[强化学习]蒙特卡洛方法

作者：莫比

引言

蒙特卡洛方法并非是一个特定的算法，而是一类随机算法的统称，其基于思想是：用事件发生的“频率”来替代事件发生的“概率”。在机器学习中，这种方法可以用于模型是未知的情况中，它只需要从经验中去学习，这个经验包括样本序列的状态、动作和奖励。得到若干经验后，通过平均所有样本的回报来解决强化学习的任务。

1.策略评估

我们在给定策略的情况下，可以用蒙特卡洛方法来学习状态价值函数，也就是这个状态开始的期望回报——期望的累积未来折扣奖励，公式如下：

$v_\pi(s) = E_\pi[G_t|S_t = s]$

但是蒙特卡洛方法在策略评估时不是求的回报的期望，而是使用经验平均回报。随着我们的样本越来越多，这个平均值是会收敛于期望的。

也就是说，我们想要估计 $v_\pi(s)$的值，即遵循策略 π的情况下，状态 s 的价值，我们可以计算所有回合中首次访问状态 s 的平均回报， 以此作为 $v_\pi(s)$ 的估计值，这种方法就是首次访问MC方法； 与之对应的，另一种方法计算所有回合中每次访问状态 s 的平均回报，就是每次访问MC方法。

因为在模型未知的情况下，状态值函数不能够直接决定策略，我们要通过状态动作值函数$q_\pi(s,a)$即从状态s 开始，并采取动作 a,然后遵循策略π 得到的回报的期望去决定。我们这两种评估方法去估计$q_\pi(s,a)$就是求所有回合访问(s,a)所得到回报的均值。

首次访问MC算法流程：

输入：用来评估的策略 π

初始化：

  对所有的 $s \in S, a \in A(s)$，任意 $Q(s,a) \in R$

  $Returns(s) \leftarrow$ 一个空的列表，对所有 $s \in S$

一直循环（对每一个回合）：

  使用π生成一个回合：$S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  对于回合中的每一步循环，$t = T-1, T-2, \ldots, 0$：

    $G \leftarrow \gamma G + R_{t+1}$

    除非 S_t, A_t出现在 $S_0, A_0, S_1, A_0, \ldots, S_{t-1}, A_{t-1}$中：

      将G添加到列表$Returns(S_t, A_t)$中

      $Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$

2.策略控制

2.1问题探讨

在我们得到值函数之后，下一步就是进行提升，去近似最优值函数和最优策略。

图1.策略控制的过程

策略提升的方法是针对当前的价值函数，即使策略贪婪。我们只需要对每个$s \in S$选择使动作价值函数最大的那个动作：

$\pi(s) = argmaxQ(s, a)$

然后我们对于每个$\pi_{k+1}(s)$都取$Q_{\pi k}$的贪婪，这样我们就可以得到：

$Q_{\pi k}(s, \pi_{k+1}(s)) = Q_{\pi k}(s, argmaxQ_{\pi k}(s, a)) \geq Q_{\pi k}(s, \pi_k(s))$

也就是说每个$\pi_(k+1)$都比π_k更好，只要经过足够多的回合，就能收敛到最优的策略和动作价值函数。而想要这个过程收敛就必须满足下面两个假设：

（a）我们有无限个回合供策略评估使用。

（b）回合都是探索开端的方式。

2.2解决第一个假设

要经过无限个episode显然是不可能的，这里我们最好的方法就是去拟合$Q_{\pi k}$，拟合的主要方法有以下两种：

方法一：让每次策略评估都无限接近$Q_{\pi k}$。使用一些方法和一些假设，并且经过足够多的步骤后， 就可以保证一定程度的收敛

方法二：在跳转到策略提升前，放弃尝试完成策略评估。评估的每一步，我们将价值函数向$Q_{\pi k}$移动。一个极端的例子是价值迭代，就是每执行一步策略提升就要执行一步迭代策略评估。

2.3解决第二个假设

解决第二个假设具体来讲有两种方法，我们称之为在策略(on-policy)方法和离策略（off-policy）。on-policy方法尝试去估计和提升我们用作决策的那个策略相同；而off-policy估计和提升的策略与用来生成数据的策略不同。

2.3.1 on-policy策略

具体我们使用的是$\epsilon - greedy$策略，这种算法其实就是权衡开发与探索。即在大多数时间选择有最大估计动作价值的动作，但仍有ϵ的概率选择随机的动作。

on-policy首次访问MC控制算法流程：

初始化：

$\pi \leftarrow \epsilon - greedy$策略

对所有的 $s \in S, a \in A(s)$，任意 $Q(s,a) \in R$

对所有的$s \in S, a \in A(s)$， $Returns(s) \leftarrow$ 空列表

一直循环：

遵循策略 π ，生成一个回合：$S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

对于这个回合中的每一步，$t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

除非 S_t, A_t出现在 $S_0, A_0, S_1, A_0, \ldots, S_{t-1}, A_{t-1}$中:

将G添加到列表$Returns(S_t, A_t)$中

$Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$

$A^* \leftarrow argmaxQ(S_t, a)$

对所有$a \in A(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1-\epsilon+\epsilon/|A(S_t)| & if\ a = A^* \\ \epsilon/|A(S_t)| & if\ a \neq A^* \end{cases}$$

2.3.2 off-policy策略

off-policy在策略估计和策略提升的时候使用两种策略，一个是行为策略μ ：具有探索性的策略，专门用于产生episode积累经验；另一个则是目标策略 π：对行为策略μ产生的经验进行学习，使奖励最大化，成为最优策略 。

2.3.2.1 off-policy策略中的重要性采样

重要性采样就是去估计随机变量在一个分布上的期望值，但是采样的样本来自另一个分布。 离策略上应用重要性采样的方法，是为了根据在目标策略和行为策略下得到发生的事件轨迹的概率比，将得到的概率对回报进行加权。 两个概率的比值称为重要性采样率。给定初始状态S_t，那么在策略π下， 接下来的状态动作轨迹 $A_t, S_{t+1}, A_{t+1}, \cdots, S_t$ 发生的概率是

$$\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)$$

其中p代表状态转移概率函数。因此，在目标策略和行为策略下的重要性采样率为：

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$$

从上式可以看出重要性采样率最终仅仅依赖于两个策略和序列，而与状态转移概率无关。而我们的回报G_t都是从行为策略得到的，所以这些回报期望是错误的$E[G_t|S_t = s] = V_b(s)$,不能平均得到V_π.这个时候我们就可以通过重要性采样率获得正确的期望：

$$E[\rho_{t:T-1} G_t | S_t = s] = V_\pi(s)$$

接下来就是off-policy评估策略的公式：

（1）原始重要性采样

$$V(s)=\frac{\sum_{t\in\tau(s)}\rho_{t:T(t)-1}G_t}{|\tau(s)|}$$

（2）加权重要性采样

$$V(s)=\frac{\sum_{t\in\tau(s)}\rho_{t:T(t)-1}G_t}{\sum_{t\in\tau(s)}\rho_{t:T(t)-1}}$$

其中$\tau(s)$:代表所有状态 s 在某个回合中第一次被访问的时刻的集合

$T(t)$:从时刻t到$T(t)$的回报

$\{G_t\}_{t\in\tau(s)}$：属于状态s的回报

$\{\rho_{t:T-1}\}_{t\in\tau(s)}$：代表相应的重要性采样率

2.3.2.2增量式求均值

我们除了直接求均值的方式，还可以增量式求均值。假设我们得到了一系列回报$G_1,G_2\cdots,G_t-1$,都是从相同的状态开始的， 且每个回报值对应一个权值W_i (重要性采样率) ,除了跟踪 V_n， 我们还必须保持给定前 n个回报下每个状态的累积权值 C_n 。

V_n的更新规则为：

$$V_{n+1}=V_n+\frac{W_n}{C_n}(G_n-V_n)$$

$$C_{n+1}=C_n+W_{n+1}$$

2.3.2.3 off-policy策略算法

综合前面的重要性采样和增量式求均值，我们就可以得到off-policy算法。这里我们的行为策略用的是$\epsilon-soft$，目标策略是贪婪算法。

off-policy首次访问MC控制算法流程：

初始化:对所有$s\in S,a\in A(s)$：

$Q(s,a)\in R$（随机值）

$C(s,a)\leftarrow 0$

一直循环（对每一个回合）：

$b\leftarrow$任何覆盖π的策略

使用策略b生成一个回合：$S_0,A_0,R_1,S_1,\ldots,S_{T-1},A_{T-1},R_T$

$G\leftarrow 0$

$W\leftarrow 1$

对回合的每一步循环，$t=T-1,T-2,\ldots,0$，：

$G\leftarrow\gamma G+R_{t+1}$

$C(S_t,A_t)\leftarrow C(S_t,A_t)+W$

$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\frac{W}{C(S_t,A_t)}[G-Q(S_t,A_t)]$

$\pi(S_t)\leftarrow argmaxQ(S_t,a)$

如果$A_t\neq\pi(S_t)$则退出内循环（进行下一个回合）

$$W \leftarrow W\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

3.实例 21点

3.1游戏规则

21点的游戏规则是这样的：游戏里有一个玩家（player）和一个庄家（dealer），每个回合的结果可能是玩家获胜、庄家获胜或打成平手。回合开始时，玩家和庄家各有两张牌，玩家可以看到玩家的两张牌和庄家的其中一张牌。接着，玩家可以选择是不是要更多的牌。如果选择要更多的牌（称为"hit"），玩家可以再得到一张牌，并统计玩家手上所有牌的点数之和。其中牌面A代表1点或11点。如果点数和大于21，则称玩家输掉这一回合，庄家获胜；如果点数和小等于21，那么玩家可以再次决定是否要更多的牌，直到玩家不再要更多的牌。如果玩家在总点数小等于21的情况下不要更多的牌，那么这时候玩家手上的总点数就是最终玩家的点数。接下来，庄家展示其没有显示的那张牌，并且在其点数小于17的情况下抽取更多的牌。如果庄家在抽取的过程中总点数超过21，则庄家输掉这一回合，玩家获胜；如果最终庄家的总点数小于等于21，则比较玩家的总点数和庄家的总点数。如果玩家的总点数大于庄家的总点数，则玩家获胜；如果玩家和庄家的总点数相同，则为平局；如果玩家的总点数小于庄家的总点数，则庄家获胜。

3.2代码实现

这里我使用的是off—policy策略，其实对于蒙特卡罗方法，最主要的就是解决策略评估和策略控制。下面我将给出实验代码：

```
def obs2state(obs):
    # 将观测信息转换为状态信息
    return (obs[0], obs[1], int(obs[2]))
#策略评估
def evaluate(env, target_policy, behavior_policy, episode_num=500000):
    #初始化
    q = np.zeros_like(target_policy)#q（s,a）
    c = np.zeros_like(target_policy)#前 n 个回报下每个状态的累积权值
    for _ in range(episode_num):
        state_actions = []#状态动作键值对
        observation = env.reset()#获取观测值
        #使用行为策略生成一个回合
        while True:
            state = obs2state(observation)
            action = np.random.choice(env. action_space.n,
p=behavior_policy[state])
            state_actions.append((state, action))
            obs, reward, done, _ = env.step(action)
            if done:
                break
        g = 0  # 回报
        rho = 1.  # 重要性采样比率
        for state, action in reversed(state_actions):
            g = gamma*g+reward #G←γG+Rt+1
            c[state][action] += rho #C(St,At)←C(St,At)+W
            q[state][action] += (rho / c[state][action]*(g - q[state]
[action]))#Q(St,At)←Q(St,At)+W/C(St,At)[G—Q(St,At)]
            rho *= (target_policy[state][action]/ behavior_policy[state]
[action])#W←W*π(At|St)/b(At|St)
    return q

#策略控制
def off_policy(env,target_policy,behavior_policy,espisode_num=500):
```

```
    q=np.zeros_like(target_policy)
    c=np.zeros_like(target_policy)
    for i in range(espisode_num):
        state_action=[]
        obs=env.reset()
        while True:
            state = obs2state(obs)
            action =
np.random.choice(env.action_space.n,p=behavior_policy[state])
            state_action.append((state,action))
            observation, reward, done, _ = env.step(action)
            if done:
                break
            #完成了一个episode
        g=0 # 回报
        rho=1#重要性采样比率
        for state,action in reversed(state_action):
            g = gamma*g+reward #G←γG+Rt+1
            c[state][action]+=rho#C(St,At)←C(St,At)+W
            q[state][action]+=(rho / c[state][action]*(g - q[state]
[action]))#Q(St,At)←Q(St,At)+W/C(St,At)[G−Q(St,At)]
            #策略提升 π(St)←argmaxaQ(St,a)
            a =q[state].argmax()
            if a!=action:
                break
            rho *= (target_policy[state][action]/ behavior_policy[state]
[action])
    return target_policy,q
```

3.3实验结果

```
观测 = [1, 4]
玩家 = [1, 4], 庄家 = [5, 10]
动作 = 1
观测 = (14, 5, False), 奖励 = 0.0, 结束指示 = False

玩家 = [1, 4, 9], 庄家 = [5, 10]
动作 = 1
观测 = (23, 5, False), 奖励 = -1.0, 结束指示 = True
```

图2.一个episode

其中一个episode如图，最开始玩家获得[1,4]的牌，庄家显示了5，策略决定要牌（动作为1），这时候玩家的牌就为[1,4,9]，奖励为0，策略继续决定要牌，结果下一回合的观测得到玩家的牌总和为23点，超过21点，所以游戏结束，奖励-1，庄家获胜。

最优策略图如下：

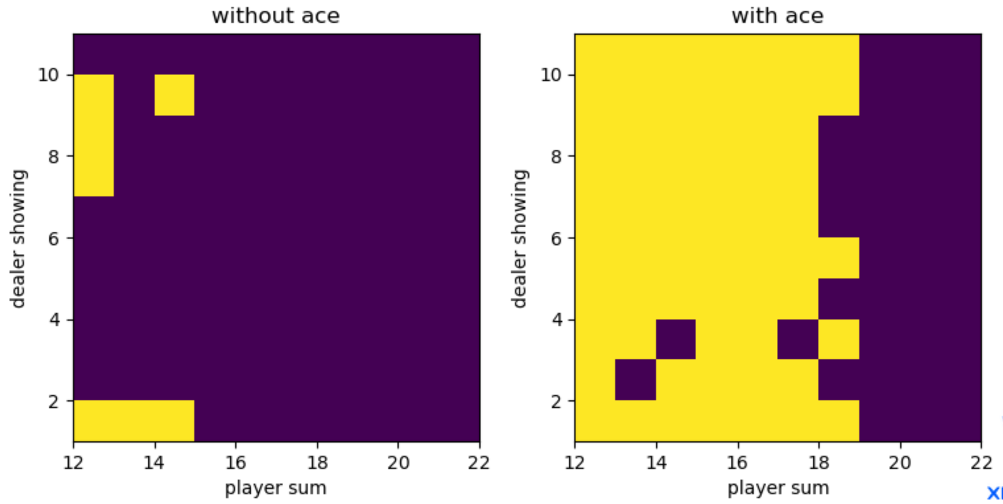

图3.最优策略图

with ace：使用了ace，即a当1；without ace：没使用ace，即a当11，可以看出在最优策略图中，without ace：大部分情况都选择不加牌，with ace：在玩家总和小于18时，大概率都是选择继续加牌。

4总结

（a）蒙特卡洛方法是一个用于估计价值函数和发现最优策略的学习方法。与DP不同的是，我们不需要对环境的完全了解。蒙特卡洛方法只需要状态、动作和与环境实际或模拟交互的奖励的经验样本序列。

（b）蒙特卡洛方法对每个状态 - 动作对的回报进行采样和平均。

（c）策略评估通过平均回报估计状态动作值函数。策略提升使用贪婪策略$\pi(s) = argmaxq(s,a)$

（d）on-policy和off-policy：on-policy方法尝试去估计和提升我们用作决策的那个策略相同；而off-policy估计和提升的策略与用来生成数据的策略不同。

5参考文献

[1]Reinforcement Learning

> 如果您有任何关于内容安全、游戏机器人、数据平台、智能音乐和自动化测试方面的需求，我们也非常荣幸能为您服务。可以联系：*contact@xingzhe.ai*