Monte-Carlo(MC) Learning and Temporal-Difference(TD) Learning

---

实验目的：

- 蒙特卡罗学习 *Monte-Carlo Learning*:
    - First-visit
    - Every-visit
- 时序差分学习 *Temporal-Difference Learning*

运行本项目的方法：

- 添加必要依赖包：
    - numpy
    - secrets
- 添加main.py的执行路径，直接运行即可

实验原理：

1 蒙特卡罗学习 *Moten-Carlo Learning*

- 可以直接从随机 *episodes* 中学习
- *Model-free*，不需要知道 *MDP* 的传递 / 奖励情况，*MDP* 中选取的 *episode* 必须有终点
- 基本思想：**某状态的值函数等于平均采样返回值**

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, \cdots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Monte-Carlo policy evaluation uses *empirical* average sample returns, instead of *expected* return

---

1.1 First-visit

- 算法流程：

  - To evaluate state s
  - The first time-step t that state s is visited in an episode,
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by average return $V(s) = S(s)/N(S)$
  - By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

- 伪代码：

First-visit MC prediction, for estimating $V \approx v_\pi$

Initialize:
  $\pi \leftarrow$ policy to be evaluated
  $V \leftarrow$ an arbitrary state-value function
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
  Generate an episode using π
  For each state s appearing in the episode:
    $G \leftarrow$ return following the first occurrence of s
    Append G to $Returns(s)$
    $V(s) \leftarrow \text{average}(Returns(s))$

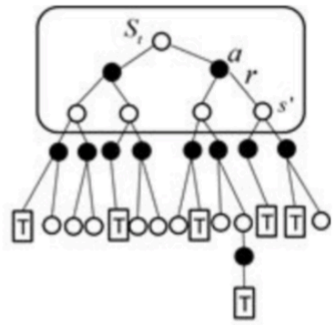
1.2 Every-visit

- 算法流程：
  - To evaluate state s
  - Every time-step t that state s is visited in an episode,
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
  - Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

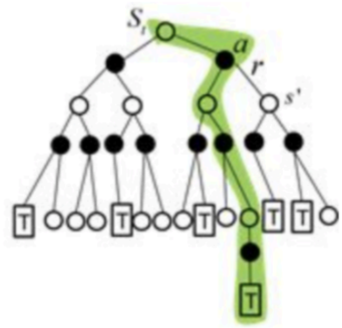
---

2 时序差分学习 *Temporal-Difference Learning*

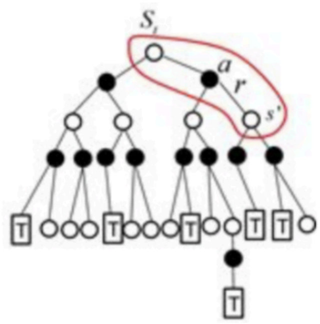
- 时序差分算法是一种无模型的强化学习算法。它继承了动态规划(Dynamic Programming)和蒙特卡罗方法(Monte Carlo Methods)的优点，从而对状态值(state value)和策略(optimal policy)进行预测。从本质上来说，时序差分算法和动态规划一样，是一种bootstrapping的算法。同时，也和蒙特卡罗方法一样，是一种无模型的强化学习算法，其原理也是基于了试验。虽然，时序差分算法拥有动态规划和蒙特卡罗方法的一部分特点，但它们也有不同之处。以下是它们各自的backup图：



动态规划backup图



蒙特卡罗方法backup图



时序差分算法backup图

根据它们的backup图可以知道，动态规划的backup操作是基于当前状态和下一个状态的reward，蒙特卡罗方法的backup是基于一个完整的episode的reward，而时序差分算法的backup是基于当前状态和下一个状态的reward。其中，最基础的时序差分算法被称为TD(0)。它也有许多拓展，如n-step TD算法和TD(lambda)算法。

- ***TD0* 算法**
    - 也需要随机生成 *episode*

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the TD *target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
        $S \leftarrow S'$
    until S is terminal

实验代码分析：

- 由于本次实验对报告不作要求，这里不详细进行项目代码的分析，可以参见`./code`目录下各`.py`文件中完备的代码注释。

实验结果：

- *First-visit in Monte-Carlo Learning*

```
/opt/anaconda3/envs/RLProject/bin/python /Users/dicardo/PycharmProjects/RLProject/main.py
Begin First-visit in Monte-Carlo Learning...
Iteration:  10000 / 30000
Iteration:  20000 / 30000
Iteration:  30000 / 30000
------------------------------------
Monte-Carlo First Visit Policy
------------------------------------
Reshaped Policy (0=UP, 1=RIGHT, 2=DOWN, 3=LEFT):
[[1 0 3 3 3 3]
 [0 0 0 3 3 2]
 [0 0 0 3 2 2]
 [0 0 0 2 2 2]
 [0 0 1 1 2 2]
 [1 1 1 1 1 0]]
------------------------------------
Final Value function:
[[-3.10152496  0.         -3.12369299 -5.42119837 -6.82943082 -7.46415884]
 [-4.6242951  -3.48745637 -4.86440355 -6.20179136 -7.05942877 -7.40848878]
 [-6.00242979 -5.64255494 -6.18582053 -6.79048633 -6.95069977 -6.79495829]
 [-6.97973887 -6.86264934 -6.91087472 -6.72120829 -6.15917367 -5.44208512]
 [-7.6584149  -7.44391829 -6.96076148 -6.11331323 -4.73614512 -3.14715122]
 [-7.96040812 -7.51955209 -6.68223444 -5.24193645 -3.11677254  0.        ]]
------------------------------------
```

- *Every-visit in Monte-Carlo Learning*

```
Begin Every-visit in Monte-Carlo Learning...
Iteration:  10000 / 30000
Iteration:  20000 / 30000
Iteration:  30000 / 30000
Iteration:  40000 / 30000
------------------------------------
Monte-Carlo Every Visit Policy
------------------------------------
Reshaped Policy (0=UP, 1=RIGHT, 2=DOWN, 3=LEFT):
[[1 0 3 3 3 3]
 [0 0 0 3 3 2]
 [0 0 0 0 2 2]
 [0 0 0 2 2 2]
 [0 0 1 1 2 2]
 [1 1 1 1 1 0]]
------------------------------------
Final Value function:
[[-3.01758401  0.         -3.0573248  -5.16448117 -6.50996705 -7.23079251]
 [-4.41637804 -3.33394084 -4.69251593 -5.96234996 -6.79278525 -7.1459652 ]
 [-5.72805433 -5.4186593  -6.04951649 -6.60557481 -6.71485899 -6.54449666]
 [-6.75226138 -6.66425847 -6.76664931 -6.57525289 -5.93771018 -5.18768297]
 [-7.4375885  -7.2348696  -6.79121774 -5.90408989 -4.56038002 -3.0627388 ]
 [-7.71072341 -7.31942471 -6.50951427 -5.1422389  -3.02825946  0.        ]]
------------------------------------
```

- *Temporal Difference Learning*

```
Begin Temporal Difference Learning...
Iteration:  10000 / 50000
Iteration:  20000 / 50000
Iteration:  30000 / 50000
Iteration:  40000 / 50000
Iteration:  50000 / 50000
50000
------------------------------------
Temporal Difference Learning
------------------------------------
Reshaped Policy (0=UP, 1=RIGHT, 2=DOWN, 3=LEFT):
[[1 0 3 3 3 3]
 [0 0 0 3 3 2]
 [0 0 0 0 2 2]
 [0 0 0 1 2 2]
 [0 0 1 1 1 2]
 [1 1 1 1 1 0]]
------------------------------------
Final Value function:
[[-3.23489368  0.         -2.97911011 -5.5911924  -6.94601947 -7.70990246]
 [-4.59902692 -3.41361503 -4.96559358 -6.31674865 -7.25639261 -7.61099477]
 [-6.12573742 -5.72924027 -6.35700056 -7.02520711 -7.13873374 -6.99776407]
 [-7.16874552 -7.15853201 -7.1547871  -6.94469433 -6.12648599 -5.57222909]
 [-7.97062713 -7.76659314 -7.2618965  -6.26008457 -4.6906273  -3.07465217]
 [-8.28138454 -7.81075577 -6.99239598 -5.4507401  -3.27118541  0.        ]]
------------------------------------
```

结果分析 & 实验结论

- 经验证，上述三种算法得到的策略均具有最优性，实验成功。
- 相较于 *Every-visit* 方案，*First-visit* 的收敛速度更快，但稳定性较差，因此需要设置一个最少迭代次数来保证算法的稳定性，对另外两种算法也设置了这样的 *baseline*
- 对于时序差分算法，由于并没有像前两种算法那样基于大数定律，其最终结果是无法达到在阈值 θ 很小情况下的收敛的，因此将其 θ 设置为一个较大的，较易满足的值，例如0.2，而主要控制其最少迭代次数
- 为了进一步保证稳定性，我们同时设置前两种方案需要累计达到两次阈值，才能退出
- 对于时序差分算法，我们对其中的 α （最终取值为0.01，想法是在迭代次数较大时尽量减少V的波动）、θ 和最少迭代次数进行了一定程度的调参，以保证算法的稳定性