强化学习算法实践报告

程业翔 2000012158

实践内容：

分别使用 Q-learning 与 Policy Gradient 算法训练 Agent, 在大小为 10*10 的方格世界中，走出一条途经且途径一次所有中间城市((0, 4), (2, 1), (2, 3), (3, 5), (6, 2), (7, 7), (9, 5))并回到起点位置(0, 0)的最短路径。

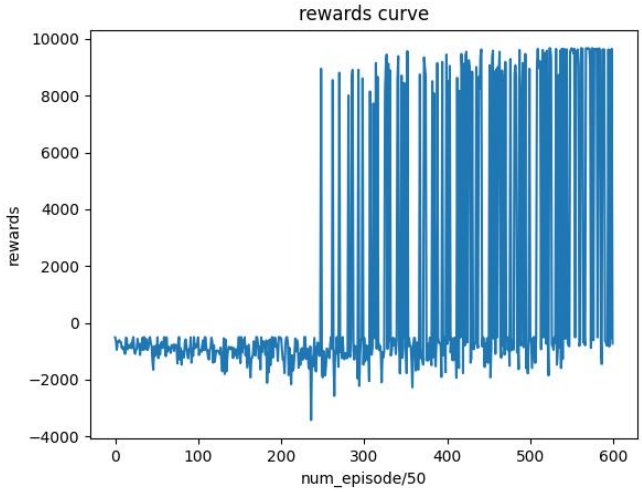
项目描述：

详见 README.md

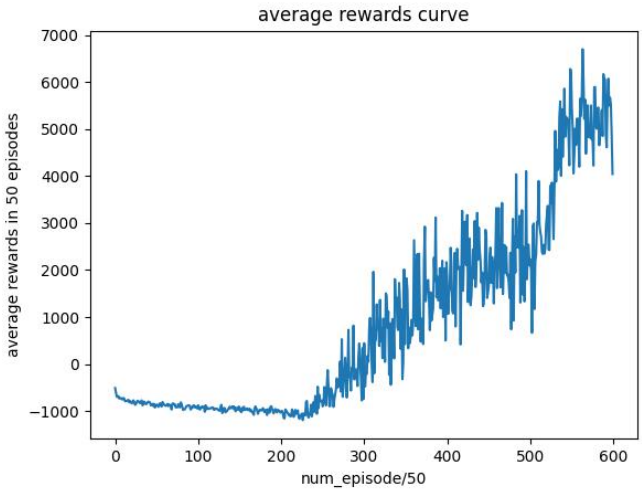
实现细节：

此模块的参数见 parameter3 文件夹，学习曲线见 learning_fig3 文件夹。

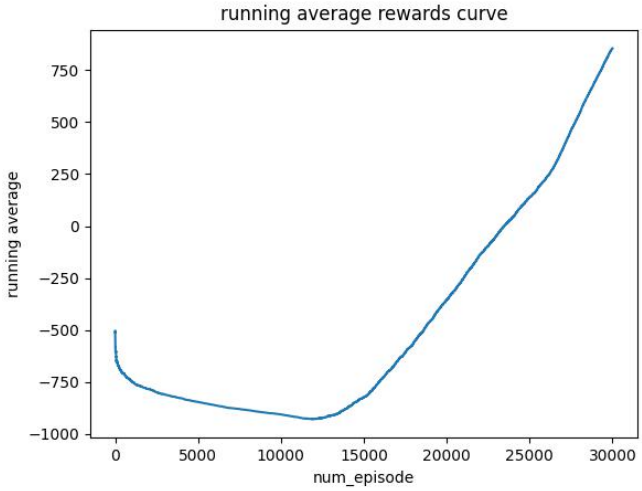
1. environment
   (1) 将走出边界/经过相同的城市作为终止态，并令其返回的 reward 为-500(记作 r_1)；将途径所有城市且仅经过一次并走回起点作为另一终止态，并令其返回的 reward(记作 r_f)为 10000；
   (2) 当基于 Q-learning 的 Agent 到达一座新的城市时，不获得 reward; 当基于 Policy gradient 的 Agent 到达一座新的城市时，获得值为 666 的 reward。
   (3) 为了得到最短路径，agent 每次行动若未到达上述状态，会得到值为-10 的 reward(记作 r_3)；
   由此得到马尔可夫决策过程：S 用一个三元列表表示，三个元素分别为纵坐标、横坐标、到达的城市（用七位二进制数表示）; A 为[‘UP’, ‘DOWN’, ‘LEFT’, ‘RIGHT’]，状态转移过程即为在方格世界中移动的过程，如果到达一个新的城市则将 S[2]中对应的位变为 1；R 如上所述；P 为 1 或 0，因为在每个状态的每个行动都只可能到达一个特定的状态；γ根据算法不同有不同的设置。
   实验发现，若$r_1 < \frac{1}{1-\gamma} r_3$, 则 Agent 会更倾向于直接进入终止态而非探索，这点对使用 Policy Gradient 算法训练的 Agent 更为明显。
2. Q-learning
   (1) Q-table 形式：10*10*2^7*4 的 numpy 数组，前三维对应不同的状态，后一维对应四个行动，初始化为 0；
   (2) 行动的选择：在学习阶段，使用ε-greedy 策略，设置ε=0.1，即 Agent 有 0.1 的概率随机选择一个行动，以探索更多的可能性，同时 Agent 有 0.9 的概率选择 Q 值最大的行动；在评估阶段，Agent 选择 Q 值最大的行动。
   (3) 超参数设置：学习率α=0.5, 折扣因子γ=0.5；
   (4) 共训练了 30000 个 episodes, 每 1000 个 episodes 更新α=0.99α, 保证 Q-learning 论文中的收敛条件$\sum \alpha^2 < \infty$成立（虽然在此任务中，固定α也可以得到最优解）
   训练过程中 reward 曲线如下：



每 50 个 episodes 记录一次



每 50 个 episodes 求平均并记录



滑动平均

可以看出，由于使用了ε-greedy 策略，reward 在升高的趋势下有较大波动。在 1000 个 episode 左右就找到了一条可以完成巡回的路径。最终达到全局最优。

3. Policy Gradient
   (1) 参数θ形式：10*10*2^7*4 的 tensor 张量，前三维对应不同的状态，后一维对应四个行动，初始化为 0；
   (2) 行动的选择：在学习阶段，使用 softmax 函数得到当前状态采取四个行动的概率，依此概率进行采样；在评估阶段，直接采取概率最大的行动。
   (3) 超参数设置：学习率 lr=0.02, 折扣因子γ=0.98, 每次更新参数时的 episodes 数 batch_size=100；
   (4) 学习方法：参数θ的更新公式如下，其中 N 为更新参数时的 episodes 数，$R(\tau^n)$为在这 N 个 episodes 中的第 n 个 episode 中的收益 $p\left(a_t^n \middle| s_t^n, \theta\right)$为在状态$s_t^n$及参数θ下选择行动$a_t^n$的概率，在本框架中即为(2)中 softmax 函数的输出中选择a_t^n的概率。

$$\theta = \theta + lr * \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} R(\tau^n)\nabla logp\left(a_t^n \middle| s_t^n, \theta\right)$$

   但在实验过程中，使用上述公式的效果很不理想，原因主要有两点：
   (i) 对于在某个状态下的最优行动，可能由于之前操作累积的负 reward 太多，从而在梯度上升后被选择的概率下降。在大量抽样下该行动被选择的概率最终也可能达到很高，但在样本量不足且该状态下其他行动也能得到较高的 reward 的情况下，有可能收敛到局部最优。
   (ii) 对于一条能够得到非常大的 reward(如到达正确的终止态)的路径，在一次梯度上升过程后，Agent 有较大概率再次走出此路径（尽管这条路径可能不是最优的），使梯度再次上升，之后有更大概率走出此路径，使梯度再次上升……从而陷入局部最优。尽管 softmax 函数使 Agent 拥有一定探索其他路径的可能性，但这可能性太小，而且往往得到的 reward 差于上述局部最优路径的 reward。

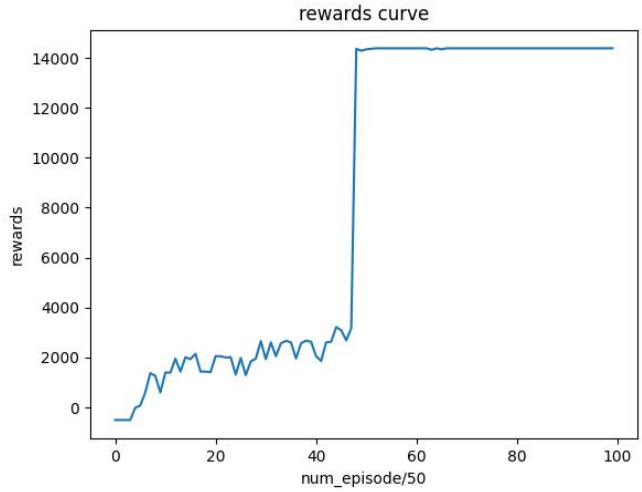
   综上，在算法中使用下述公式代替 $R(\tau^n)$:

$$\sum_{i=t}^{T} \gamma^{i-t} r_i^n - ER$$

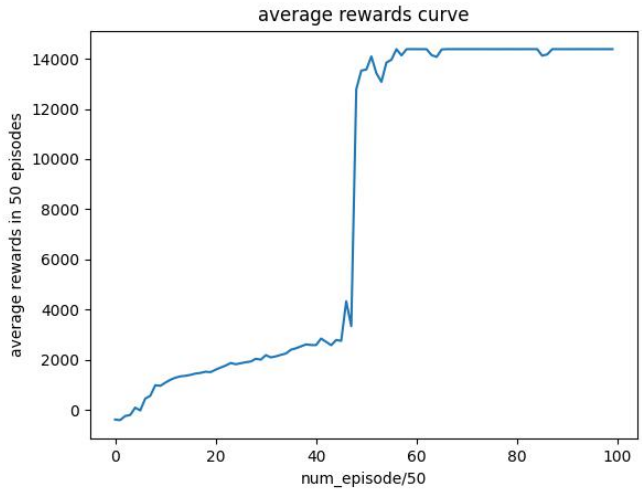
   其中，ER 为前一项均值的滑动平均，即：

$$ER = (1 - lr) * ER + lr * \left(\frac{\sum_{t=1}^{T}\sum_{i=t}^{T}\gamma^{i-t} r_i^n}{T} - ER\right)$$

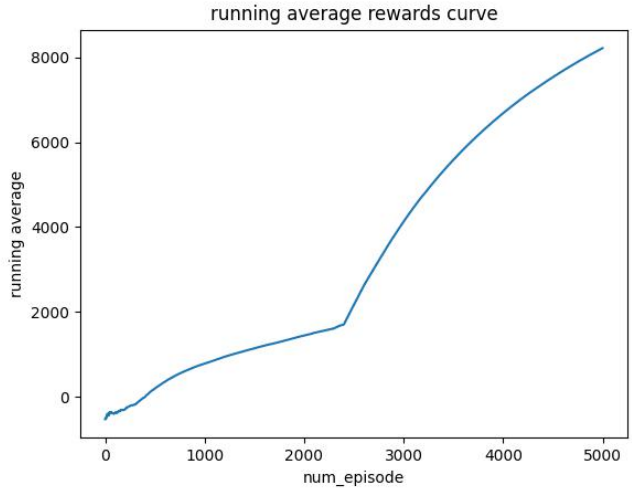
   如此，当前状态下行动的收益只与之后的过程相关；一个 reward 绝对值很大的决策过程不会使得参数向一个方向无限积累。
   (5) 更新参数：将$-rlogp$ 作为 loss 累加，每 batch_size 个 episodes 反向传播并使用 SGD 优化器更新一次参数。（负的梯度下降相当于正的梯度上升）
   (6) 共训练了 5000 个 episodes，训练过程中 reward 曲线如下：



每 50 个 episodes 记录一次



每 50 个 episodes 求平均并记录



滑动平均

由于 Policy gradient 算法只保证收敛到局部最优，在本次实验中，基于 Policy gradient 算法的 Agent 最终巡回路径长为 36。

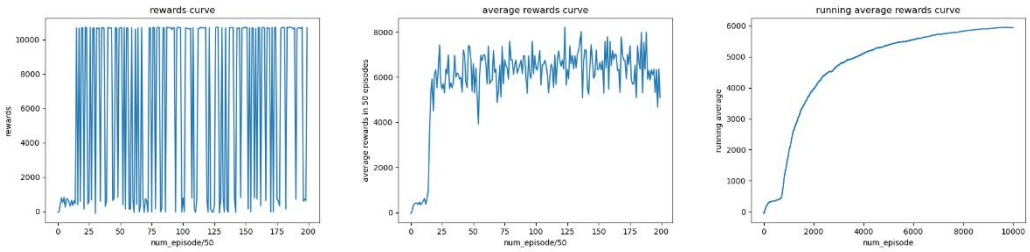
自主探索：

1. 带先验的学习：

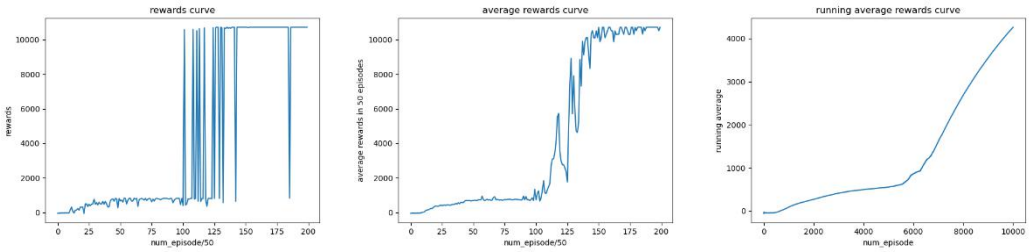
为了能让基于 Policy gradient 的 Agent 收敛到全局最优，针对此 TSP 问题的实例，引入先验知识：只有在 agent 第一次到达[(0, 4), (2, 3), (3, 5), (7, 7), (9, 5)]这五座城市时，赋予其值为 200 的 reward(记作 r_2)。

在此基础上，基于 Policy gradient 的 Agent 可以得到全局最优解。得到的参数见 parameter 文件夹，学习曲线见 learning_fig 文件夹。

Q-learning:



Policy gradient:



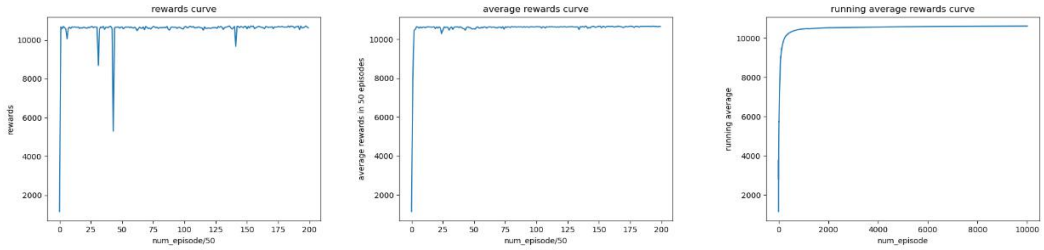
2. 只有一个终止态的学习：

对于上述 environment 稍作调整，仅设置一个终止态，即遍历所有城市并回到起点。这并不能保证 Agent 仅经过一次城市并且不出界，但由于每次行动得到的负 reward，Agent 会倾向于走更短的路径，亦可达到目的。

在此基础上，得到的参数见 parameter2 文件夹，学习曲线见 learning_fig2 文件夹。

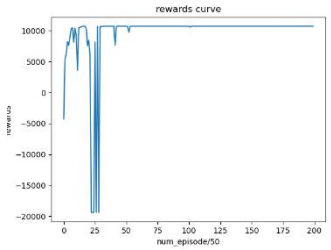
Q-learning:

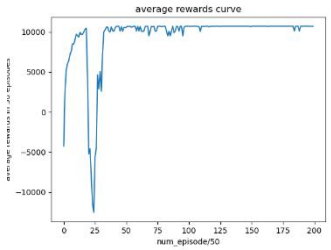
设置单个 episode 上限行动次数为 5000

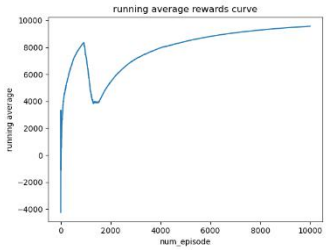


发现 Agent 的学习过程可以看作得到可以巡回的路径再逐步优化，最终也能得到最优解。

Policy Gradient:
设置单个 episode 上限行动次数为 2000







在较早的 episode 中，agent 可能在某些 state 中往复转移，陷入死循环，故只能在达到行动上限后终止，reward 接近-20000；在多次更新参数后找到可行路径，并逐渐优化最优解。（如果没有使用归一化的 reward，如前所述 Agent 很可能只关注一条可行路径。实践中也确实如此，使用不同的参数，出现过行动次数锁定在 36 次、40 次、44 次等的情况）

3. 带有基础随机性的 Policy gradient：

在实验中，发现 Policy gradient 算法在参数受限的情况下，每次梯度上升对概率分布的影响可能会非常大，导致某些行动在之后被执行的概率接近 1，从而难以进行探索。为了让 Agent 能够探索不常选择的 action, 改变 Agent 的策略为：0.2random + 0.8softmax。但经过多次实验，并未得到全局最优解，推测为每次 random 带来负收益（相对于当前情况）的可能性更大，难以跳出局部最优。

算法分析：

1. 超参数方面

在此任务下，使用ε-greedy 策略的 Q-learning 算法很容易收敛到最优解，因为ε-greedy 引入的随机性可以使得 Agent 更轻易地从局部最优中跳出。与之相符，使用ε-greedy 策略的 Q-learning Agent 对环境超参数以及自身超参数的设置要求并不高；

但参数受限的 Policy Gradient Agent 在此任务下则略显乏力，由于只能使用 $10*10*2^7*4$ 大小的参数，每次梯度上升对概率分布的影响可能会非常大，导致某些行动在之后被执行的概率接近 1，从而难以进行探索。为解决此问题，只能采取在多个 episode 后才更新参数、使用折扣因子、归一化 reward、设置基础随机性等方法来克服此问题。在这种情况下，仍无法收敛到全局最优解，并且对超参数的选择十分苛刻：学习率过大或 batch_size 过小则容易导致一次梯度上升带来巨大的参数变化；学习率过小或 batch_size 过大则训练耗时过长，而且有时会陷入死循环长时间不能跳出，时间成本较大；折扣因子过小则不利于更新一个 episode 中较前状态的参数；折扣因子过大则可能导致 Agent 放弃探索直接出界或重复进入城市到达终止态，与此相关，状态转移 reward 的设置也十分苛刻……

实验过程中，在到达每座城市的 reward 相同的前提下，没能找到一组超参数使 Policy Gradient Agent 收敛到最优解。为达到全局最优，引入先验知识：

在环境构建过程中，仅给五座城市赋予值为 200 的 reward，目的在于引导 Policy Gradient Agent 先找到一条经过(0, 4), (2, 3), (3, 5), (7, 7), (9, 5)这五座城市

的最短路径，然后再从(9, 5)回到(0, 0)并在过程中经过剩下两座城市。

如此设置虽限制了 Policy Gradient Agent 收敛于局部最优的可能，但也同时使得 Agent 很难找到其他最优解。

2. 讨论

在此任务下，状态空间与行动空间较小，Q-learning 能够很快地收敛，但若状态空间与行动空间较大，Q-table 的内容较多，则需要更多次数的采样，也可能受到内存限制的影响；

Policy Gradient Agent 在此任务下由于参数量较小，受较早的探索、较大的 reward 影响大，且算法仅能保证收敛到局部最优，故对环境要求较为苛刻。但其具有处理连续空间动作的能力，而 Q-learning 无法做到。

总结：

在本次实践中，为得到使用最少步数单次遍历 Grid World 中所有城市并回到起点的 Agent，使用 Q-learning 与 Policy Gradient 算法，前者可以达到全局最优（34 步），而后者只达到了局部最优（36 步）。

在自主探索中，为 Agent 引入先验知识，最终基于两种算法的 Agent 在两个具有不同终止态的环境中都得到了最优解。

最后分析了这两种算法的表现。