基于蒙特卡罗方法的博弈游戏，Gaming with Monte Carlo Methods

本章共有3节内容：

- 蒙特卡罗方法
- 蒙特卡罗预测
- 蒙特卡罗控制

开篇

- 在不知道 ***环境的模型*** 的情况下，可以将蒙特卡罗算法应用于强化学习 (RL)。
- 对比马尔可夫决策过程：
  - 马尔可夫决策过程 使用了 **动态规划 (DP)** 来寻找一个最优的策略，**需要了解模型的动态，即转换概率和奖励概率**（transition and reward probabilities）。
- 使用蒙特卡洛算法，可以在不知道模型动态、不了解环境的情况下，找到最佳策略（或接近最佳）。

本章要学的主要内容如下：

- Monte Carlo methods，**蒙特卡罗方法**
- Monte Carlo prediction，**蒙特卡罗预测**
- Playing Blackjack with Monte Carlo，用蒙特卡罗玩21点游戏
- Model Carlo control，蒙特卡罗控制
- Monte Carlo exploration starts，蒙特卡罗探索的开始
- On-policy Monte Carlo control，在线策略的蒙特卡罗控制
- Off-policy Monte Carlo control，离线策略的蒙特卡罗控制

1. 蒙特卡罗方法

开篇

- 蒙特卡罗方法通过随机抽样求取近似解, 即通过运行多个轨迹来近似结果的概率。
- 一种统计技术：通过抽样找到一个近似的答案。

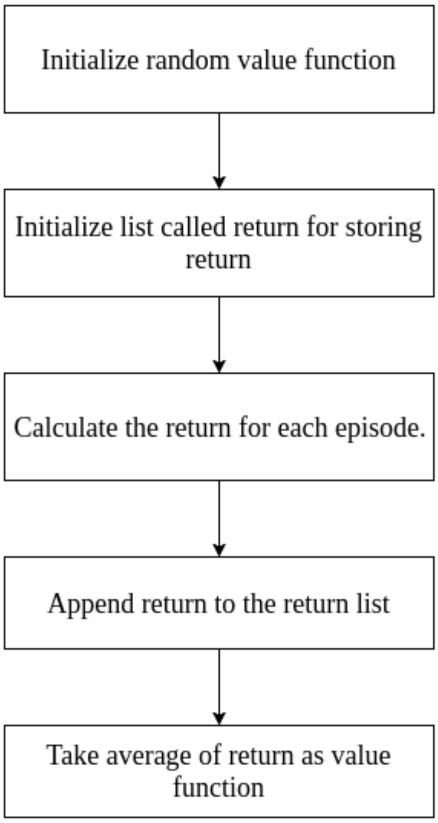
用蒙特卡罗估计 pi 的值 （通过一个例子更好地理解蒙特卡洛）

- 步骤：

1. 首先, 我们在正方形内生成一些随机点。
2. 然后我们可以用方程计算出属于圆内的点数。
3. 然后, 我们通过将4乘以圆内的点数与正方形内的点数的除法来计算的值。
4. ** 如果我们增加样本的数量 (随机点的数量), 我们就能近似得更好

2. 蒙特卡罗预测（用于估计给定策略的值函数）

- 马尔可夫决策的问题
  - 在 DP 中，我们利用值迭代和策略迭代来求解马尔可夫决策过程 (MDP)。
    - 这两种技术都需要转换和奖励概率来找到最佳策略。
- 但是, 在不知道过渡和奖励概率时, 该如何解决问题呢? ——使用蒙特卡洛方法
  - 蒙特卡罗方法只需要状态、action 和奖励的样本序列。
  - 蒙特卡罗方法只适用于情节任务。
  - 由于蒙特卡罗不需要任何模型，因此称为**无模型学习算法**。
- **蒙特卡罗方法的基本思想:**
  - 在马尔可夫决策问题中，**值函数** 约等于一个状态 S（依据策略π）的 Return的期望值。
  - 在这里, 我们使用的不是 **Return的期望值**，而是 **Return的均值**。
  - 因此, 在 ***蒙特卡洛预测*** 中, 我们通过取 **Return的均值** 而不是 **Return的期望值** 来粗略估计 **值函数的值**。
- 利用蒙特卡罗预测，我们 **可以估计任何给定策略的值函数**。
- 蒙特卡罗预测中涉及的步骤非常简单, 如下所示:
  - 首先，我们初始化一个随机的值函数
  - 然后，我们初始化一个空列表return_list，来存储所有Returns
  - 接着，对于 **episode** 中的 **每个state**，我们计算其 **Return**
  - 之后，我们将上一步求出的Return追加到列表 - return_list
  - 最后，我们以 **Return的平均值** 作为值函数 的值（v(s_i) = returns_i 的平均值）
- 下面是表示蒙特卡罗预测的步骤的流程图:



- 蒙特卡罗预测算法有两种类型:
  - 首次访问蒙特卡洛，First visit Monte Carlo
  - 每次访问蒙特卡洛，Every visit Monte Carlo

2.1 首次访问蒙特卡洛，First visit Monte Carlo

- But in the **first visit MC method**, we average the return **only the first time the state is visited in an episode**.
  - 比如，Agent 在玩“蛇和梯子”游戏时，如果被蛇咬了，Agent 很有可能会回到先前的状态。
- 当Agent重新访问状态时，我们不考虑将此次获得的Return用于计算平均Return。
- 只有当Agent首次访问状态时，我们才会考虑将此次获得的Return用于计算平均Return。

2.2 每次访问蒙特卡洛，Every visit Monte Carlo

- In every visit Monte Carlo, we average the return **every time the state**

is visited in an episode.

- 以相同的“蛇和梯子”游戏的为例：如果 Agent 在被蛇咬之后返回到先前的状态，我们可以认为这是**一个待平均的Return**，虽然 Agent 正在重新审视状态。
- 在这种情况下，我们会将 **<u>每次 Agent 访问state时的 Return</u>** 都做平均。

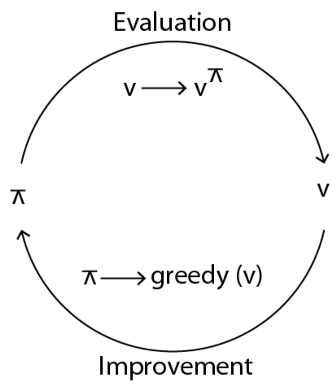
2.3 利用蒙特卡罗方法玩21点游戏，Let's play Blackjack with Monte Carlo

- Blackjack，也叫 21点, 是一种流行的纸牌游戏。
- 21点的游戏规则主要如下：
    - 游戏可以由 若干个玩家 和 一个庄家 一起玩。
        - 每个玩家只与庄家比拼，而不与其他玩家比拼。
    - 最初，会发给玩家两张牌，这两张牌都是正面的，即别人都能看到。
    - 庄家也会获得两张卡，其中一张牌是正面朝上的，另一张是脸朝下的，即庄家只出示了一张牌。
    - 在每一轮比赛中, 玩家决定是否需要另一张牌, 以接近 21点 的距离。
    - 如果玩家需要一张牌，那么它就被称为 **hit**。
    - 如果玩家不再需要牌，那么就叫 **stand**。
    - 如果一个玩家的牌的总和超过 21，那么它被称为 bust (爆炸/破产)；那么庄家就会赢得比赛。
- The rewards here are:
    - +1 if the player won the game
    - -1 if the player loses the game
    - 0 if the game is a draw
- The possible actions are:
    - Hit: If the player needs a card
    - Stand: If the player doesn't need a card
- 接着，我们将使用第一次访问蒙特卡洛算法来玩21点游戏，具体过程可以看书 和 .ipynb文件中的实例。

- 在每个episode中玩一局游戏，
    - 保存其中的 states 和 rewards
    - 对于一局游戏中的每一步（从后往前）

3. 蒙特卡罗控制，Monte Carlo control

- 对比 control 与 prediction 的作用
    - **Monte Carlo prediction**：对 给定策略的值函数 进行估计。
    - **Monte Carlo control**：对 值函数、 策略不断进行优化，从而使得值函数更加准确。
- 在 **control methods** 中，用到了一种新的 **iteration**，即 **“generalized policy iteration”**
    - ***policy evaluation*** **and** ***policy improvement*** 相互作用，循环运行
        - **the policy-π** is always improved with respect to **the value function** （*policy improvement*）
        - but **the value function (v)** is always improved according to **the policy** （*policy evaluation*）
    - 循环运行直到 **π和v** 收敛时，就意味着得到了最优值函数和最优策略:



3.1 Monte Carlo exploration starts

3.1.1 背景与问题

- 与 DP 方法不同，这里我们不估计 **state values** (状态值)。
    - 相反, 我们关注的是 **action values**
    - 如果环境模型已知, 那么仅有 **state values** 就足够了。
- 估计 **action value** 比 估计 **state value** 更直观, 因为 **state value** 因我们选择的策略而异。
    - 例如, 在21点游戏中, 假设我们处于点数为20的状态。此时的 **state value** 是多少，完全取决于政策。
    - 如果我们选择 hit 作为 policy，那么将不会有一个好的状态, 这个 **state value** 是非常低的。然而, 如果我们选择 stand 作为 policy, 那么

很可能会有一个很好的状态。因此, 取决于我们选择的策略。因此, 更重要的是估计 **action value** , 而不是 **state value**。

- 如何 estimate the **action values**?
    - 使用 Q(s, a) 函数进行 estimate
    - But here the problem of exploration comes in.
        - How can we know about *the state-action value* *,* if we haven't been in that state?
        - If we don't *explore all the states* with *all possible actions*, we might probably miss out the good rewards.
- 比如，在21点游戏中, 我们处于一个纸牌之和为20的状态。
    - 如果我们盲目地选择 **hit action**，就很可能会得到 **negative reward** 和 **bad state**。
    - 但如果我们选择 **stand action**，就会得到 **positive reward** 和 **best state**。
    - 所以, 每次来到这个特殊的状态, 我们都站着而不是打。
- 要想知道 哪个是最好的 **action**，我们必须 ***explore all possible actions in each state***, 以找到 **the optimal value** 。
    - 这时候，就需要用到 **MC-ES**。

3.1.2 MC-ES（算法）

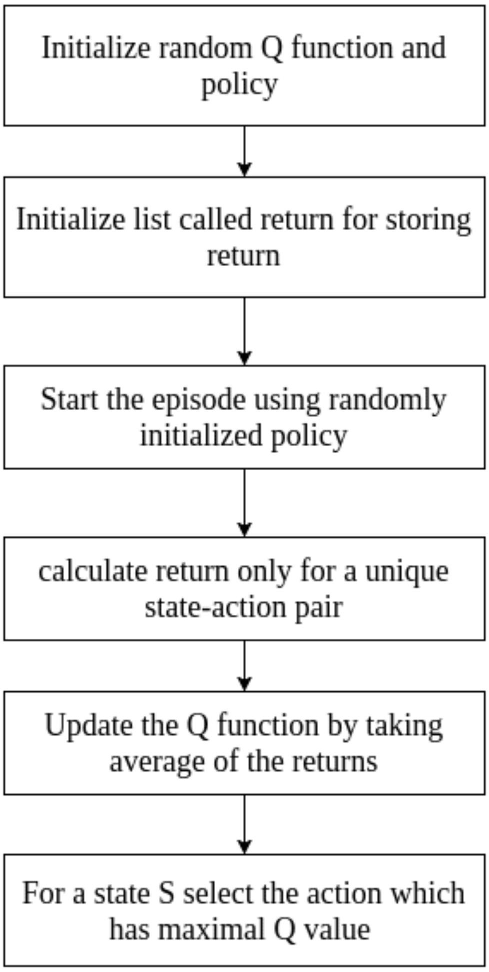
- **Monte Carlo exploring starts ---- 一个新的概念，蒙特卡罗探索开始（MCES）**
    - **MCES**，也被称为 **MC-ES 算法**，it implies that：
        - for each episode
            - ***以 a random state 作为 an initial state***
            - **and perform an action**
    - 如果有足够多个 **episodes**，就可以 cover all the states ***with*** all possible actions.
- **MC-ES 算法的步骤（为每个episode 执行下面的步骤①-⑥）：**
    - ①② 随机初始化 **Q-function 和 policy**，然后初始化一个空列表（用于存储 return）。
    - ③ **we start the episode** with ***our randomly initialized policy***（*~上一步*）
    - ④ ***calculate the return*** for **all the unique state-action pairs**

occurring in the episode

- 然后，append return to our return list
- 注意：We calculate a return only for a unique state-action pair
  - 因为 *the same state-action pair* **occurs** ***in an episode multiple times***，没有额外的信息、意义

- ⑤ 我们在 ***the return list*** 中获取 ***return平均值***, 并将平均值赋予 ***Q-function***
  - we take ***an average of the returns in the return list*** and **assign that value to our Q function**
- ⑥ 我们将为某个状态选择一个最佳策略, 再来选择一个 **action** 使得在该状态下可以获得 最大 **Q(s, a)**。
  - we will ***select an optimal policy for a state***，**choosing an action that has the maximum Q(s,a) for that state**
- **补充说明：**
  - 我们为许多个 **episodes** 重复整个过程（步骤①-⑥），这样我们就可以**覆盖/cover** all the states ***with*** all possible actions.
    - We ***repeat this whole process forever or for a large number of episodes*** so that ***we can cover all different states and action pairs***

- **MC-ES 算法的流程图（步骤）：**



3.2 On-policy Monte Carlo control

- 第一段
  - 在 **Monte Carlo exploration starts**，我们探索所有**state-action pairs**, 并选择给我们最大的价值的那一对。
    - In Monte Carlo exploration starts，we ***explore all state-action pairs*** and ***choose the one that gives us the maximum value***.
  - But think of a situation where we have a large number of states and actions.
    - 但试想这种情况：**states and actions 非常多。**
  - 在这种情况下, 如果我们使用 MC-ES 算法, 那么将需要大量时间来
    - In that case，*if we use the MC-ES algorithm*，then it will ***take a lot of time***
      - 探索状态和操作的所有组合

            - ***to explore all combinations of <u>states and actions</u>***
        - 并选出最佳的组合
            - and ***to choose the best one.***
- 如何解决这个问题?
    - How do we get over this?
- 有两种不同的控制算法。
    - There are two different control algorithms.
        - On policy and off policy.
- In on-policy Monte Carlo control, we use the ε greedy policy.
    - 在 on-policy 蒙特卡罗控制中，我们使用了贪婪的政策。
- Let's understand **what a greedy algorithm is.**
    - 让我们了解**什么是贪婪算法。**

- 第二段
    - 贪婪的算法*可以找到当前可用的最佳选择*, <u>尽管</u>*当你考虑整体问题时*, <u>这种选择可能不是最佳选择</u>。
        - A greedy algorithm ***picks up the best choice available at that moment***, **<u>although that choice might not be optimal</u>** ***when you consider the overall problem***.
    - 设想你想从 **<u>a list of numbers</u>** 中 <u>找到最小的数字</u>。
        - Consider you want to **<u>find the smallest number</u>** from **<u>a list of numbers</u>**.
        - 您将*<u>把列表划分为三个子列表</u>*, 而不是*<u>直接从列表中找到最小的数字</u>*。
            - Instead of ***<u>finding the smallest number directly from the list</u>***, you will ***<u>divide the list into three sublists</u>***.
        - 然后, 您将在 <u>每个子列表</u> 中 *找到最小的数字* (局部最优解) 。
            - Then you will ***find the smallest number*** in **<u>each of the sublists</u>** (local optima).
        - 在*一个子列表中找到的最小数字* 可能不是最小的数字，<u>当考虑到整个列表 (全局优化) 时。</u>
            - ***The smallest number you find in one sublist*** might not be the smallest number **<u>when you consider the whole list (global optima).</u>**
    - 然而, *如果你贪婪*, 那么你<u>只会看到当前子列表中最小的数字 (此时)</u>,

并*<u>认为它是最小的数字</u>*。

        - However, ***if you are acting greedy*** then you **<u>will see the smallest number in only the current sublist (at the moment)</u>** and ***<u>consider it the smallest number</u>***.

- **第三段**
    - 贪婪的 **policy** 会在所探索的行动中选择最佳action（即局部解）。
        - The greedy policy denotes the optimal action within the actions explored.
    - 最佳action是有着最高价值的action。
        - The optimal action is the one which has the highest value.
- 接着，结合一个示例，来进行讲解。
    - exploration-exploitation dilemma：探索新的 - 享受当下 的两难
    - the epsilon-greedy policy
        - ***all actions are tried*** with **<u>a non-zero probability (epsilon)</u>**
            - explore different actions randomly

3.3 Off-policy Monte Carlo control

- 两种 policy：
    - behavior policy
        - explores all possible states and actions and that is why a behavior policy is called a soft policy
    - target policy
        - is said to be a greedy policy (it selects the policy which has the maximal value)
- 两种重要性采样采样的方法：
    - Ordinary importance sampling
    - Weighted importance sampling