

1 强化学习的三种算法

1.1 动态规划

动态规划 (Dynamic Programming, DP) 是运筹学的一个分支, 是求解决策过程最优化的一种数学方法. 动态规划可用于模型 $p(s', r|s, a)$ 已知的马尔科夫决策过程的最优策略的求解, 因此可以作为一种强化学习的方法使用.

1.1.1 策略评估

根据前一节的介绍, 状态价值函数 $v_\pi(s)$ 满足贝尔曼方程:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_\pi(s'))$$

这是以 $v_\pi(s)$ 为变量的线性方程组, 因此可以用线性方程组的解法直接求解. 但是, 这样做的效率往往比较低, 因此动态规划通常将公式看作对 $v_\pi(s)$ 的迭代公式, 利用如下迭代法来进行求解:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_k(s'))$$

数学上可以证明

$$\lim_{k \rightarrow \infty} v_k(s) = v_\pi(s)$$

因此, 实际求解时只需将 $v_k(s)$ 初始化, 例如赋值为 0, 然后迭代直至收敛即可.

1.1.2 策略改进

利用策略评估得到某个策略 $\pi(a|s)$ 下的价值函数 $v_\pi(s)$ 以后, 就可以据此进行策略改进 (Policy Improvement), 即通过优化当前策略以获得更优策略, 其核心目标是让新策略的价值函数不低于原策略, 以便最终逐步逼近最优策略.

当智能体面对状态 s 时, 如果一直遵循策略 π , 则回报是 $v_\pi(s)$. 如果智能体不遵循策略 π 而采取行动 a , 之后再遵循 π , 此时的回报就是策略 π 下的行动价值函数

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_\pi(s'))$$

如果 $q_\pi(s, a) > v_\pi(s)$, 那么这样的选择就比一直遵循 π 更好. 于是下次处于状态 s 时就仍应选择行动 a . 这样得到的新策略 π' 就应当比 π 更优. 一般的, 我们有如下定理.

定理 1.1 策略改进定理 对于确定性的策略 π 和 π' , 如果对于任一状态 $s \in \mathcal{S}$ 都有

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

则 π' 不劣于 π , 即 $v_{\pi'}(s) \geq v_{\pi}(s)$.

如果是随机性策略, 则上述式子应改写为

$$\sum_{a \in \mathcal{A}} \pi'(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_{\pi}(s')) \geq v_{\pi}(s)$$

证明. □

由此, 我们可以采用前面所述的贪心的方法更新策略. 对于基于策略 π 的价值函数 $v_{\pi}(s)$, 可以做前向单步搜索, 构造如下贪心策略:

$$\pi'(s) = \arg \max_a q_{\pi}(s, a) = \arg \max_a \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_{\pi}(s'))$$

由上式的最大化结果可知 $q_{\pi}(s, \pi'(s)) = \max_a q_{\pi}(s, a)$, 是所有行动 a 中使得 $q_{\pi}(s, a)$ 中最大的. 另一方面又有 $v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_{\pi}(s, a)$ 是所有 $q_{\pi}(s, a)$ 的加权平均, 因此总有 $q_{\pi}(s, \pi'(s)) \geq v_{\pi}(s)$, 满足策略改进定理, 因此 π' 总是比 π 更好的策略 (除非 π 已经是在 s 下最优的策略).

总之, 我们证明了使用单步的贪心算法不断迭代就能使策略最优化. 这是动态规划求解强化学习问题的核心机制之一.

1.1.3 策略迭代与价值迭代

按照上述过程, 我们可以把策略迭代过程描述如下:

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \dots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi_*}$$

其中 E 为策略评估步骤, I 为策略改进步骤. 由于每次策略改进都使策略更优, 因此算法能保证收敛到最优解. 这样的 EI 循环与聚类问题中的 EM 循环是非常相似的.

在策略评估 E 步骤中, 通常需要反复迭代才能得到策略 π_k 的价值函数 $v_{\pi_k}(s)$. 由于 π_k 本来就作为需要改进的量, 因此精确计算并无必要, 我们可以只进行一次迭代. 于是就有以下的价值迭代算法:

$$v_{k+1}(s) = \max_a \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_k(s'))$$

这就把 E 和 I 步骤合为一步. 在收敛后即可导出最优策略 π_* .

1.2 蒙特卡罗方法

在强化学习中, 蒙特卡罗 (Monte Carlo, MC) 方法是一类基于采样轨迹来学习价值函数或策略的方法. 它不需要关于环境的完整信息 (即 $p(s', r|s, a)$), 而是直接从经验中学习, 即利用来自真实或模拟的环境交互中采样得到的状态, 行动, 回报的序列数据. 蒙特卡罗方法的核心思想是通过多次试验 (采样) 的平均结果来逼近真实的期望价值.

1.2.1 价值估计与策略改进

状态 s 在策略 π 下的价值函数就是 G_t 的期望值, 即 $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$. 从一条轨迹可以直接得到

$$G_t = \sum_{k=0}^T \gamma^k R_{k+t+1}$$

蒙特卡罗方法就是通过在策略 π 下多次采样包含状态 s 的完整轨迹, 计算每次轨迹中从 s 开始的 G_t , 取其均值作为 $v_\pi(s)$ 的估计.

常用的**首次访问型 MC 算法 (First-visit MC)** 的流程如下:

1. 输入待评估的策略 π .
2. 对于所有状态 s , 初始化 s 的采样列表 $R(s)$.
3. 根据 π 生成一幕序列

$$S_0, A_0, R_1, S_1, A_1, R_1, \dots, S_{T-1}, A_{T-1}, R_T$$

对本幕中的每一步循环, 分别令 $t = T-1, T-2, \dots, 0$, 根据 $G_t = \gamma G_{t+1} + R_{t+1}$ 计算 G_t , 然后如果 S_t 没有在 S_0, \dots, S_{t-1} 中出现过, 就将 G_t 加入 S_t 的采样列表 $R(S_t)$ 中.

4. 重复上述采样操作, 最后根据 S_t 的采样列表 $R(S_t)$ 的均值计算 $v_\pi(S_t)$.

即对每条轨迹中首次出现的 s 进行记录和平均. 另外, 还有**每次访问型 MC 算法 (Every-visit MC)**, 即对每条轨迹中所有出现的 s 都进行记录和平均.

当采样的轨迹足够多时, 两种算法都能收敛到 $v_\pi(s)$. 当轨迹有限时, 首次访问型 MC 算法是无偏的, 而每次访问型 MC 算法则是有偏的. 当轨迹较少时, 每次访问型 MC 算法效果更好, 因为它能提取更多的数据进行计算; 当轨迹较多时, 首次访问型 MC 算法收敛更快. 另外, 对动作价值函数 $q_\pi(s, a)$ 的计算也是类似的.

1.2.2 基于重要性采样的离轨策略

所有的学习控制方法都面临一个困境: 希望学到的策略可以使随后的行为是最优的 (此时往往只有一个最优行动), 但是为了搜索所有的行动 (以保证找到最优行动与策略), 又需要采取非最优行动. 我们可以采用两个策略, 一个用于学习并成为最优策略, 而另一个则更加具有探索性, 用于形成采样的样本. 普适地讲, 它们分属于以下两种类型:

同轨策略: 用于生成采样数据序列的策略 b 和用于实际决策的待评估和改进的策略 π 是相同的.

离轨策略: 用于生成采样数据序列的策略 b 和用于实际决策的待评估和改进的策略 π 是不同的.

通俗而言, 同轨策略可以类比为为自己下棋并提高水平, 而离轨策略可以类比为看别人下棋提高水平. 我们现在来证明离轨策略的有效性.

证明. 从 S_t 出发, 轨迹 $S_t, A_t, S_{t+1}, A_{t+1}, \dots, S_T$ 在策略 π 下发生的概率为

$$p(A_t, S_{t+1}, \dots, S_T | S_t, \pi) = \prod_{k=t}^{T-1} \pi(A_k | S_k) p(S_{k+1} | S_k, A_k)$$

类似地可得出同样的轨迹在策略 b 下的概率. 因此, 同一条轨迹在两种策略下的概率之比为

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)}$$

因此, 注意到这概率与环境的响应机制 $p(S_{k+1}|S_k, A_k)$ 无关, 只与两个策略有关.

根据定义, 策略 b 下的价值函数为

$$v_b(s) = \mathbb{E}_b[G_t|S_t = s] = \sum_{A_t, S_{t+1}, \dots, S_T} p(A_t, S_{t+1}, \dots, S_T|S_t, b)G_t$$

而策略 π 下的价值函数可以写成

$$\begin{aligned} v_\pi(s) &= \sum_{A_t, S_{t+1}, \dots, S_T} p(A_t, S_{t+1}, \dots, S_T|S_t, \pi)G_t \\ &= \rho_{t:T-1} \sum_{A_t, S_{t+1}, \dots, S_T} p(A_t, S_{t+1}, \dots, S_T|S_t, b)G_t \\ &= \mathbb{E}_b[\rho_{t:T-1}G_t|S_t = s] \end{aligned}$$

因此, 根据策略 b 下产生的蒙特卡罗模拟轨迹可以调整计算得到策略 π 下的结果:

$$v_\pi(s) = \frac{\sum_{\text{traj}} \rho_{t:T-1}G_t}{\sum_{\text{traj}} 1} = \frac{1}{N} \sum_{\text{traj}} \rho_{t:T-1}G_t$$

其中 traj 是满足 $S_t = s$ 的采样轨迹, N 是采样的轨迹数目.

另外, 我们知道轨迹概率满足如下归一性质:

$$\sum_{A_t, S_{t+1}, \dots, S_T} p(A_t, S_{t+1}, \dots, S_T|S_t, \pi) = 1$$

于是我们也可以把 $v_\pi(s)$ 改写为

$$v_\pi(s) = \frac{\sum_{A_t, S_{t+1}, \dots, S_T} p(A_t, S_{t+1}, \dots, S_T|S_t, \pi)G_t}{\sum_{A_t, S_{t+1}, \dots, S_T} p(A_t, S_{t+1}, \dots, S_T|S_t, \pi)}$$

然后把 $\rho_{t:T-1}$ 代入上式可得

$$v_\pi(s) = \frac{\mathbb{E}_b(\rho_{t:T-1}G_t|S_t = s)}{\mathbb{E}_b(\rho_{t:T-1}|S_t = s)} = \frac{\sum_{\text{traj}} \rho_{t:T-1}G_t}{\sum_{\text{traj}} \rho_{t:T-1}}$$

□

于是我们得到了两种计算 $v_\pi(s)$ 的公式. 它们在采样轨迹数目足够多时都收敛到真值; 在轨迹有限时, 前者是无偏的, 但方差较大; 后者是有偏的, 但方差较小. 实际应用中一般采用后者.

1.3 时序差分算法

1.3.1 时序差分预测

时序差分算法 (Temporal Difference, TD) 是强化学习中一种结合了动态规划和蒙特卡罗方法优点的重要算法, 核心是通过 bootstrapping(利用估计值更新估计值) 和在线学习 (无需等待完整轨迹结束) 来更新价值估计.

在蒙特卡罗模拟的在线学习中, 根据 $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$ 可得如下更新公式

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

其中 $V(S_t)$ 是 $v_\pi(S_t)$ 的估计值, η 是学习率, G_t 需要等待一条轨迹结束之后才能运算得到.

现在, 根据 G_t 的性质可知 $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$, 于是可以把更新公式改写为

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

其中 R_{t+1} 和 S_{t+1} 在单步完成后即可得到. 利用上式即可实现简单的时序差分算法, 称作单步 TD 法.

1.3.2 时序差分控制与 Q 学习

在决策优化问题中更重要的是动作价值函数. 类似地, 我们可以写出 $q_\pi(s, a)$ 的估计 $Q(S_t, A_t)$ 的更新公式:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

这里需要使用轨迹数据的五元组 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, 因此算法称作 **SARSA 算法**. 这本质上是一种同轨策略. 这一算法的具体步骤为:

- (i) 初始化 $Q(s, a)$ 表, 并且选定初始状态 S .
- (ii) 根据当前状态 S_t 和当前策略选择行动 A_t .
- (iii) 执行动作 A_t , 获得回报 R_{t+1} , 到达下一个状态 S_{t+1} .
- (iv) 根据状态 S_{t+1} 和当前策略选择行动 A_{t+1} .
- (v) 使用上述公式

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

更新 $Q(S_t, A_t)$.

- (vi) 重复上述 (ii) 到 (v) 步骤直到终止状态, 然后结束当前回合, 进入下一回合学习.

当然, 也可以使用离轨策略下的时序差分控制. 根据贝尔曼最优方程

$$q_*(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) \left(r + \gamma \max_{a' \in \mathcal{A}} q_*(s', a') \right)$$

并且根据时序差分预测中的更新公式可得

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta \left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

这就是 **Q-学习 (Q-Learning)** 的基本思想. 这是一种离轨策略: 学习的是最优策略 (通过 \max 操作实现), 而实际模拟轨迹的是探索策略 (例如 ε -贪心算法). 与 SARSA 相比, Q-学习不需要 A_{t+1} , 而是直接取可能的最大价值, 因此能更加稳定地逼近最优的 Q 函数.