机器学习 作业4

1 介绍

2 Bellman Equation

2.1 状态值函数定义

依据定义有

$$V^{\pi}(s) = E_{\pi}[G_t|S_t = s]$$

其中

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

2.2 贝尔曼期望方程

Bellman期望方程如下

$$V^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma V^{\pi}(s_{t+1})|S_t = s]$$

2.3 迭代策略评估

迭代过程中，更新公式为

$$v_{k+1}(s) = \sum_{a \in A} \pi_0(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$$

其中π_0为随机策略

进行一步更新的状态值计算如下

$$V_1^{\pi_0}(A) = \pi_0(ab|A)R_A^{ab} = -4$$
$$V_1^{\pi_0}(B) = \pi_0(ba|B)R_B^{ba} + \pi_0(bc|B)R_B^{bc} = 1/2 \times 1 + 1/2 \times 2 = 1.5$$
$$V_1^{\pi_0}(C) = \pi_0(ca|C)R_C^{ca} + \pi_0(cb|C)R_C^{cb} = 1/2 \times 8 + 1/2 \times 0 = 4$$

2.4 贪心法

贪心策略如下：

$$\pi_1(s) = \arg\max_{a \in A} q_{\pi_0}(s, a) = \arg\max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')$$

执行如下：

$$\pi_1(A) = ab$$

$$q_{\pi_0}(B, ba) = 1 + 0.5 \times 1 \times (-4) = -1$$
$$q_{\pi_0}(B, bc) = 2 + 0.5 \times 1 \times 4 = 4$$
$$\Rightarrow \pi_1(B) = bc$$

$$q_{\pi_0}(C, ca) = 8 + 0.5 \times (3/4 \times (-4) + 1/4 \times 4) = 7$$
$$q_{\pi_0}(C, cb) = 0 + 0.5 \times 1 \times 1.5 = 0.75$$
$$\pi_1(C) = ca$$

3 Monte Carlo & Temporal-Difference

3.1 首次访问蒙托卡洛

估计如下：

$$V(A) = Q(A, a) = \frac{(3 + (-3)) + (3 + 2 + (-4) + 4 + (-3))}{2} = 1$$
$$V(A) = Q(A, a) = \frac{((-2) + 3 + (-3)) + ((-4) + 4 + (-3))}{2} = -2.5$$

3.2 每次访问蒙托卡洛

估计如下：

- 初始值

$$V(A) = Q(A, a) = 0$$
$$V(B) = Q(B, b) = 0$$

- 轨迹（1）
  - 第一步

$$V(A) = Q(A, a) = 0$$
$$V(B) = Q(B, b) = 0 + 0.1 \times (-3 - 0) = -0.3$$

  - 第二步

$$V(A) = Q(A, a) = 0 + 0.1 \times (0 - 0) = 0$$
$$V(B) = Q(B, b) = -0.3$$

  - 第三步

$$V(A) = Q(A, a) = 0$$
$$V(B) = Q(B, b) = -0.3 + 0.1 \times (-2 - (-0.3)) = -0.47$$

- 轨迹（2）
  - 第一步

$$V(A) = Q(A, a) = 0$$
$$V(B) = Q(B, b) = -0.47 - (-3 - (-0.47)) = -0.732$$

  - 第二步

$$V(A) = Q(A, a) = 0 + 0.1 \times (1 - 0) = 0.1$$
$$V(B) = Q(B, b) = -0.732$$

  - 第三步

$$V(A) = Q(A, a) = 0.1$$
$$V(B) = Q(B, b) = -0.732 + 0.1 \times (-3 - (-0.732)) = -0.9588$$

  - 第四步

$$V(A) = Q(A, a) = 0.1 + 0.1 \times (-1 - 0.1) = -0.01$$
$$V(B) = Q(B, b) = -0.9588$$

  - 第五步

$$V(A) = Q(A, a) = -0.01 + 0.1 \times (2 - (-0.01)) = 0.191$$
$$V(B) = Q(B, b) = -0.9588$$

3.3 时序差分方法

估计如下:

- 初始值

$$V(A) = Q(A, a) = 0$$
$$V(B) = Q(B, b) = 0$$

- 轨迹（1）
  - 第一步

$$V(A) = Q(A, a) = 0$$
$$V(B) = Q(B, b) = 0 + 0.1 \times (-2 + 0 - 0) = -0.2$$

  - 第二步

$$V(A) = Q(A, a) = 0 + 0.1 \times (3 - 0.2 - 0) = 0.28$$
$$V(B) = Q(B, b) = -0.2$$

  - 第三步

$$V(A) = Q(A, a) = 0.28$$
$$V(B) = Q(B, b) = -0.2 + 0.1 \times (-3 + 0 - (-0.2)) = -0.48$$

- 轨迹（2）
  - 第一步

$$V(A) = Q(A, a) = 0.28 + 0.1 \times (3 + 0.28 - 0.28) = 0.58$$
$$V(B) = Q(B, b) = -0.48$$

  - 第二步

$$V(A) = Q(A, a) = 0.58 + 0.1 \times (2 - 0.48 - 0.58) = 0.674$$
$$V(B) = Q(B, b) = -0.48$$

  - 第三步

$$V(A) = Q(A, a) = 0.674$$
$$V(B) = Q(B, b) = -0.48 + 0.1 \times (-4 + 0.674 - (-0.48)) = -0.7646$$

  - 第四步

$$V(A) = Q(A, a) = 0.674 + 0.1 \times (4 - 0.7646 - 0.647) = 0.9328$$
$$V(B) = Q(B, b) = -0.7646$$

  - 第五步

$$V(A) = Q(A, a) = 0.9328$$
$$V(B) = Q(B, b) = -0.7646 + 0.1 \times (-3 + 0 - (-0.7646)) = -0.9881$$

4 Q-Learning & Sarsa

4.1 Q-Learning

- 见`algorithm.py`

4.2 Sarsa

- 见`algorithm.py`

4.3 learning rate

- 考察最终的平均奖励值（训练10000个Episode）

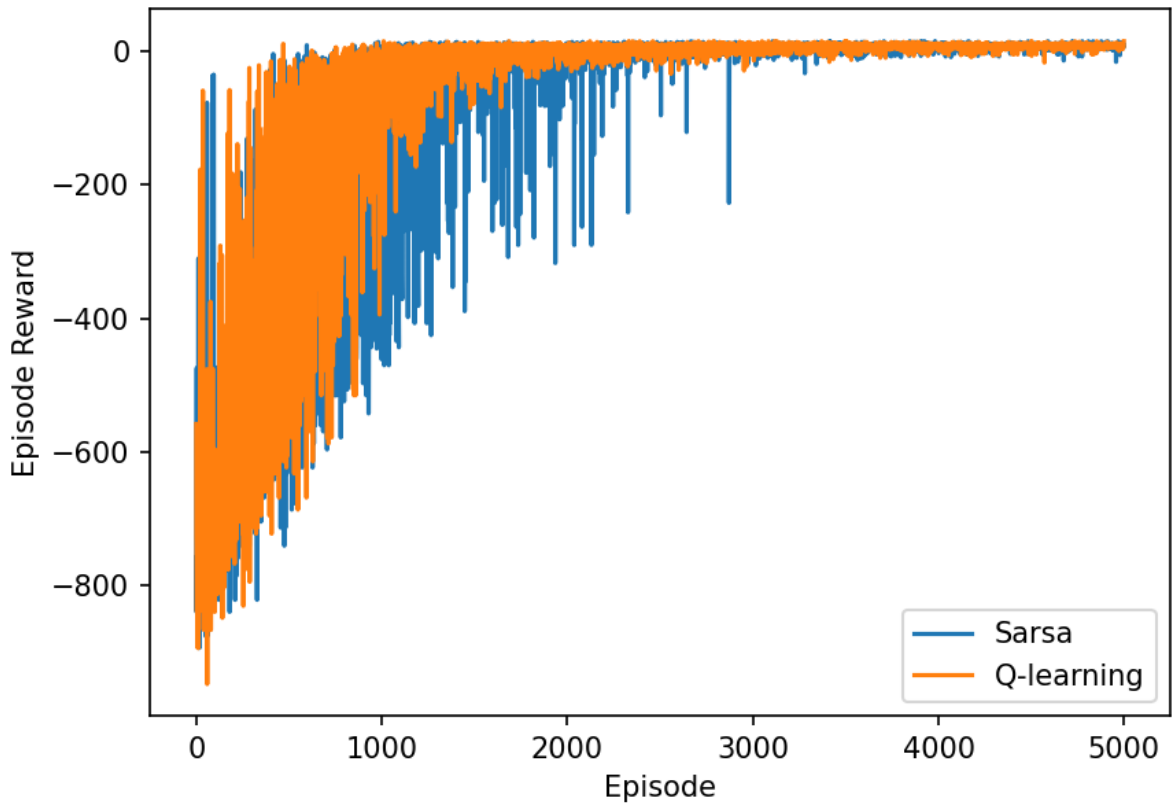
lr	Sarsa	Q-Learning
0.01	-105.141	-103.877
0.05	8.013	8.011
0.1	7.971	7.974
0.5	7.715	7.809
1	-123.719	7.916

- 可以看出，lr过小或者过大均不利于模型收敛：在lr过小的时候，模型参数更新太慢；lr过大时，模型容易发散
- 对比Sarsa与Q-Learning，在lr = 0.05/0.1/0.5时二者训练效果类似，Q-Learning效果略好一些；而在lr = 1下Sarsa无法收敛，Q-Learning则可以收敛。这可以说明Q-Learning的训练稳定性更高。

4.4 对比分析

- 训练5000个Episode，lr均取0.1

4.4.1 训练曲线



4.4.2 实验数据

	Sarsa	Q-Learning
Avg Reward	7.45	8.11
Avg Steps	13.55	12.89

4.4.3 分析

从训练结果可以看出：

- 与Sarsa相比，Q-Learning训练得到的模型具有更高的最终奖励值与更少的所需动作数，即训练效果更佳
- 与Sarsa相比，Q-Learning的收敛速度更快，且训练稳定性也更好；这是因为Sarsa是On-Policy的算法，在选取下一步时采取$\epsilon-$greedy；而Q-Learning是Off-Policy的，选取下一步时采取greedy，故收敛速度更快，也更稳定。

5 REINFORCE & AC

5.1 REINFORCE

- 见`./code/policy_gradient`

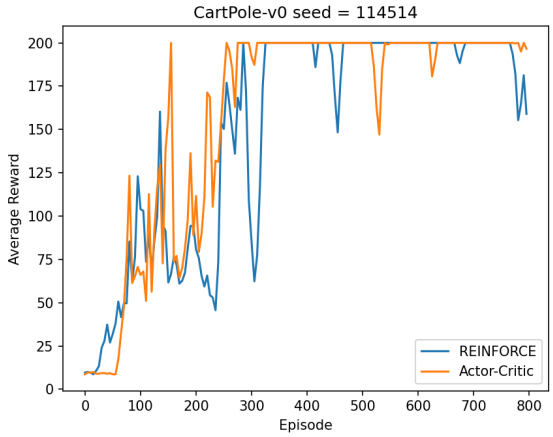
5.2 AC

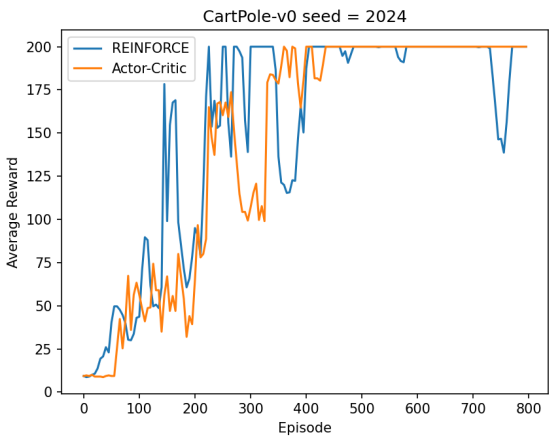
- 见`./code/policy_gradient`

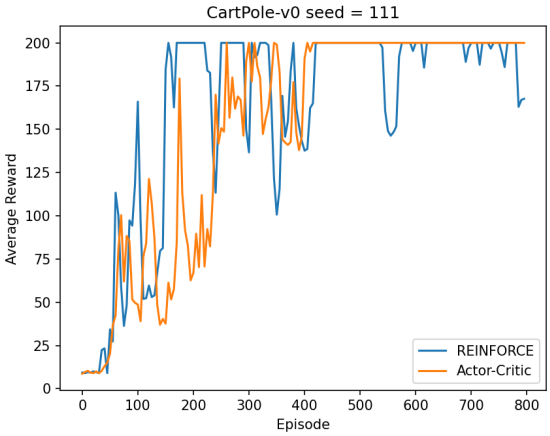
5.3 训练

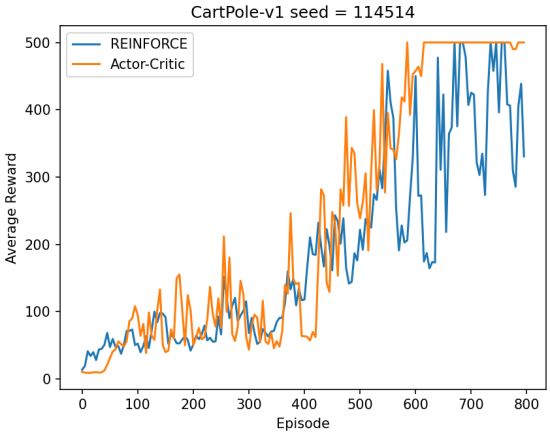
5.3.1 训练曲线

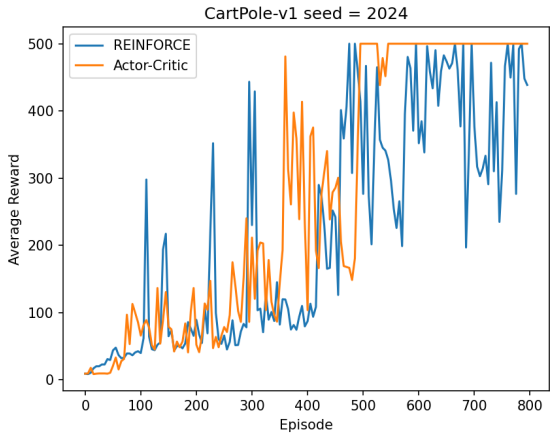
- 学习率为3e-3，分别在两个环境上(每个环境基于三个种子)的训练曲线如下：

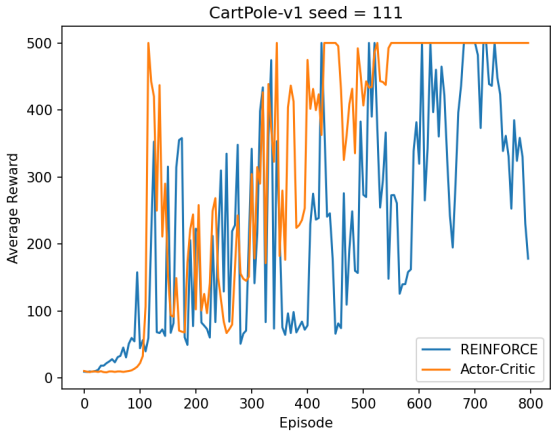












- 可以看到
  - CartPole-v0环境下，REINFORCE和AC均能稳定地获取最终的200奖励值

- CartPole-v1环境下，AC能稳定地获取最终的500奖励值，而REINFORCE也能获得最终的500奖励，但抖动较大，不够稳定

5.3.2 训练稳定性与收敛速率

- 稳定性
  - 训练曲线表明：AC算法比REINFORCE算法更稳定，训练中的抖动更小
  - 这是因为REINFORCE是基于蒙特卡洛的策略梯度算法，具有较大的方差；而AC使用价值函数估计来引导策略的更新，能够降低方差，提高稳定性
- 收敛速率
  - 训练曲线表明：在Cartpole v0环境下，REINFORCE与AC的收敛速度比较接近；而在Cartpole v1环境下，AC的收敛速度明显更快
  - 这是因为AC采用价值函数引导策略更新，与蒙特卡洛采样相比利用了更准确的信息（能够更准确地估计动作状态对的预期回报），同时具有更低的训练抖动，故而更快收敛