1. 项目介绍

本项目含3个python文件Env.py，Qlearn.py，PGAgent.py， 2个参数文件“Qlearn.pkl”，“PG.pkl”，以及两张图片“Qlearn收敛曲线.png”，“PG收敛曲线.png”。python文件的功能如下

Env.py: 游戏环境。项目游戏环境为在10x10的网格中解决旅行商问题。

Qlearn.py: 使用Qlearning算法学习的Agent，运行Qlearn.py后可以得到使用Qleanring算法后的最优路径，以及Q-table文件“Qlearn.pkl”, 学习过程中的回报曲线 “Qlearn收敛曲线.png”。

PGAgent.py: 使用PolicyGradient(策略梯度)算法进行学习的Agent，运行PGAgent.py后可以得到使用策略梯度算法后的最优路径，以及策略的参数文件“PG.pkl”, 学习过程中的回报曲线 “PG收敛曲线.png”。

1. 代码实现心得
2. 环境中奖励reward的设置

本项目中默认 reward = -1，如果发生越界或者两次访问同一城市则结束游戏; 如果解决了旅行商问题，则reward = 10000; 对于首次访问城市设置奖励reward = 20。

一开始我未对首次访问城市设置奖励，结果是Q-learning算法在300000 episodes下仍然不收敛，策略梯度算法则收敛到回报价值-1 ------ 对应agent的动作为向上走，直接越界。

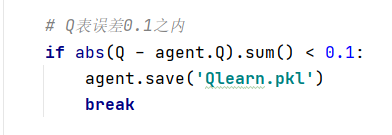
究其原因，是因为策略梯度算法在很难走到正确路径情况下，选择直接越界结束游戏以避免继续探索而累积负值。

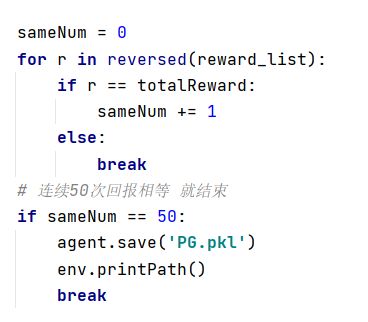
于是我对于首次访问城市单独设置奖励，从1到5到10到25。Q-learning 可以找到最优路径，且随着奖励值的增大收敛速度加快； 策略梯度算法在reward较小时如1,5,10等情况下仍无法找到符合正确路径。

1. Q-learning 行动选择标准

我设计的Q-learning 采取 贪心策略，其中。 由于Q-learning是异策略(off-policy)，因此在QlearnAgent.take\_action函数中设置了模式选择的参数evaluate = False， 当进行评估时，使用evaluate = True可以保证使用最优值。

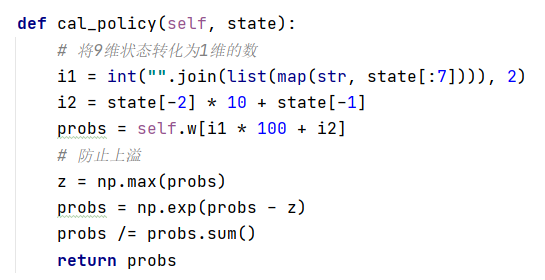
1. 收敛时刻的界定
2. learning与PolicyGradient的收敛界定有所区别。对于Q-learning，可以直接通过判定Q表值的变化大小进行判定是否收敛, 这样做不仅是因为Q-learning由于贪心策略训练时的回报价值不稳定，而且这样能够保证Q矩阵的精度。对于PolicyGradient，由于尝试通过参数矩阵变化率大小判定时无法收敛，而PolicyGradient收敛时策略价值回报能稳定，所以在判定PolicyGradient是否收敛时，将回报记录，连续若干次回报不变时即判定收敛。





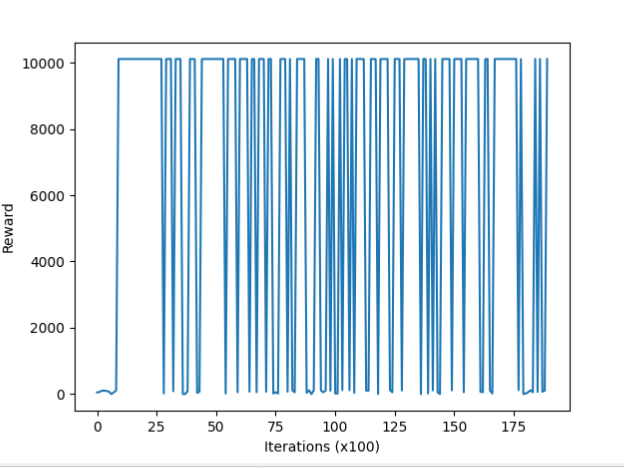
1. Softmax溢出问题

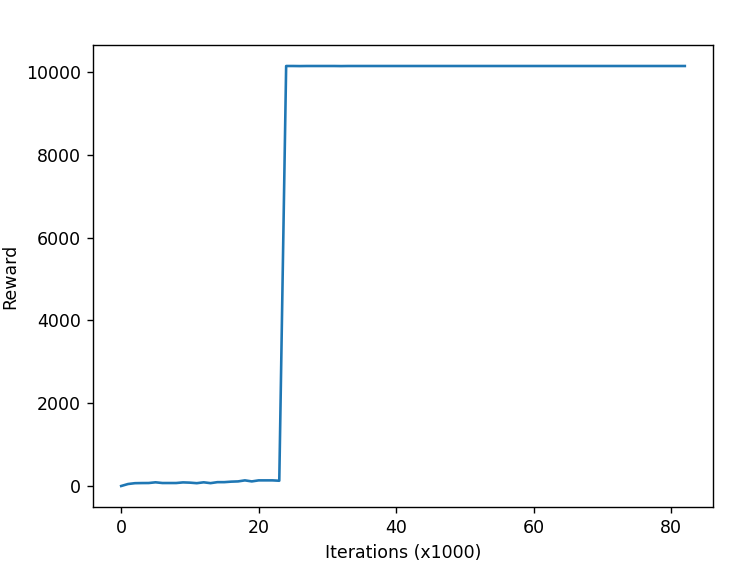
当x较大时，np.exp(x)会溢出，所以计算softmax时采取用向量减去最大值的方式，即np.exp(x-np.max(x))放置上溢。



1. 算法比较
2. Qlearning是异策略，PolicyGradient是同策略，因此Q-learning收敛速度更快。

下面是收敛曲线图展示





可以看到，由于Qlearning是异策略，Q表收敛时由于贪心策略仍会震荡，而PolicyGradient收敛时会稳定。收敛速度方面，Qleaning约为PolicyGradient的10倍。

1. PolicyGradient 会收敛到局部最优， 而Qlearning会收敛到全局最优

Qlearning 路径结果

路径长度 36

路径 (0, 0)->(0, 1)->(1, 1)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(1, 4)->(0, 4)->(1, 4)->(2, 4)->(2, 5)->(3, 5)->(3, 6)->(3, 7)->(4, 7)->(5, 7)->(6, 7)->(7, 7)->(8, 7)->(8, 6)->(8, 5)->(9, 5)->(8, 5)->(8, 4)->(8, 3)->(7, 3)->(7, 2)->(6, 2)->(6, 1)->(6, 0)->(5, 0)->(4, 0)->(3, 0)->(2, 0)->(1, 0)->(0, 0)

PolicyGradient 路径结果

路径长度 38

路径 (0, 0)->(1, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(1, 3)->(0, 3)->(0, 4)->(1, 4)->(1, 5)->(2, 5)->(3, 5)->(4, 5)->(4, 6)->(5, 6)->(6, 6)->(7, 6)->(7, 7)->(8, 7)->(8, 6)->(8, 5)->(9, 5)->(9, 4)->(9, 3)->(8, 3)->(7, 3)->(7, 2)->(6, 2)->(7, 2)->(7, 1)->(7, 0)->(6, 0)->(5, 0)->(4, 0)->(3, 0)->(2, 0)->(1, 0)->(0, 0)