**Subject:** Westlake University, Reinforce Learning, Lecture 5, Monte Carlo Learning

**Date:** from January 17, 2025 to January 20, 2025

---

# Contents

# Lecture 5, Monte Carlo Learning

Bilibili:Lecture 5, Monte Carlo Learning

**Outline**

**Lecture 4** 中介绍了寻找 optimal policy 的三种 model-based 算法，本章中将介绍三种 model-free 的算法. 对于算法而言，模型(mode)与数据(data)必需其一，因此 model-free 方法的关键是从数据中学习(learning from data)，故将先用均值估计问题引出这一思想.

$$\text{Outline}\begin{cases} 1. & \text{Motivating example — mean estimation problem} \\ 2. & \text{The simplest MC-based RL algorithm — MC basic} \\ 3. & \text{Use data more efficiently — MC Exploring Starts} \\ 4. & \text{MC without exploring starts — MC } \epsilon\text{-Greedy} \end{cases}$$

## Motivating example

**Motivating example**

**Why need mean estimation?** – 考虑均值估计的原因在于 state value 与 action value 实际上就是在进行均值估计：

$$v_\pi(s) \triangleq \mathbb{E}[G_t \mid S_t = s]$$
$$q_\pi(s,a) \triangleq \mathbb{E}[G_t|S_t = s, A_t = a]$$

我们考虑最简单的掷硬币的实验，掷一次的结果记为 $X$，其是一个随机变量，若为正，则 $X = +1$，反之 $X = 0$，我们的目标是估计 $\mathbb{E}[X]$.

**Method 1: Model-based**

当我们已知概率模型时，如

$$p(X=1) = 0.5, \quad p(X=0) = 0.5$$

那么根据定义就有：

$$\mathbb{E}[X] = \sum_x x \cdot p(x) = 1 \times 0.5 + 0 \times 0.5 = 0.5$$

**Method 2: Model-free**

但有时准确的概率模型获取非常困难，那么就可以进行多次采样以逼近 $\mathbb{E}[X]$. 例如掷了 $N$ 次硬币，结果记为 $\{x_1, x_2, \ldots, x_N\}$，根据大数定律(Law of Large Numbers)就有

$$\mathbb{E}[X] \approx \bar{x} = \frac{1}{N}\sum_{j=1}^{N} x_j$$

这就是 Monte Carlo estimation 的基本思想. 一个简单的数值结果可见 Figure 1.

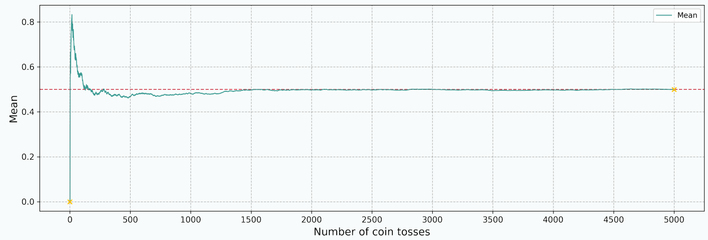


Figure 1: Numerical result of coins

### MC basic —- the simplest MC-based RL algorithm

**Convert policy iteration to be model-free**

MC basic 最本质的 idea 是将 policy iteration 中 policy evaluation 使用 monte-carlo estimation 替换为 model-free 的。回顾 policy iteration 的两个步骤:

1. **Policy evaluation:** $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$
2. **Policy improvement:** 计算 greedy policy $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$, 即

$$\begin{aligned}\pi_{k+1} &= \arg\max_\pi \sum_a \pi(a|s) \left[ \sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s') \right] \\ &= \arg\max_\pi \sum_a \pi(a|s) q_{\pi_k}(s,a)\end{aligned}$$

Policy improvement 的关键是计算 action value $q_{\pi_k}$. 此处分为 model-based 和 model-free 两种方式:

**Method 1: model-based**, 与 Lecture 4 中一样，根据给定的概率模型有:

$$q_{\pi_k}(s,a) = \sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s') \tag{1}$$

**Method 2: model-free(using MC)**, 根据 action value 最原始的定义式有:

$$q_{\pi_k}(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a] \tag{2}$$

其中 $G_t$ 为discounted return，我们需要估计它.

从 $(s,a)$ 开始，根据 policy $\pi_k$ 可以进行 $N$ 次模拟(episode)，它们的 return 分别记为 $g^{(i)}(s,a)$:

$$g^{(i)}(s,a) = r^{(i)}_{t+1} + \gamma r^{(i)}_{t+2} + \gamma^2 r^{(i)}_{t+3} + \cdots, \quad i = 1, 2, \cdots, N$$

那么 $\{g^{(i)}(s,a)\}$就是 $G_t$ 的样本(samples，在强化学习中称为经验，experience)。根据 Monte-Carlo estimation 就有:

$$q_{\pi_k}(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \approx \frac{1}{N} \sum_{i=1}^{N} g^{(i)}(s,a) \tag{3}$$

**MC basic algorithm**

**Algorithm 1** MC Basic Algorithm (a model-free variant of policy iteration)

**Initialization:** Initial guess $\pi_0$.
**Aim:** Search for an optimal policy.
**while** the value estimate has not converged, for the $k$-th iteration **do**
  **for** every state $s \in \mathcal{S}$ **do**
    **for** every action $a \in \mathcal{A}(s)$ **do**
      Collect sufficiently many episodes starting from $(s, a)$ following $\pi_k$.
      **MC-based policy evaluation step:**

$$q_{\pi_k}(s, a) \leftarrow \text{average return of all the episodes starting from } (s, a)$$

    **end for**
    **Policy improvement step:**

$$a_k^*(s) \leftarrow \arg\max_a q_{\pi_k}(s, a)$$

$$\pi_{k+1}(a \mid s) = 1 \text{if } a = a_k^*(s), \text{and } \pi_{k+1}(a \mid s) = 0 \text{ otherwise}$$

  **end for**
**end while**

关于 MC basic algorithm 有几点说明：

1. model-free 算法 MC basic 是建立在 model-based 算法 policy iteration 之上的，唯一区别在于对 action value $q_{\pi_k}$ 的计算从基于模型直接计算变成了基于 MC 的估计.
2. MC basic 直接估计 action value $q_{\pi_k}$ 而非 state value，因为 $q_{\pi_k}$ 的计算仍然需要模型，且若做两次估计可能误差较大.
3. MC basic 在理论上具有启发性，是别的算法的基石，但是由于其效率较低，因此几乎没有应用.
4. 虽然使用了 MC 估计，但是算法与 policy iteration 一样，仍然是收敛的，因为理论上若有足够多的 experience 就可以足够精确地逼近 action value $q_{\pi_k}$，在实际使用中其实没那么多 experience 也可以收敛.

**Sparse reward:** 在使用 MC estimation 时 episode 的长度会对最终的估计产生影响，一个有趣的现象是 **sparse reward**，是指 reward 的设计很稀疏，可能只在达到 target 时才有 positive reward，因此对于一个 episode 的最短步数有要求，因为如果步数不够就没有能到达 target 的可能，也就不能获得 reward，这对学习效率有很大影响，因此解决此问题的一个方案是设计 **nonsparse rewards**，例如到达靠近 target 的区域也设置 positive reward.

## MC Exploring Starts —- Use data more efficiently

### Use data more efficiently

在一个 episode 中，我们会连续访问许多 state-action pair，例如对于这样的一个 episode:

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots$$

我们称每一个 state-action pair $(s_i, a_j)$为一个 **visit**.

在 MC basic 算法中，使用的策略是 **Initial-visit method**，即用一整个 episode 计算出发点处 state-action pair $(s_1, a_2)$的 action value $q_\pi(s_1, a_2)$, 但是其缺点是一个 episode 的数据没有被充分利用，因为一个 episode 事实上可以依次拆分成如下多个 subepisodes:

$$\begin{aligned} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots & [\text{original episode, estimate } q_\pi(s_1, a_2)] \\ s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots & [\text{episode starting from } (s_2, a_4), \text{ estimate } q_\pi(s_2, a_4)] \\ s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots & [\text{episode starting from } (s_1, a_2), \text{ estimate } q_\pi(s_1, a_2)] \\ s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots & [\text{episode starting from } (s_2, a_3), \text{ estimate } q_\pi(s_2, a_3)] \\ s_5 \xrightarrow{a_1} \dots & [\text{episode starting from } (s_5, a_1), \text{ estimate } q_\pi(s_5, a_1)] \end{aligned} \tag{4}$$

**Improvement:** 为能够充分利用一个 episode 的数据，对于一个 episode，可以计算其所有 visit 的 action value.

同时，在一个 episode 中，同一个 state-action pair 可能出现多次(如在例(4)中，出现了两次 $(s_1, a_2)$)，这里就产生了两种方式：

1. **first-visit method**： 只使用第一次出现时计算得到的 $q_\pi(s_i, a_j)$ 作为返回值，第二次的不计算. 这样对于一个 state-action pair，一个 episode 只能得到一次采样.
2. **every-visit method**： 对于重复出现的 state-action pair，每一次都计算 action value. 这样对于一个 state-action pair，一个 episode 可能得到多次采样.

### Update value estimate more efficiently

在 MC-based reinforcement 中另一个重要的问题是什么时候更新 policy，这里仍然有两种方式：

1. **Average return:** 在 MC basic 中，使用多个 episode 取平均的 average return 作为 action value 的估计值，这样需要获得多个 episode 后才能得到，效率较慢但是一次得到的 action value 较准确.
2. **Single episode:** 只使用一次 episode 来估计 action value，这样虽然可能不准确，但更新效率更高。这种思想与 truncated policy iteration 相同.

上面介绍的算法都可以归入 generalized policy iteration(GPI) 的框架中，这种框架是指算法在 policy-evaluation 和 policy-improvement 之间不断循环迭代，而 policy-evaluation 可以精确也可以使用不精确的方式。许多 model-based 和 model-free 的强化学习算法都可以归入这一框架中.

### Algorithm: MC exploring starts

MC exploring starts 算法中 exploring 指我们需要从一个 state-action pair 出发，访问

到所有的 state-action pair 才能知道某一个 action 是否为最优； starts 是指虽然对于每一个 state-action pair，visit 和 start 都能够访问，但由于 visit 有访问不到该 state-action pair 的可能，故还是选用 start 来保证遍历所有的 state-action pair。算法如下：

**Algorithm 2** MC Exploring Starts (a sample-efficient variant of MC Basic)

**Initialization:** Initial guess $\pi_0$.
**Aim:** Search for an optimal policy.
**for** each episode **do**
  **Episode generation:**
  Randomly select a starting state-action pair $(s_0, a_0)$ and ensure that all pairs can be possibly selected. ▷ exploring starts requirement
  Following the current policy, generate an episode of length $T$: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
  **Policy evaluation and policy improvement:**
    **Initialization:** $g \leftarrow 0$
  **for** each step of the episode, $t = T-1, T-2, \ldots, 0$ **do** ▷ compute inversely to improve efficiency
    $g \leftarrow \gamma g + r_{t+1}$
    Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t)$+$g$
    Num$(s_t, a_t) \leftarrow$ Num$(s_t, a_t)$ + 1
    **Policy evaluation:** $q(s_t, a_t) \leftarrow$ Returns$(s_t, a_t)$/Num$(s_t, a_t)$
    **Policy improvement:** $\begin{cases} 1 & \text{if } a = \arg\max_a q(s_t, a), \\ 0 & \text{otherwise.} \end{cases}$
  **end for**
**end for**

**Note 1.** 在计算一个 *episode* 中如果正向计算 $g_t$ 需要预先知晓后续的 $g_{t+1}$

$$g_t = \gamma g_{t+1} + r_{t+1}$$

因此采用反向计算的方式，先计算后续的 $g_t$，以提高计算效率.

此外，只有当一个 *state-action pair* 被充分访问到时才能准确估计其 *action value*，这被称为 ***exploring starts requirement***.

### MC $\epsilon$-Greedy —- MC without exploring starts

**Soft policies / $\epsilon$-greedy policy**

在实际情况中，对于每一个 state-action pair，使用 start 进行计算有可能很难实现(例如真实的机器人在网格世界中每一次 start 都要物理搬运至相应格点)，这时候就需要将 starts 改变成有保障的 visits.

如果在进行 policy update 时，每一个 action 都有正概率被选择，那么只需一个足够长的 episode 也可以访问每一个 state-action pair 很多次，满足 exploring starts requirement. 这样的 policy 被称为 soft policy，即在任意 state 任意 action 都有概率被执行.

MC $\epsilon$-Greedy 的核心就是在 policy update 时将 policy 变成 soft policy 以去除 exploring starts requirement:

$$\pi(a \mid s) = \begin{cases} 1 - \frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1), & \text{for the greedy action} \\ \frac{\varepsilon}{|\mathcal{A}(s)|}, & \text{for the other } |\mathcal{A}(s)| - 1 \text{ actions} \end{cases} \tag{5}$$

其中 $\epsilon \in [0,1]$，$\mathcal{A}(s)$ 为 state $s$ 对应的 action space.

**Note 2.** 可以看到 *soft policy* / *$\epsilon$-greedy policy* 仍然将最大的概率保留给了 *action value* 最大的 *greedy action*，这是由于：

$$1-\frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)|-1)=1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|}\geq\frac{\varepsilon}{|\mathcal{A}(s)|} \tag{6}$$

**Why use $\epsilon$-greedy?** 使用 $\epsilon$-greedy policy 的原因就是平衡 **exploitation**(剥削/选择最优) 和 **exploration**(探索)。Exploitation 是指只选择最优的 action，也就是 greedy，而 exploration 指也指给别的 action 机会，防止陷入局部最优.

1. 当 $\epsilon=0$ 时，就变成了 greedy，此时 less exploration but more exploitation!
2. 当 $\epsilon=1$ 时，所有的 action 选择概率相同，此时 more exploration!
3. 在实际使用中，可以设置较小的 $\epsilon$，然后再逐渐减小.

**How to implement?** 通过 [0,1]上的均匀分布产生一随机数 $x$，若 $x \geqslant \epsilon$ 则选择 greedy action，否则对于 $\mathcal{A}(s)$ 中所有的 action 都有 $\frac{1}{|\mathcal{A}(s)|}$ 的概率选择到.

**Consistency:** 在 $\epsilon$-Greedy 中，每一个 state 采取的 action 都有一定的概率，但是最大概率对应的 action 与 optimal action 一致，那么就称此 $\epsilon$ 下的算法是 consistent 的。一般来说当 $\epsilon$ 较小时会是 consistent.

**MC $\epsilon$-Greedy algorithm**

将 $\epsilon$-greedy 的策略替换掉原先的 greedy 的策略，就可得到 MC $\epsilon$-Greedy algorithm：

**Algorithm 3** MC $\epsilon$-Greedy (a variant of MC Exploring Starts)

**Initialization:** Initial guess $\pi_0$ and the value of $\epsilon \in [0,1]$.
**Aim:** Search for an optimal policy.
**for** each episode **do**
  **Episode generation:**
  Randomly select a starting state-action pair $(s_0, a_0)$. Following the current policy, generate an episode of length $T$: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
  **Policy evaluation and policy improvement:**
    **Initialization:** $g \leftarrow 0$
  **for** each step of the episode, $t = T-1, T-2, \ldots, 0$ **do** ▷ for efficiency
    $g \leftarrow \gamma g + r_{t+1}$
    Use the **every-visit** method: ▷ maybe some visits are long enough
    **if** $(s_t, a_t)$ does not appear in $(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1})$ **then**
      Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t) + g$
      $q(s_t, a_t) \leftarrow$ average(Returns$(s_t, a_t)$)
      Let $a^* = \arg\max_a q(s_t, a)$ and
      $$\pi(a \mid s_t) \leftarrow \begin{cases} \frac{1-\epsilon}{|\mathcal{A}(s_t)|}+\epsilon & \text{if } a = a^*, \\ \frac{\epsilon}{|\mathcal{A}(s_t)|} & \text{if } a \neq a^*. \end{cases}$$
    **end if**
  **end for**
**end for**

## Summary

### Summary

在 **Lecture 4** 中介绍了 model-based 方法，在本节利用数据进行 MC estimation 替换 model-based 中计算 action value 所需的模型部分.

在 **MC Basic** 算法中，介绍了基本思想，即将 action value 的计算替换为 MC estimation.

在 **MC Exploring Starts** 算法中，

1. 为更高效利用数据，使用一个 episode 计算多个 action value，同时对于一个 episode 中重复出现的 visits 使用 **every-visit method** 进行充分利用.
2. 为更高效进行迭代更新，又改用单个 episode 估计 action value.

在 **MC $\epsilon$-Greedy** 算法中，为移除 **MC Exploring Starts** 算法的 exploring starts requirement，引入了 $\epsilon$-greedy policy，如此只需一个足够长的 episode 就可以对所有的 action value 准确估计.

### Codes for plotting

```python
import numpy as np
import matplotlib.pyplot as plt

ite = 5000
random_values = np.random.uniform(low=0.0, high=1.0, size=ite)
positive = np.cumsum(random_values > 0.5)
negitive = np.cumsum(random_values <= 0.5)
mean = positive / (positive + negitive)

qing = (36/255, 144/255, 135/255)
hong = (200/255, 29/255, 49/255)
huang = '#FFC200'

plt.figure(figsize=(16, 5), dpi = 200)
plt.axhline(y=0.5, color=hong, alpha = 0.9, linestyle='--', linewidth=1.2, zorder=0)
plt.plot(mean, color=qing, alpha = 0.9, label='Mean', linewidth=1.2, zorder=1)
plt.scatter(0, mean[0], marker = 'x', zorder=2, color = huang)
plt.scatter(len(mean), mean[-1], marker = 'x', color = huang, zorder=2)


plt.xlabel('Number of coin tosses', fontsize=14, color='black')
plt.ylabel('Mean', fontsize=14, color='black')
plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.8, alpha=0.6, color='gray')
plt.tick_params(axis='both', which='major', labelsize=11, colors='black')
plt.xticks(range(0, len(mean)+1, 500))
plt.legend()
plt.savefig('flip.png', dpi=300, transparent=True)
plt.show()
```

First updated: January 20, 2025
Last updated: April 25, 2025

## References