强化学习经典入门书的读书笔记--第二篇（上）

2017-04-25 小吕 神经网络与强化学习

正文

区分强化学习和其他种类的学习方式最显著的特点是：在强化学习中，训练信息被用于评估动作的好坏，而不是用于指导到底该是什么动作。这也是为何需要主动去做exploration的原因。纯粹的评估性反馈可以表明一个动作的好坏、但并不能知道当前动作是否是最佳选择或者是最差选择。评估性反馈（包括evoluationary method）是方程优化的基础。相对的，纯粹的指导性反馈，表明了当前的最优动作，这个最优动作是独立于实际采取的动作的。这种反馈形式是监督学习的基础，被用于模式识别、人工神经网络等方面。当然了，也有一些交叉案例中，这两种反馈形式同时使用。

这一章主要在一种简单的情境下研究强化学习的这种反馈形式，这种简单情形只包含一个环境状态，这样能避免完整的强化学习问题的很多复杂方面，从而专注于研究evaluative feedback 和instuctive feedback的不同，以及如何对两者进行结合使用。

这个简单的案例叫做多臂赌博机问题。我们用这个案例来学习一些强化学习中的基本学习方法，在这章末尾，我们会讨论一下当环境状态不止一个时，多臂赌博机的学习问题会是什么样子。

2.1 n-armed bandit problem:

这里稍微描述一下多臂赌博机这个概念。赌博机，也可以想象成老虎机，现在我有n台老虎机，每台有一个臂（也就是杠杆）我可以通过拉动这个臂来获得这台老虎机的reward，但是每台赌博机的这个reward是按照一定的概率分布的，因此有上下浮动，不是固定不变的。现在呢，我需要解决的问题就是最大限度地通过杠杆拉动序列，使得获得的奖励最大。这个序列的次数是任意的。

对于每台老虎机的臂，都有一个期望reward，也就是我们之前说的value。当然这个value的数值之前肯定是不知道的，不然这个问题就没有意义了。然而我们可以大概估计这个value值。

如果我们保持我们的这种value估计，那么在每次选择动作之前肯定至少有一个臂的value是最大的。如果我们每次都选这个最大的value值的臂，那么这就是贪婪动作。如果我们每次都采取贪婪选择，那我们就是在exploiting。如果我们每次不这么干，而是有时去选一些非贪婪选择的臂，那么我们就是在exploring（探索），因为这样我们就能改进对其他臂value信息的估计。exploitation当然是正确的，因为它是当前这一步的最优选择，然而exploration也许会帮助我们在长远的序列中得到更大的total reward。比如：假设每次做选择的时候，贪婪选项的value是可以确定的，但是非贪婪选项的value值带有一定的不确定性。这种不确定性的意思是，有可能个别的非贪婪选项的value会好于贪婪选项的value，但是不确定是哪个。这样的情况下，适当的exploration有助于帮我们找到可能存在的比贪婪选项更好的那个选择。可能在短期内reward比较低，然而一旦找到了，我们就可以反复选择（exploit）那个之前被认为非贪婪的选项，从而使得长期reward总和较高。

在特定的情形下，是exploiting还是exploring取决于很多因素。比如value估计值的精确程度、非贪婪值的不确定性或者是剩余的选择机会等等。有很多复杂的方法来平衡这两者的选择，然而大多数这样的方法都有很强的假设前提或者先验知识，而这些前提条件在很多的实际强化学习问题中是不能被保证的。当这些假设前提不被保证时，这些方法的效果也就不那么出色了。

在这一章，我们不必去用复杂的方法把exploiting和exploring之间平衡的那么好，而只是单纯的去平衡就好了。我们会用几种简单的方法去实现两者间的平衡，并表明平衡之后的学习效果要好于一味的exploiting。

2.2 action-value method:

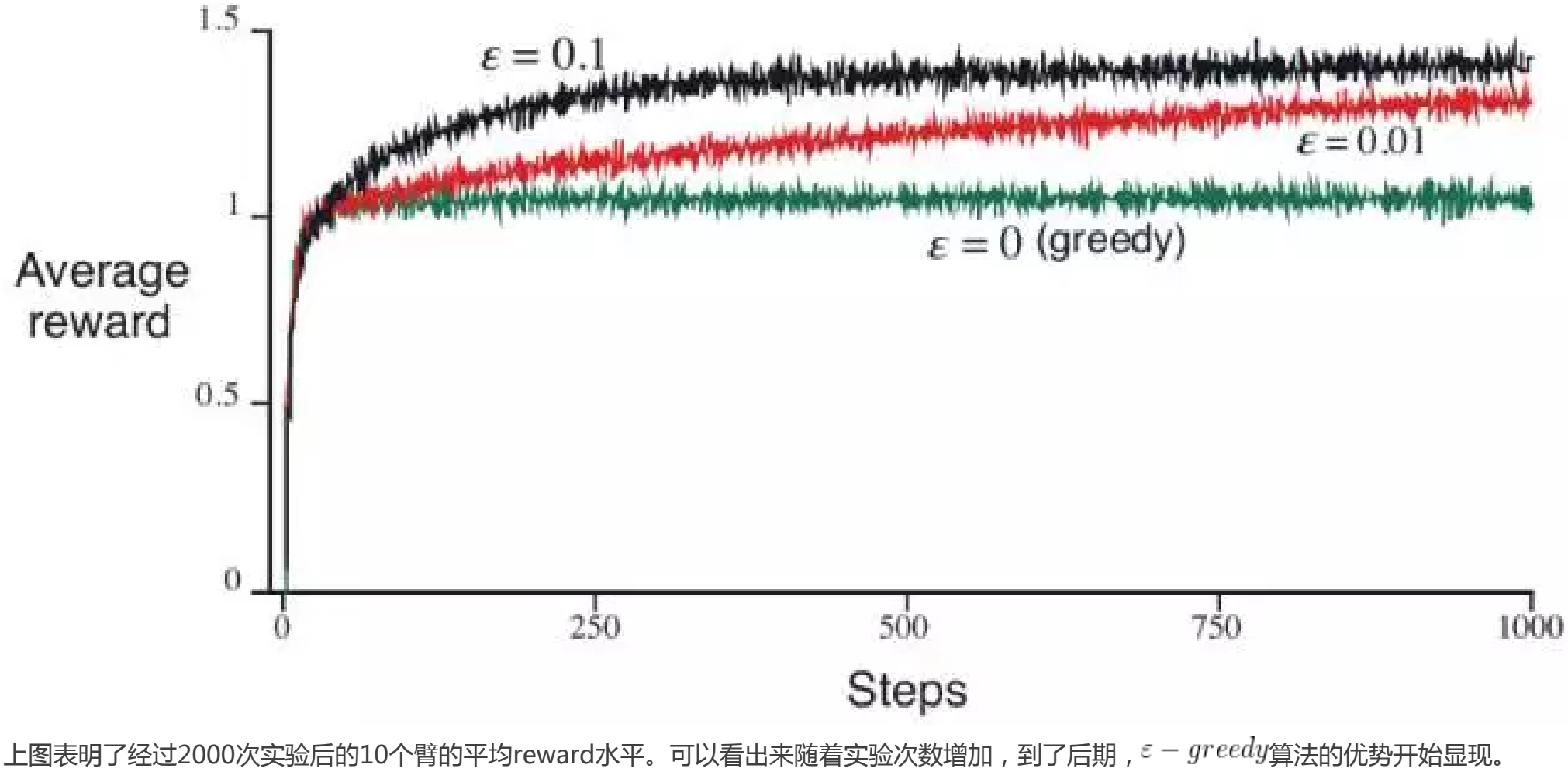
首先，我们先来仔细的做一下value的估计。我们把每个action的真实value定义为$q_*(a)$，把每个action在第t个时间步下的估计值定义为$Q_t(a)$。之前我们提到过，每个action的真实value，是当该动作被选择时所获得的期望reward，在这里，我们自然想到用最简洁的历史平均reward来表示当前动作的value估计值。假定某个action在t时间前总共被选择了K_a次，于是我们的估计值如下式：

$$Q_t(a) = \frac{R_1+R_2+...+R_{Ka}}{t}$$

这个方法叫做sample-average，因为每个动作的estimated value都是依据过往sample reward的平均值进行计算的。当K_a等于0,我们可以把定义为一个默认的初始值，当K_a趋近无穷，则最终收敛于$q_*(a)$。当然这个estimate的方法不一定是最好的，但是不要紧。

最简单的选择策略就是每次选择action value最大的那个。也就是使得$Q_t(A_t^*) = max_a Q_t(a)$，这样的策略只是exploiting，不exploring。另一个简单的方法就是大部分时间用来exploiting，然而每隔一段时间，比如说以概率为ε的可能性，来exploring。因为这种方法和贪婪方法很像，就叫做$\varepsilon - greedy$方法。这种做法的好处是随着选择次数的增加，对每个动作的选择次数都会趋向无穷，也就保证了每个动作的estimate value都无限趋近真实值。

为了粗略的评估greedy算法和$\varepsilon - greedy$算法的优劣，我们做了2000次实验，赌博机的个数,每个赌博机的臂拉下的真实value服从高斯分布。在t时间的每个臂被选择时的reward用真实值外加服从高斯标准正太分布（mean = 0 variance = 1）的噪声之和来表示。



上图表明了经过2000次实验后的10个臂的平均reward水平。可以看出来随着实验次数增加，到了后期，$\varepsilon - greedy$算法的优势开始显现。
当然，$\varepsilon - greedy$算法相对于greedy算法的优势也并非始终。当噪声方差很大时，$\varepsilon - greedy$算法可以通过较多的exploration来找到最优动作，这种情况下其相对于greedy算法的优势很明显。而当噪声方差为0时，这种优势就不尽然了。然而，就算是在deterministic的任务下，exploration想比于none-exploration也有很多优势。比如假设赌博臂是nonstationary的，也就是它的真实value是不确定的（刚才我们讨论的真实value是确定的，而在实际情况下，不确定的value更普遍）。
总之，这一节通过简单的对比实验告诉我们，exploiting和exploration之间的平衡是必须的。

2.3 soft-max action selection

提出softmax action selection的主要目的是因为上一节的$\varepsilon - greedy$算法针对剩下的所有选择都采取同样的概率，这样会导致最坏的选择也会同等机会被选中，这不是我们想要的。一个明显的解决办法是让每个赌博机臂被选中的概率和其estimate value相关联。对于在t时间选中动作a的概率，由下式给出：

$$\frac{e^{Q_t(a)/\gamma}}{\sum_{i=1}^{n} e^{Q_t(i)/\gamma}}$$

γ越大，则概率分布的越均衡，γ越小，那么不同选项被选中的概率差异越大。

softmax-greedy算法和$\varepsilon - greedy$算法都只有一个参数需要人为选择。但是很多人觉得$\varepsilon - greedy$的参数ε很好设置，但是γ却不好准确设置，因为有了指数的缘故。因此，近些年来，softmax算法逐渐不太流行，一方面因为参数难以合理设置，另一方面也是因为在不知道实际value的情况下，更不能知道其中的概率和estimate value之间的关系。但是，当我们在后文中涉及policy-based方法时会再回来看这个softmax-selection的。

2.4 Incremental Implementation

这一节主要改进的是计算和内存复杂度。

回顾一下前文中的value estimation公式

$$Q_t(a) = \frac{R_1 + R_2 + ... + R_{Ka}}{t}$$

可以发现，为了算在t时间的action a的value estimation，我们需要t时间之前所有时间该动作的reward记录。随着时间增加，这种计算方式涉及的计算复杂度和存储压力都会增加。

其实这是不必要的，我们完全可以用另一种方式来计算$Q_t(a)$。推导如下：

$$\begin{aligned}
Q(k+1) &= \frac{1}{k}\sum_{i=1}^{k} R_i \\
&= \frac{1}{k}(R_k + \sum_{i=1}^{k-1} R_i) \\
&= \frac{1}{k}(R_k + (k-1)Q_k + Q_k - Q_k) \\
&= \frac{1}{k}(R_k + kQ_k - Q_k) \\
&= Q_k + \frac{1}{k}[R_k - Q_k]
\end{aligned}$$

用这个形式的迭代公式，只需要保存Q_k和k，再加上一点简单的加减运算，就可以快速得到Q_{k+1}。

这个形式的写法贯穿于本书，需要加以注意：

$$newEstimate <- oldEstimate + stepSize(target - oldEstimate)$$

target-oldEstimate是estimate的error，乘以stepsize是不断地减小这个error，逼近target。在这个例子中，target就是第k次选择某动作的的reward。实际上，stepsize也是随着时间而变化的，在本书的其他地方，常用α来表征stepsize，或者更精确的$\alpha_t(a)$。在这里$\alpha = \frac{1}{k}$。

这次上篇先写到这里，未完待续。