MBIE and MBIE-EB

这篇论文的作者是littlman，做的一遍关于其在MDPs方面的exploration上的论文。

首先明确一个观点，如果一个学习算法是满足PAC理论的，那么这个学习算法则可以在有限的样本中达到学习的目的，**强学习的(strong learnable)**。

对于一个确定的MDP模型来说，提出模型探索模型的人来说，首先需要分析的就是这个算法的采样复杂度，前提是证明其采样复杂度是有限的，然后证明其采样复杂度的上限是关于$\mathbb{O}(\frac{1}{\delta}, \frac{1}{\epsilon}, \frac{1}{1-\gamma}, |S|, |A|)$其中几个参数的多项式，就能保证该算法能够达到收敛的效果。这个地方可以直观的理解为需要采样的样本数量需要达到$\mathbb{O}(\frac{1}{\delta}, \frac{1}{\epsilon}, \frac{1}{1-\gamma}, |S|, |A|)$，多的样本数量就行。只需要探索无数次，则在理论上是可以保证能获得最优解的。

文章中的算法和分析

作者实际上提出两种更新MBIE算法，并证明其是满足PAC-MDP的

第一种是比较复杂的MBIE，另一种则是比较简单的MBIE或者叫做MBIE with exploration Bonus 版本，作者都证明了他们是满足PAC-MDP的。

MBIE

要明白这个算法需要搞明白以下几个算法。

关于奖励估计

明确一点。在强化学习中，agent是对MDP环境是无知的，在没有任何先验的情况下，agent的仅仅只会在做动作以后立即接受到来自环境的反馈，从而更新其$Q-value$。若采用无模型更新的。在各种状态下做出的动作的奖励应该采用如下公式更新：

$\hat{R}(s,a) := \frac{1}{n(s,a)} \sum_{i=1}^{n(s,a)} r[i]$

其中$n(s,a)$是统计agent在状态s，采取了a动作的次数。$r[i]$表示第i次所获得的奖励。

作者换了一条思路（应该有前人做过，但是我在这就把这个思路当成是作者独有的）：将$r[i]$视为是从分布$R(s,a)$采样得到的，那么可以认为真正的$R(s,a)$应该是在区间$CI(R)$中。其中$CI(R)$定义为：

$CI(R) := (\hat{R}(s,a) - \epsilon^R_{n(s,a)}, \hat{R}(s,a) + \epsilon^R_{n(s,a)})$

$\epsilon^R_{n(s,a)} := \sqrt{\frac{ln(2/\delta_R)}{2n(s,a)}}$

这个置信区间是通过Hoeffding不等式计算出来的。同时也能得出上面这个式子是满足PAC框架的。

关于转移概率的估计

先给出众所周知的预估概率的方法。假设$\hat{T}(s'|s,a) := \frac{n(s,a,s)}{n(s,a)}$，显然$\sum_{s'|s,a} T(\cdot|s,a) = 1$就可以当做状态转移概率了。同样，依据上节的思路，可以认为这个真实值处于估计值表示的某一个区间中，论文是直接给出了

$CI(T) := \{\tilde{T}(s,a) \in P_s |\|\tilde{T}(s,a) - \hat{T}(s,a)\|_1 \leq \epsilon^T_{n(s,a)}\}$

$\epsilon^T_{n(s,a)} = \sqrt{\frac{2[ln(2^{|S|}-2) - ln(\delta_T)]}{m}}$

同样可以证明上述的估计是满足PAC的

更新公式

作者给出了使用上述两个估计量的更新公式:

$Q'(s,a) := \max_{\tilde{R}(s,a) \in CI(R)} \tilde{R}(s,a) + \max_{\tilde{T}(s,a) \in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s,a) \max_{a'} Q(s',a')$

几点说明:

1.该算法可以才每个常数时间更新。

2.其中的m就表示这个常数时间。

3.计算第二个式子需要一定的算法。littman给出了算法和证明，写到伪代码呢里。

正文并没有给出算法的代码，让人觉得很费解，我会在稍后的内容中给出算法的伪代码。然后给Q出算法的时间复杂度分析

第二项公式说明

令

$$M_v = \sum_{s'} \tilde{T}(s'|s,a)V(s')$$

这个式子为一个优化问题

$$\max_{(s,a)} M_v$$

$$subject\ to$$

$$\|\tilde{T}(\cdot|s,a) - \hat{T}(\cdot|s,a)\| \leq \epsilon$$

解决优化问题。

论文中给出了一个方法，这里我先做文字说明，然后给出伪代码，再给出Python代码。

在某一个循环中:

1. 修改R(s,a)的值 by $R(s,a) \leftarrow R(s,a) + \epsilon^R_{n(s,a)}$
2. 排序$Q_{max}[t+1]$
3. 并找到其中最大值对应的状态设为s^*
4. $T(s^*|s,a) \leftarrow T(s^*|s,a) + \frac{\epsilon^T_{n(s,a)}}{2}$
5. if $T(s^*|s,a) \geq 1$

   $T(s^*|s,a) \leftarrow 1$

   其余$T(s'|s,a) \leftarrow 0\ if\ s' \neq s^*$

6. while $\sum_{s'} T(s'|s,a) > 1$:

   6.1找到最小的非0$Q_{max}[t+1]$所对应的状态$s_$

   6.2更新 $T(s\_|s,a) \leftarrow max(0, T(s\_|s,a) + (1 - \sum_{s'} T(s'|s,a)))$

7. 更新Q(s,a) by $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} T(s'|s,a) Q_{max}[t+1](s')$

8. 修正整个Q表，以及当前阶段的$Q_{max}[t+1]$

分析：在采样阶段

根据递归法则更新的算法

$T(|S|) = T(|S|-1) + |S|$可以推出$T(|S|) = |S|ln|S|$

即在更新Q值时算法复杂度为$O(|S|ln|S|)$

对于上述式子的value iteration 的时间复杂度分析：

首先，我们更新算法时，需要便利所有的(s,a)，故总的时间复杂度应为$|S||A| * O(\text{一次迭代的时间复杂度})$，一次循环的时间复杂度可从伪代码计算：

步骤2的时间复杂度为$O(|S|ln|S|)$ 快速排序和归并都可以做到

步骤6的时间复杂度为$O(|S|)$

步骤8采用二分查找法修改当前(s,a)所对应的Q值时间复杂度为$O(ln(|S||A|))$,以及获得当前timestep 各个状态所对应的最大Q值，其时间复杂度为$O(ln(|A|))$

故得到时间复杂度为$O(|S|ln|S| + ln(|S||A|) + |S| + ln(|A|)))$，根据时间复杂度的性质可得最终时间复杂度为

$$O\Big(|S||A|N\big(S|ln|S| + ln(|S||A|)\big)\Big)$$

python 代码如下

```python
def compute_qVals_MBIEVI(self, R, P, R_confident, P_confident):
        '''
        通过MBIE计算Q值表
        Args:
            R - R[s,a] :  奖励平均值 数据类型是浮点型
            P - P[s,a] :  状态转移频率 数据类型是 |S|维向量
            R_confident - R_confident[s,a] = R的置信度
            P_confident - P_confident[s,a] = P的置信度

        Returns:
            qVals - qVals[state, timestep] 是timestep的Q值
            qMax - qMax[timestep] 是当前timestep的最大Q值
        '''

    qVals = {}
    qMax = {}
    qMax[self.epLen] = np.zeros(self.nState)
    for i in range(self.epLen):
        j = self.epLen - i - 1
        qMax[j] = np.zeros(self.nState)
        for s in range(self.nState):
            qVals[s, j] = np.zeros(self.nAction)
            for a in range(self.nAction):
                rOpt = R[s, a] + R_confident[s, a]
                pInd = np.argsort(qMax[j + 1]) #排序对应步骤2
                pOpt = P[s, a]
                #求最大值
                if pOpt[np.where[pInd==0]] + P_confident[s, a] * 0.5 > 1:
                    pOpt = np.zeros(self.nState)
                    pOpt[np.where[pInd==(self.nState-1)]] = 1
                else:
                    pOpt[np.where[pInd==(self.nState-1)]] += P_confident[s, a] * 0.5
                    #步骤6
```

```
                    while np.sum(pOpt) > 1:
                        worst = pInd[np.where(pInd==sLoop)]
                        pOpt[worst] = max(0, 1 - np.sum(pOpt) + pOpt[worst])
                        sLoop += 1
                    #步骤7&8
                    qVals[s, j][a] = rOpt + np.dot(pOpt, qMax[j + 1])
                #步骤8
                qMax[j][s] = np.max(qVals[s, j])
        return qVals, qMax
```

MBIE-EB

该论文的作者在研究PAC-MDP[1]中，则认为Q值可以做如下估计方式：

$$\hat{Q}(s,a) = \max_a R(s,a) + \gamma \sum_{s'} T(s'|s,a) max_a Q(s',a') + \frac{\beta}{\sqrt{n(s,a)}}$$

然后采取greedy策略来选取下一个动作，即：

$$a_{t+1} = \arg\max_{a \in \mathbb{A}} \hat{Q}(s,a)$$

这个时间复杂度就十分简单了，不做分析

采样复杂度以及完整性分析

未完待续。。。。

1. PAC-MDP是这其算法的采样复杂度是关于$(|S|, |A|, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma})$的多项式小于某个数的该概率之多为$1-\delta$↩