5 Model-Free Control

上个章节，我们讨论了如何仅在环境交互的情况也就是不清楚环境是如何运作的条件下去估计给定策略的价值。我们学习到了 Model-free policy evaluation 方法:Monte Carlo policy evaluation 和 TD Learning。在本讲座中，我们将讨论 model-free control，在相同的约束条件下学习良好的策略(只有交互，不知道奖励的结构或转移概率)。该框架在两种类型的领域中很重要:

1.当 MDP 模型未知时，我们通过 MDP 采集大量样本(sample trajectories)或者

2.当 MDP 模型已知时,由于域的大小我们通过基于模型的控制方法计算值函数是不可行的。

在本章节中，我们仍将局限于表格法，我们可以将每个 state 或 state-action 的值表示为查找表的一个元素。在下一章节中，我们将基于值函数近似的条件下重新研究今天和上一章节的算法，其中涉及尝试将一个函数转换为 state 或 state-action 值函数。

5.1 Generalized Policy Iteration

首先回顾我们基于模型的策略迭代(model-based policy iteration)算法，为了方便起见，我们在算法 1 中重写了该算法:

Algorithm 1 Model-based Policy Iteration Algorithm

1: **procedure** POLICY ITERATION(M, ϵ)
2: $\pi \leftarrow$ Randomly choose a policy $\pi \in \Pi$
3: **while** true **do**
4: $V^{\pi} \leftarrow$ POLICY EVALUATION (M, π, ϵ)
5: $\pi^*(s) \leftarrow \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s')\right]$, $\forall\, s \in S$ (policy improvement)
6: **if** $\pi^*(s) = \pi(s)$ **then**
7: break
8: **else**
9: $\pi \leftarrow \pi^*$
10: $V^* \leftarrow V^{\pi}$
11: **return** $V^*(s)$, $\pi^*(s)$ for all $s \in S$

上个章节，我们介绍了无模型策略评估(model-free policy evaluation)方法，因此我们可以用 model-free 的方式完成第 4 行。然而，为了使整个算法是 model-free 的，我们还必须找到一种方法，以 model-free 的方式执行第 5 行，即策略改进步骤。根据定义我们有:

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s')$$

因此,我们能够通过代入Q^{π}的值到 model-based policy iteration 算法得到一个 model-free policy iteration 算法(Algorithm 2)，并且始终使用 state-action，该算法如下:

Algorithm 2 Model-free Generalized Policy Iteration Algorithm

1: **procedure** POLICY ITERATION(M, ϵ)
2: $\pi \leftarrow$ Randomly choose a policy $\pi \in \Pi$
3: **while** true **do**
4: $Q^{\pi} \leftarrow$ POLICY EVALUATION (M, π, ϵ)
5: $\pi^*(s) \leftarrow \arg\max_{a \in A} Q^{\pi}(s,a)$, $\forall\, s \in S$ (policy improvement)
6: **if** $\pi^*(s) = \pi(s)$ **then**
7: break
8: **else**
9: $\pi \leftarrow \pi^*$
10: $Q^* \leftarrow Q^{\pi}$
11: **return** $Q^*(s,a)$, $\pi^*(s)$ for all $s \in S$, $a \in A$

由于我们在第 5 行进行了替换，因此对该算法有一些警告:

1.如果 policy π是已知的(确定的)或者每次行动 a 都没有使用正数的概率采取行动，那么实际上我们是无法计算第五行的 argmax 的。

2.Policy evalutaion algorithm 给我们一个估计值Q^{π}，所以不清楚第五行代码是否会像 model-based 的情况那样一直改善我们的策略(可能有时会变得糟糕?)。

5.2 Importance of Exploration

5.2.1 Exploration

在上一个小节中，我们看到一个关于 model-free policy iteration 算法的警告，是说策略π的每一个行动的概率都要求不小于 0，所以每个 state-action pair 的值都是可以确定的。换句话说，策略π需要对动作进行探索，即使这个探索的结果对我们当前 Q 值的估计可能不是最优的。

5.2.2 ε-greedy Policies

为了探索相对于我们当前的 Q 值估计而言的次优行为,我们需要一个系统的方法来平衡次优行为的探索和最优或贪婪行为的选择。一个很自然的策略就是采取概率很小的随机行动，而其余时间选择选择贪心算法。这种探索策略被称为 ε-greedy policy。数学上，ε-greedy policy 是关于 state-action 值$Q^{\pi}(s,a)$的函数，它有如下形式:

$$\pi(a|s) = \begin{cases} a & \text{with probability } \frac{\epsilon}{|A|} \\ \arg\max_a Q^\pi(s,a) & \text{with probability } 1-\epsilon \end{cases}$$

5.2.3 Monotonic ε-greedy Policy Improvement

通过策略改进定理，在第二课中我们看到，如果对当前值采取贪婪的行为，然后遵循策略π，则此策略是对策略π的改进。那么一个很自然的问题就是，关于ε-greedy policy π的ε-greedy policy 是否是对π真正的进行了改善。这帮助我们定位了对于 generalized policy iteration 算法的第二个警告。幸运的是，有一个关于ε-greedy policy 的策略改进定理的类似定理，我们在下面陈述和推导。

定理 5.1(Monotonic ε-greedy Policy Improvement). 令πi为ε-greedy policy. 则关于Q^{π_i}的ε-greedy policy, 定义π_{i+1}是基于πi的 Monotonic improvement。换句话说:$V^{\pi_i+1} \geq V^{\pi_i}$。

证明:我们首先证明对于所有的状态 s,都能得到:$Q^{\pi_i}(s, \pi_{i+1}(s) \geq V^{\pi}(s)$:

$$\begin{aligned}
Q^{\pi_i}(s, \pi_{i+1}(s)) &= \sum_{a \in A} \pi_{i+1}(a|s) Q^{\pi_i}(s,a) \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + (1-\epsilon) \max_{a'} Q^{\pi_i}(s,a') \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + (1-\epsilon) \max_{a'} Q^{\pi_i}(s,a') \frac{1-\epsilon}{1-\epsilon} \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + (1-\epsilon) \max_{a'} Q^{\pi_i}(s,a') \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1-\epsilon} \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + (1-\epsilon) \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1-\epsilon} \max_{a'} Q^{\pi_i}(s,a') \\
&\geq \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + (1-\epsilon) \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1-\epsilon} Q^{\pi_i}(s,a) \\
&= \sum_{a \in A} \pi_i(a|s) Q^{\pi_i}(s,a) \\
&= V^{\pi_i}(s)
\end{aligned}$$

第一个等式源于这样一个事实，即我们采取的第一个动作是关于策略π_{i+1}，然后我们遵循策略π_i。接下来是第四个等式成立是因为:

$$1-\varepsilon = \sum_a \left[\pi_i(a \mid s) - \frac{\varepsilon}{|A|} \right]$$

根据策略改进定理，我们可以得出：$Q^{\pi_i}(s,\pi_{i+1}(s)) \geq V^{\pi_i}(s)$ 意味着

$V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$ for all state s。

因此，monotonic ε-greedy policy improvement 向我们表明，如果我们对当前 ε-greedy policy 采取ε-greedy 行动，我们的策略实际上确实是会改进。

5.2.4 Greedy in the limit of exploration

目前我们介绍了ε-greedy，它是作为一种朴素的方式来对新状态的探索(或者说新的行动)和对现有知识利用之间的平衡；然而，我们可以通过引入一种新的探索策略来进一步改善这种平衡，这种策略允许我们对算法做出收敛保证。这类策略被称为 Greedy in the Limit of Infinite Exploration(GLIE)。

定义 5.1(Greedy in the Limit of Infinite Exploration(GLIE)).如果满足以下两个属性，策略π is greedy in the limit of infinite exploration :

1.所有的 state-action pairs 都被访问了无数次，例如,对于所有的$s \in S, a \in A$:

$$\lim_{i \to \infty} N_i(s,a) \to \infty$$

其中,$N_i(s,a)$是在 episode i 状态 s 执行动作 a 的次数。

2.(The behavior policy converges to the policy that is greedy with respect to the learned Q-function) 行为策略收敛于学习到的 Q-function 的贪婪策略。即: $for\ all\ \ s \in S, a \in A$

$$\lim_{i \to \infty} \pi_i(s,a) = \arg\max_a q(s,a)\ with\ \ probability\ \ 1$$

GLIE 策略的一个例子是一个ε-greedy policy，其中ε会慢慢被衰减到零,$\varepsilon_i = \frac{1}{i}$,其中 i 是 episode 数,我们可以看到，由于$\varepsilon_i > 0$ for all i，因此我们将在每个时间步以一定的正概率进行探索,从而满足第一个 GLIE 条件。因为$\varepsilon_i \to 0$等于$i \to \infty$,所以我们也知道该策略在极限内是 greedy 的，因此满足第二个 GLIE 条件。

5.3 Monte Carlo Control

现在，我们将把上面描述的探索策略与上节课的 model-free policy evaluation 算法结合起来,给出一些 model-free control 方法。我们讨论的第一个算法是 Monte Carlo online controle 算法。在算法 3 中，我们给出了 first-visit online Monte Carlo Control 算法。如果在算法 3 的第七行检测是否为第一次访问给删除，就变成了 every-visit control。

Algorithm 3 Online Monte Carlo Control/On Policy Improvement

procedure ONLINE MONTE CARLO CONTROL
  Initialize $Q(s,a) = 0$, $Returns(s,a) = 0$ for all $s \in S, a \in A$
  Set $\epsilon \leftarrow 1$, $k \leftarrow 1$
  loop
    Sample kth episode $(s_{k1}, a_{k1}, r_{k1}, s_{k2}, \ldots, s_T)$ under policy π
    for $t = 1, \ldots, T$ **do**
      if First visit to (s,a) in episode k **then**
        Append $\sum_{j=t}^{T} r_{kj}$ to $Returns(s_t, a_t)$
        $Q(s_t, a_t) \leftarrow average(Returns(s_t, a_t))$
    $k \leftarrow k + 1$, $\epsilon = \frac{1}{k}$
    $\pi_k = \epsilon$-greedy with respect to Q (policy improvement)
  Return Q, π

现在，如前所述，GLIE 策略可以帮助我们实现无模型控制方法的收敛保证。特别是，我们有以下结果:

定理 5.2.GLIE Monte Carlo control 收敛域 optimal state-action value function。即:$Q(s,a) \rightarrow q(s,a)$

换句话说，如果算法 3 中使用的ε-greedy policy 是 GLIE，那么从该算法导出的 Q 值将收敛到最优的 Q 函数。

5.4 Temporal Difference Methods for Control

上一个章节，我们已经介绍了 TD(0)作为求解 model-free policy evaluation 的一个方法。现在，我们可以在 TD(0)的基础上，用我们的探索思想来获得 TD-style 的 model-free control。有两种方法: on-policy 和 off policy。我们首先介绍基于 on-policy 的算法 4，因为算法 4 使用(s,a,r,s',a')的值所以被称为 SARSA。

Algorithm 4 SARSA

procedure SARSA(ϵ, α_t)
  Initialize $Q(s,a)$ for all $s \in S, a \in A$ arbitrarily except $Q(terminal, \cdot) = 0$
  $\pi \leftarrow \epsilon$-greedy policy with respect to Q
  for each episode **do**
    Set s_1 as the starting state
    Choose action a_1 from policy $\pi(s_1)$
    loop until episode terminates
      Take action a_t and observe reward r_t and next state s_{t+1}
      Choose action a_{t+1} from policy $\pi(s_{t+1})$
      $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
      $\pi \leftarrow \epsilon$-greedy with respect to Q (policy improvement)
      $t \leftarrow t + 1$
  Return Q, π

SARSA 是一个 on-policy 方法，因为在方程式中更新所使用的操作 a 和 a'都是来自即时更新时遵循的策略。

就像在 Monte Carlo 中一样，如果步长上新增一个额外的条件，我们可以得出 SARSA 的收敛，如下命题所示:

定理 5.3.如果满足下列两个条件，对于 finite-state 和 finite-action 的马尔可夫决策过程,SARSA 将会收敛到最优 action-value，例如:$Q(s,a) \to q(s,a)$.

1.policy π遵循 GLIE(The sequence of policies π from is GLIE)

2.The step-size α_t 满足 Robbins-Munro 序列:

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

练习 5.1。算法 4 第 11 行中的每次更新之后执行策略改进步骤有什么好处?减少执行政策改进步骤的频率好处是什么?

每次都执行改进的好处是每次都改进策略，更快收敛。

减少改进频率的好处是减少运行的次数，提高效率。

5.5 Importance Sampling for Off-Policy TD

在深入学习 off-policy TD-style control 之前，让我们先回退一步,看一种 off-policy TD policy evaluation 的方法。回顾一下 TD 更新的公式:

$$V(s) \to V(s) + \alpha(r + \gamma V(s') - V(s))$$

现在，让我们假设，就像在 off-policy Monte Carlo 案例中一样，我们有来自策略π_b的数据，并且我们想要估计策略π_e的值。然后，就像在 Monte Carlo policy evaluation 案例中一样，我们可以通过行为策略中看到样本的概率与通过重要性抽样评估的策略的比率来加权目标。然后，新的更新采用以下形式:

$$V^{\pi_e}(s) \to V^{\pi_e}(s) + \alpha \left[\frac{\pi_e(a|s)}{\pi_b(a|s)} (r + \gamma V^{\pi_e}(s') - V^{\pi_e}(s)) \right]$$

请注意，因为我们只使用一个轨迹样本，而不是像蒙特卡罗那样对整个轨迹进行采样，所以我们只从一个步骤中引入似然(likelihood)比。出于同样的原因，该方法的方差也明显低于蒙特卡罗方法。

此外，π_b不需要在每一步都相同，但是我们需要知道每一步的概率。就像在蒙特卡洛一样，我们需要这两个策略得到同样的支持。也就是说，如果$\pi_e(a|s) \times V^{\pi_e}(s') > 0$,那么$\pi_b(a \mid s) > 0$。

5.6 Q-Learning

现在，我们回到为 TD-style control 寻找一个 off-policy 方法。在上面的公式中，我们再次利用了重要性抽样，但是在控制情况下，我们不需要依赖它。取而代之的是，我们可以保持 state-action Q 估计值，并引导未来最佳动作的值。我们的 SARSA 更新采取了以下形式:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

但是我们可以基于下一个状态引导 Q 值，以获得以下更新:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a'} Q(s_{t+1}, a' - Q(s_t, a_t)]$$

这就产生了 Q-Learning，这在算法 5 中有详细描述。现在我们对下一个状态的动作取最大值，这一行动不一定与我们从当前策略中得出的行动相同。因此，Q-Learning 被认为是一种 off-policy 算法。

Algorithm 5 Q-Learning withϵ-greedy exploration

procedure Q-LEARNING$(\epsilon, \alpha, \gamma)$
  Initialize $Q(s, a)$ for all $s \in S, a \in A$ arbitrarily except $Q(terminal, \cdot) = 0$
  $\pi \leftarrow \epsilon$-greedy policy with respect to Q
  for each episode **do**
    Set s_1 as the starting state
    $t \leftarrow 1$
    loop until episode terminates
      Sample action a_t from policy $\pi(s_t)$
      Take action a_t and observe reward r_t and next state s_{t+1}
      $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$
      $\pi \leftarrow \epsilon$-greedy policy with respect to Q (policy improvement)
      $t \leftarrow t + 1$
  return Q, π

6 Maximization Bias

最后，我们将讨论一种被称为最大化偏差(**maximization bias**)的现象。我们将首先在一个小例子中考察最大化偏差。

6.1 Example: Coins

假设有两个相同(公平)的硬币，但是我们不知道它们是否是相同的(或者说个公平的)。如果硬币正面朝上，我们得到一美元，如果硬币反面朝上，我们损失一美元。我们想回答以下两个问题:

1.哪种硬币会给未来的翻转带来更多的钱(哪种硬币未来的期望收益更高)?

2.使用问题 1 中的硬币，我每次掷硬币能期望赢/输多少?

为了回答这个问题，我们每枚硬币投一次。然后我们选择能产生更多钱的硬币作为问题 1 的答案。不管硬币给了我们多少，我们都回答了问题 2。例如，如果硬币 1 落在正面，硬币 2 落在反面，我们会用硬币 1 回答问题 1 也就是用一美元回答问题 1。

让我们检查一下这个过程的可能结果。如果至少有一枚硬币是头像，那么我们对问题 2 的回答是一美元。如果两枚硬币都是尾部，那么我们的答案是负一美元。因此，我们对问题 2 的回答的期望值是(3/4)×1+(1/4)×-1 = 0.5。这使我们对投出更好硬币的期望值的估计高于投出该硬币的真实期望值。换句话说，我们会系统地认为硬币比它们实际上更好。

这个问题来自于这样一个事实，我们用我们的估计来选择更好的硬币并估计它的价值。我们可以通过分开这两个步骤来缓解这种情况。这样做的一个方法是改变程序如下:在选择了更好的硬币后，再次翻转更好的硬币，并使用这个值作为问题 2 的答案。这个答案的期望值现在是 0，这与投硬币的真实期望值相同。

6.2 Maximization Bias in Reinforcement Learning

现在让我们将上文中提出的一个状态两个动作给出正式的定义。假设我们处于状态 s，有两个动作a_1和a_2，它们的平均奖励为 0。然后，我们得到 state 以及 state-action 的真实价值为零。也就是说，$Q(s,a_1) = Q(s,a_2) = V(s) = 0$。假设我们已经对每次的行动进行采样，并获得了奖励，这样我们就有了对状态行动值$\hat{Q}(s,a_1), \hat{Q}(s,a_2)$的一些估计。进一步假设这些样本是无偏的，因为它们是使用蒙特卡罗方法生成的。换句话说，$\hat{Q}(s,a_i) = \frac{1}{n(s,a_i)} \sum_{j=1}^{n(s,a_i)} r_j(s,a_i)$，其中$n(s,a_i)$是 state-action 二元组$(s,a_i)$的样本数。令$\hat{\pi} = argmax_a \hat{Q}(s,a)$，也就是对我们的 Q 值估计采取贪婪的政策。那么，我们有了

$$
\begin{aligned}
\hat{V}(s) &= E[\max(\hat{Q}(s,a_1), \hat{Q}(s,a_2)] \\
&\geq \max(E[\hat{Q}(s,a_1)], E[\hat{Q}(s,a_2)]) && \text{by Jensen's inequality} \\
&= \max(0,0) && \text{since each estimate is unbiased and } Q(s,\cdot) = 0 \\
&= 0 = V^*(s)
\end{aligned}
$$

因此，我们的状态值估计值至少与状态 s 的真实值一样大，所以在有限样本存在的情况下，我们系统地高估了状态的值。

6.3 Double Q-Learning

正如我们在上个章节所看到的，当我们有有限多个样本时，状态值依然会遭受 maximization bias。正如我们在硬币示例中讨论的，通过解耦最大值和估计最大值可以消除这种偏差。在 Q-Learning 中，我们能够保持两个独立的无偏估计Q_1, Q_2,我们可以使用一个其中一个估计去选择最大值，另一个去估计最大值。这就产生了 double Q-Learning，该算法如下所示。当我们提到关于Q_1, Q_2的$\epsilon - greedy$策略，意味着在状态 s 时的最优动作等同于:$argmax_a Q_1(s,a) + Q_2(s,a)$。

Algorithm 6 Double Q-Learning

procedure DOUBLE Q-LEARNING(ϵ, α, γ)
  Initialize $Q_1(s,a), Q_2(s,a)$ for all $s \in S, a \in A$, set $t \leftarrow 0$
  $\pi \leftarrow \epsilon$-greedy policy with respect to $Q_1 + Q_2$
  loop
    Sample action a_t from policy π at state s_t
    Take action a_t and observe reward r_t and next state s_{t+1}
    if (with 0.5 probability) **then**
      $Q_1(s_t,a_t) \leftarrow Q_1(s_t,a_t) + \alpha(r_t + \gamma Q_2(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1},a')) - Q_1(s_t,a_t))$
    else
      $Q_2(s_t,a_t) \leftarrow Q_2(s_t,a_t) + \alpha(r_t + \gamma Q_1(s_{t+1}, \arg\max_{a'} Q_2(s_{t+1},a')) - Q_2(s_t,a_t))$
    $\pi \leftarrow \epsilon$-greedy policy with respect to $Q_1 + Q_2$ (policy improvement)
    $t \leftarrow t + 1$
  return $\pi, Q_1 + Q_2$

Double Q-Learning 可以比正常 Q-Learning 学习更快地消除次优动作，从而显着加快训练速度。