point。于是分化出了两种学习方式：on-policy和off-policy。on-policy方法一边exploration一边在此基础上选取最优解；off-policy则彻底把exploration和policy improvement过程彻底分开：更彻底地exploration，更彻底地选择greedy actions，最后得到的optimal policy甚至和behavior policy没有任何关系。

最后说一下蒙特卡洛算法和DP算法的不同：第一点：前者可以从样本经验学习，也就是说，它不需要任何准备模型的过程，可以直接在真实情形下学习，而后者必须有一个精确的模型；第二点：前者不需要bootstrap，也就是不基于其他value estimates来更新某个value，后者则是递归

更新。

在下一章，我们会讨论这两种算法的某种意义上的结合体：既从经验中学习，又进行bootstrap过程。

敬请期待！

欢迎大家提出问题或者意见，作者会在后台给大家回复。

公众号：神经网络与强化学习