目录

第十六章　强化学习

强化学习任务可以用 $E = \langle X, A, P, R\rangle$ 的马尔科夫决策过程 (Markov Decision Process, 简称 MDP) 表示，其中 E, X, A, P, R 分别表示当前环境、状态空间、动作空间、指定状态转移概率和奖赏函数。在强化学习中可以把状态、动作和策略分别对应监督学习中的示例、标记和分类器或回归器；但是强化学习中并没有监督学习中的有标记样本 (示例-标记对)，只有等到策略执行完，才能通过最终的学习结果才能评估动作执行是否正确。强化学习可以看做具有"延迟标记信息"的监督学习问题。

0.1　K-摇臂赌博机

0.1.1　$\epsilon-$ 贪心

每个臂被等概率探索。令 $Q(k)$ 记录摇臂 k 的平均奖赏，若摇臂 k 被尝试了 n 次，得到的奖赏为 $v_1, v_2, \cdots, v_n$，则平均奖赏为:

$$\begin{aligned} Q(k) &= \frac{1}{n}\sum_{i=1}^{n} v_i \\ &= \frac{1}{n}((n-1) \times Q_{n-1}(k) + v_n) \\ &= Q_{n-1}(k) + \frac{1}{n}(v_n - Q_{n-1}(k)) \end{aligned}$$

$\epsilon-$ 贪心算法

输入: 摇臂数 K；奖赏函数 R；尝试次数 T；探索概率 ϵ。

$r = 0$
$\forall i = 1, 2, \cdots, K : Q(i) = 0, count(i) = 0$；$count(i)$ 记录臂 i 被选中的次数
for $i = 1, 2, \cdots, T$ **do**
  if rand() $< \epsilon$ **then**
    $k =$ 从 $1, 2, \cdots, K$ 中以均匀分布随机选取
  else
    $k = \arg\max_i Q(i)$
  end if
  $v = R(k)$;
  $r = r + v$;
  $Q(k) = \frac{Q(k) \times counk(k) + v}{count(k)+1}$
  $count(k) = count(k) + 1$
end for

输出: 累积奖赏 r

0.1.2 Softmax 算法

平均奖赏越高的臂被选中的概率越大。Softmax 算法中摇臂概率的分配是基于 Boltzmann 分布的：

$$P(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}} \tag{1}$$

其中 $Q(i)$ 记录第 i 个臂的平均奖赏；$\tau > 0$ 称为“温度”；τ 越小则平均奖赏越高的摇臂被选取的概率越高。

Softmax 算法

输入： 摇臂数 K；奖赏函数 R；尝试次数 T；探索概率 ϵ。

$r = 0$
$\forall i = 1, 2, \cdots, K : Q(i) = 0, count(i) = 0$；$count(i)$ 记录臂 i 被选中的次数
for $i = 1, 2, \cdots, T$ **do**
  $k =$ 从 $1, 2, \cdots, K$ 中以式子 (1) 选取
  $v = R(k)$;
  $r = r + v$;
  $Q(k) = \frac{Q(k) \times counk(k) + v}{count(k)+1}$
  $count(k) = count(k) + 1$
end for

输出： 累积奖赏 r

0.1.3 UCB 方法

UCB 方法考虑了每个臂被探索的次数。UCB(Upper Confidence Bound，上置信界) 方法每次选择 $Q(k) + UC(k)$ 最大的摇臂，$UC(k)$ 定义为置信区间，例如：

$$Q(k) + \sqrt{\frac{2 \ln n}{n_k}}$$

其中 n 为已执行所有摇臂的总次数，n_k 为已执行摇臂 k 的次数。

0.2 有模型学习

有模型学习 (model-based learning) 即马尔科夫决策过程四元组 $E = \langle X, A, P, R \rangle$ 均为已知。

0.2.1 策略评估

用 $V^{\pi}(x)$ 表示从状态 x 出发，使用策略 π 所带来的累积奖赏，可分为 T 步累积奖赏和 $\gamma \in (0, 1)$ 折扣累计奖赏，可得状态值函数：

$$\begin{cases} V_T^{\pi}(x) = \mathbb{E}_{\pi}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x\right] = \sum_{a \in A} \pi(x, a) \sum_{x' \in X} P_{x \to x'}^{a} (\frac{1}{T} R_{x \to x'}^{a} + \frac{T-1}{T} V_{T-1}^{T}(x')) \\ V_{\gamma}^{\pi}(x) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x\right] = \sum_{a \in A} \pi(x, a) \sum_{x' \in X} P_{x \to x'}^{a} (R_{x \to x'}^{a} + \gamma V_{\gamma}^{T}(x')) \end{cases}$$

函数 $Q^{\pi}(x, a)$ 表示从状态 x 出发，执行动作 a 后在使用策略 π 带来的累积奖赏，用 x_0 表示起始状态，a_0 表示起始状态上采取的第一个动作，可得状态-动作值函数：

$$\begin{cases} Q_T^{\pi}(x, a) = \mathbb{E}_{\pi}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x, a_0 = a\right] = \sum_{x' \in X} P_{x \to x'}^{a} (\frac{1}{T} R_{x \to x'}^{a} + \frac{T-1}{T} V_{T-1}^{T}(x')) \\ V_{\gamma}^{\pi}(x, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x, a_0 = a\right] = \sum_{x' \in X} P_{x \to x'}^{a} (R_{x \to x'}^{a} + \gamma V_{\gamma}^{T}(x')) \end{cases}$$

0.2.2 策略改进

理想的策略应能最大化累积奖赏：

$$\pi^* = \arg\max_{\pi} \sum_{x \in X} V^{\pi}(x)$$

无论是状态值函数还是状态-动作值函数对于策略的每一点改进都是单调递增的。

0.2.3 策略迭代与值迭代

策略迭代 (policy iteration) 是按照策略评估-策略迭代-策略评估-策略迭代。。。以此迭代。

基于 T 步累积奖赏的策略迭代算法

输入： MDP 四元组 $E = \langle X, A, P, R \rangle$；累积奖赏参数 T

$\forall x \in X : V(x) = 0, \pi(x, a) = \frac{1}{|A(x)|}$
loop
  for $t = 1, 2, \cdots$ **do**
    $\forall x \in X : V'(x) = \sum_{a \in A} \pi(x, a) \sum_{x' \in X} P^a_{x \to x'} (\frac{1}{t} R^a_{x \to x'} + \frac{t-1}{t} V^T_{T-1}(x'))$
    if $t = T + 1$ **then**
      break
    else
      $V = V'$
    end if
  end for
  $\forall x \in X : \pi'(x) = \arg\max_{a \in A} Q(x, a)$;
  if $\forall x : \pi'(x) = \pi(x)$ **then**
    break
  else
    $\pi = \pi'$
  end if
end loop

输出： 最优策略 π

值迭代 (value iteration) 改进就是先更新值函数，最后在形成策略。

基于 T 步累积奖赏的值迭代算法

输入： MDP 四元组 $E = \langle X, A, P, R \rangle$；累积奖赏参数 T；收敛阈值 θ

$\forall x \in X : V(x) = 0$
for $t = 1, 2, \cdots$ **do**
  $\forall x \in X : V'(x) = \max_{a \in A} \sum_{x' \in X} P^a_{x \to x'} (\frac{1}{t} R^a_{x \to x'} + \frac{t-1}{t} V^T_{T-1}(x'))$
  if $\max_{x \in X} |V(x) - V'(x)| < \theta$ **then**
    break
  else
    $V = V'$
  end if
end for

输出： 最优策略 $\pi(x) = \arg\max_{a \in A} Q(x, a)$

0.3 免模型学习

免模型学习 (model-free learning) 指算法不依赖与环境建模，有两个困难：**策略无法评估和奖赏函数未知**。也称为无模型学习。

0.3.1 蒙特卡罗强化学习

多次采样，求取平均累积奖赏来作为期望累积奖赏的近似。因为采样次数必须有限，所以这个方法更适合于 T 步累积奖赏的强化学习任务。采样时对策略的选取采用 $\epsilon-$ 贪心法，将确定性的策略 π 称为原始策略：

$$\pi^{\epsilon}(x)=\begin{cases}\pi(x), & \text{以概率}1-\epsilon\\ A\text{中以均匀概率选取的动作}, & \text{以概率}\epsilon\end{cases}$$

因为这里被评估和被改进的是同一个策略，因此被称为“同策略”(one-policy) 蒙特卡罗强化学习算法。

同策略蒙特卡罗强化学习算法

输入： 环境 E；动作空间 A；起始状态 x_0；策略执行步数 T

$Q(x,a)=0, count(x,a)=0, \pi(x,a)=\frac{1}{|A(x)|}$
for $s=1,2,\cdots$ **do**
  在 E 中执行策略 π 产生轨迹：$<x_0,a_0,r_1,x_1,a_1,r_2,\cdots,x_{T-1},a_{T-1},r_T,x_T>$
  for $t=0,1,\cdots,T-1$ **do**
    $R=\frac{1}{T-t}\sum_{i=t+1}^{T} r_i$;
    $Q(x_t,a_t)=\frac{Q(x_t,a_t)\times count(x_t,a_t)+R}{count(x_t,a_t)+1}$
    $count(x_t,a_t)=count(x_t,a_t)+1$
  end for
  对所有已见状态 x：

$$\pi(x)=\begin{cases}\arg\max_{a'} Q(x,a'), & \text{以概率}1-\epsilon\\ \text{以均匀概率从 A 中选取动作}, & \text{以概率}\epsilon\end{cases}$$

end for

输出： 策略 π

异策略 (off-policy) 蒙特卡罗强化学习算法是策略评估时引用 ϵ-贪心，策略改进时改进原始策略。

Algorithm 1 *

异策略蒙特卡罗强化学习算法

输入： 环境 E；动作空间 A；起始状态 x_0；策略执行步数 T

$Q(x,a)=0, count(x,a)=0, \pi(x,a)=\frac{1}{|A(x)|}$
for $s=1,2,\cdots$ **do**
  在 E 中执行 π 的 ϵ-贪心策略 π 产生轨迹：$<x_0,a_0,r_1,x_1,a_1,r_2,\cdots,x_{T-1},a_{T-1},r_T,x_T>$

$$p_i=\begin{cases}1-\epsilon+\epsilon/|A|, & a_i=\pi(x_i);\\ \epsilon/|A|, & a_i\neq\pi(x_i)\end{cases}$$

  for $t=0,1,\cdots,T-1$ **do**
    $R=\frac{1}{T-t}(\sum_{i=t+1}^{T} r_i)\prod_{i=t+1}^{T-1}\max(1,\frac{\mathbb{I}(a_i=\pi(x_i))}{p_i})$;
    $Q(x_t,a_t)=\frac{Q(x_t,a_t)\times count(x_t,a_t)+R}{count(x_t,a_t)+1}$
    $count(x_t,a_t)=count(x_t,a_t)+1$
  end for
  对所有已见状态 $\pi(x)=\arg\max_{a'} Q(x,a')$

11: **end for**
输出: 策略 π

0.3.2 时序差分学习

蒙特卡罗强化学习算法通过完成一个采样估计后再更新策略的值估计，解决了模型未知问题，但是效率很低。时序差分学习 (Temporal Difference Leanring) 则结合了动态规划和蒙特卡罗方法的思想，充分利用了强化学习任务重的 MDP 结构：每执行一步策略就更新一次值函数估计。

分别可得到同策略的 Sarasa 算法和异策略的 Q-学习算法 (Q-learning)。

0.4 值函数近似

前面的强化学习算法都是在**有限状态空间**上进行的，如果状态空间无限，那么就需要将连续状态空间的每一个状态表示成一个状态向量 $\boldsymbol{x}$，然后结合参数向量 $\boldsymbol{\theta}$ 来表示状态的线性函数 (也是值函数):

$$V_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}$$

线性值函数近似 Sarsa 算法

输入: 环境 E；动作空间 A；起始状态 $\boldsymbol{x_0}$；奖赏折扣 γ；更新步长 α
1: $\boldsymbol{\theta} = 0, \boldsymbol{x} = \boldsymbol{x_0}, a = \pi(\boldsymbol{x}) = \arg\max_{a''} \boldsymbol{\theta}^T(\boldsymbol{x}; a'')$
2: **for** $s = 1, 2, \cdots$ **do**
3: $r, \boldsymbol{x}' =$ 在 E 中执行动作 a 产生的奖赏与转移的状态；
4: $a' = \pi^{\epsilon}(\boldsymbol{x}')$
5: $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha(r + \gamma\boldsymbol{\theta}^T(\boldsymbol{x}'; a') - \boldsymbol{\theta}^T(\boldsymbol{x}; a))(\boldsymbol{x}; a)$;
6: $\pi(\boldsymbol{x}) = \arg\max_{a''} \boldsymbol{\theta}^T(\boldsymbol{x}; a'')$;
7: $\boldsymbol{x} = \boldsymbol{x'}, a = a'$
8: **end for**
输出: 策略 π

0.5 模仿学习

0.5.1 直接模仿学习

直接模仿人类专家的“状态-动作对”。

0.5.2 逆强化学习

逆强化学习 (inverse reinforcement learning) 就是**从人类专家提供的范例数据中反推出奖赏函数。**

假设奖赏函数可以表达为状态特征的线性函数即：$R(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$，那么策略 π 的累积奖励可以写成状态向量的加权和的期望与系数 $\boldsymbol{w}$ 的内积:

$$\rho^{\pi} = \mathbb{E}\left[\sum_{i=0}^{+\infty} \gamma^t R(\boldsymbol{x}_t)|\pi\right] = \mathbb{E}\left[\sum_{i=0}^{+\infty} \gamma^t \boldsymbol{w}^T\boldsymbol{x}_t|\pi\right] = \boldsymbol{w}^T\mathbb{E}\left[\sum_{i=0}^{+\infty} \gamma^t \boldsymbol{x}_t|\pi\right]$$

把 $\mathbb{E}\left[\sum_{i=0}^{+\infty} \gamma^t \boldsymbol{x}_t|\pi\right]$ 简写为 $\bar{\boldsymbol{x}}^{\pi}$，把每条范例轨迹上的状态加权求和再平均记为 $\bar{\boldsymbol{x}}^*$，对于最优奖赏函数 $R(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x}$ 和其他任意策略产生的 $\bar{x}^{\pi}$，有:

$$\boldsymbol{w}^{*T}\bar{\boldsymbol{x}}^* - \boldsymbol{w}^{*T}\bar{\boldsymbol{x}}^{\pi} = \boldsymbol{w}^{*T}(\bar{\boldsymbol{x}}^* - \bar{\boldsymbol{x}}^{\pi}) \geq 0$$

所以有最优策略：

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \min_{\pi} \boldsymbol{w}^T(\bar{\boldsymbol{x}}^* - \bar{\boldsymbol{x}}^\pi) \qquad \text{s.t.} \quad \|\boldsymbol{w}\| \leq 1$$

迭代式逆强化学习算法

输入： 环境 E；动作空间 A；状态空间 X；范例轨迹数据集 $D = \{\tau_1, \tau_2, \cdots, \tau_m\}$

1: $\bar{\boldsymbol{x}}^* =$ 从范例轨迹中算出状态加权和的均值向量；
2: $\pi =$ 随机策略
3: **for** $t = 1, 2, \cdots$ **do**
4: $\quad \bar{\boldsymbol{x}}^\pi =$ 从 π 的采样轨迹算出状态加权和的均值向量；
5: $\quad$ 求解 $\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \min_{\pi} \boldsymbol{w}^T(\bar{\boldsymbol{x}}^* - \bar{\boldsymbol{x}}^\pi) \qquad \text{s.t.} \quad \|\boldsymbol{w}\| \leq 1$
6: $\quad \pi =$ 在环境 $\langle X, A, R(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x} \rangle$ 中的求解最优策略
7: **end for**

输出： 奖赏函数 $R(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x}$ 与策略 π

0.6 其他

RL 的缺点：

- 需要大量有效的样本
- 最终表现很多时候并不好; 表现好的时候可能是过拟合
- 需要奖励函数，但是通常难以获得
- 因不当探索和利用 (exploration and exploitation) 导致的局部最优 (local optima) 难以避免
- 结果不稳定，而且很难重现

RL 表现好的条件：

- 容易获得大量样本
- 问题可以化简成一个简单的形式
- 可以进行左右互搏 (self-play)
- 奖励函数可以被定义; 奖励信号丰富，反馈及时

RL 未来的方向和发展：

- 局部最优或许已经足够好
- 靠硬件加速
- 增加更多的学习信号
- 用基于模型的 RL 来提高样本利用率
- 把 RL 当做一种微调手段 (fine-tuning)
- 迁移学习和强化学习结合
- 自学习奖励函数
- 好的先验知识可以大大降低学习时间
- 复杂的问题可能更容易用 RL 解决