人工智能基础

第 三 次 大 作 业

——强化学习

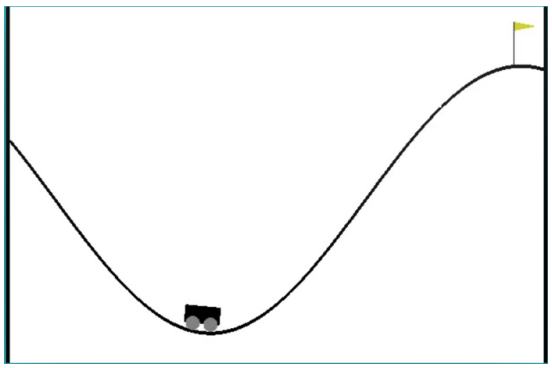
班级：自 71

姓名：屈晨迪

学号：2017010928

1 任务描述

- 必做：使用强化学习算法，解决MountainCar − v0；
  使用强化学习算法，解决MountainCarContinous − v0
- 选做：使用其他强化学习算法解决上述问题



2 环境配置

- python(3.6) + gym(0.15.4) + tensorflow(1.2.1) + keras(2.2.4)

3 算法设计

3.1 离散版本MountainCar − v0

（1）问题背景

现有一小车在两座山峰之间的谷底，小车动力有限，无法直接登上右侧山峰，需要借助动能和势能之间的转化才能到达目的地。在离散版本的MountainCar中，小车的行为（action）是离散的，有向左、向右、静止三个选项，每个状态（state）下小车的观测值包含位置（position）和速度（velocity）两个方面，小车从-0.4—-0.6 之间的任意位置开始运动，在一个 episode（200 步）内抵达 0.5 处即为成功，每走一步获得-1 的回报值。

Action	Push left	No push	Push right
Num	0	1	2

State	Max	Min
Position	0.6	−1.2
Velocity	0.07	−0.07

（2）Q-learning 算法

Q-learning 算法主要流程如图 1 所示，算法借助 Q 表完成，Q 表列出了在不同状态时采取不同行动的行动价值，需要在程序中不断更新。

在每个 episode 开始时初始化 Q 表，在每一步后更新 Q 表，Q-learning 中，行为策略采取ϵ −贪心策略，目标策略为贪心策略，从当前状态 S 下选择一个行动 A，有ϵ的概率随机选择，1 − ϵ的概率选取价值最大的行动，获得的 Q 值作为 Q 估计，获得回报 R，进入下一状

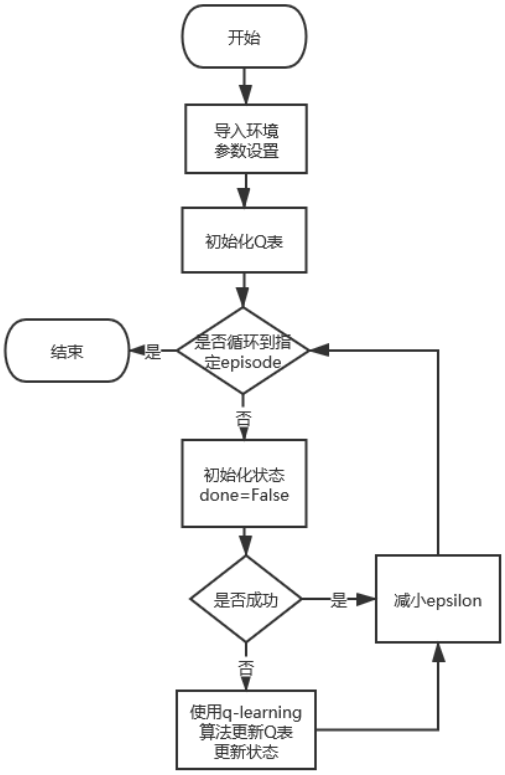
态S′；在S′状态下根据贪心策略选取最大价值的行动，该最大价值乘以回报因子γ，再加上之前的回报 R，作为 Q 现实；Q 现实减去 Q 估计，乘以学习率，作为误差更新 Q 表，其中学习率代表了从误差中学到新东西的能力。

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

图 1 Q-learning 算法

（3）程序流程

程序开始时，先导入环境，设置参数，如 episode 个数、学习率、回报因子等，初始化 Q 表，将小车状态中的位置和速度分别离散化为 m 和 n 份，即有 m*n 个状态，配合 3 种行为，Q 表包含 m*n*3 个元素；初始化$\epsilon = 1$，在后续学习过程中不断以epsilon_decay降低，即开始时勇于探索，之后相对保守；在每个 episode 循环开始时，使用 env.reset 重置环境，设置 done 为 False，在每一步完成后判断 done 是否为 True，若为 True 进入下一个 episode，不是则按照 Q-learning 的算法更新 Q 表，更新状态；一次 episode 完成后动态降低ϵ.



3.2 连续版本

MountainCarContinues与离散情况下相比，将行动改成了连续的，即不再是向左、向右或静止三种，而是一个范围区间，向左是负值，向右是正值；到达 0.45 的位置算做成功，每个 episode 成功抵达一次获得 100 的回报值，减去从开始到最终结束行动值的平方和。

针对此种情况，可以手动将行为离散化，本次作业中，将动作空间的（-1，1）分成了10段，类比离散情况，Q表有m*n*10个元素，其余步骤同3.1。

3.3 其他算法（选做）

（1）SARSA

SARSA的算法流程如图3所示，SARSA与Q-learning不同处在于，SARSA在行为策略和目标策略中均使用$\epsilon-$贪心，即在S′的状态上，按照ϵ的概率随机选取，$1-\epsilon$的概率取最大价值行动，用此行动（s′，a′）下的行动价值与当前（s，a）价值的误差来更新Q表，而这一估算的动作也是接下来一步执行的动作。

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

图2 SARSA算法

（2）SARSA(λ)

SARSA(λ)的算法流程如图4，Q-learning和SARSA都是单步更新，仅用到了上一步的信息误差来更新Q表，而成功抵达路径上的每一步都对最后的结果产生影响，SARSA(λ)即考虑了先前的多步，每往前一步重要性以λ的权值递减。在算法中加入E表来记录路径，目标策略的选择同SARSA，在S′状态上以$\epsilon-$贪心策略选择行为A′，以图4所示的公式更新Q表和E表，且每次均更新整张表。

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S',A') - Q(S,A)$
    $E(S,A) \leftarrow E(S,A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
      $E(s,a) \leftarrow \gamma\lambda E(s,a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

图3 SARSA(λ)算法

（3）DQN

Q-learning和SARSA均利用Q表进行实现，但现实情况中可能有大量的状态，表格法在时间和空间上难以实现，因此产生了函数逼近的方法。DQN是基于Q-learning的一种函数逼近方法，它利用神经网络逼近值函数，利用经验回放的策略训练强化学习的学习过程。在DQN中，有两个模型相同的神经网络，分别是用于动作值函数逼近的Model Prediction和用于计算TD目标的Model Target，其中Model Prediction每步均会更新，而Model Target则是隔一段时间更新一次；还有一个存放学习结果的“经验池”，用于经验回放，作为输入送进神经网络中进行训练，训练过程如图5所示。

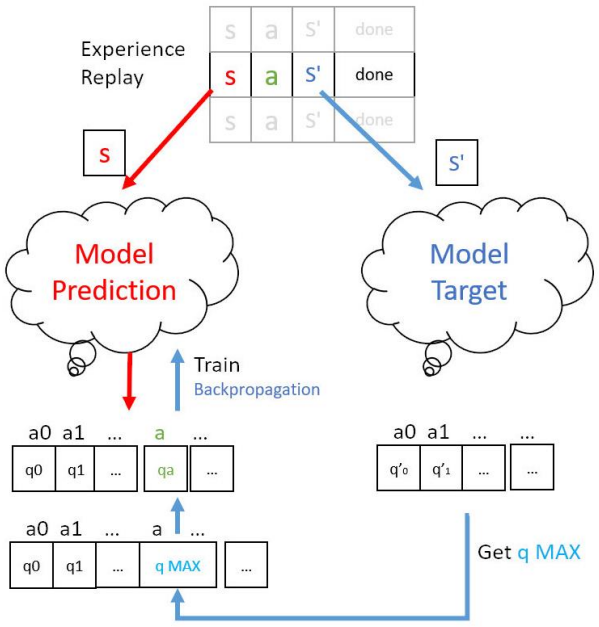


图 4 DQN 算法

在一个 episode 中，每步完成后，将当前状态 s，选择的行为 a，下一状态 s'，获得的回报 r，是否完成 done 组成一个五元组，放入“经验池”，当“经验池”满后，从中按一定大小分批次取出 batch，其中的 s 送入 Model Prediction，s'送入 Model Target，分别获得在 s 和 s'两个状态下所有行动的价值列表，从 q'中选出最大价值来更新 q，更新后的 q 再在 Model Prediction 中进行反向传播，更新网络参数；Model Target 定期与 Model Prediction 同步。

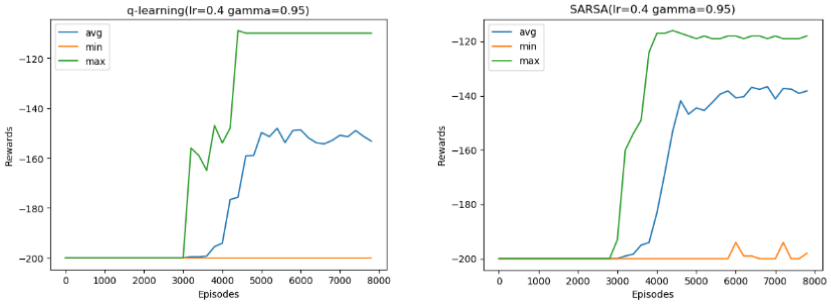
算法同样采样动态变化的ϵ策略，在开始时$\epsilon = 1$，随机选取动作，之后ϵ逐步递减，越来越多地根据网络预测出的 Q 表最大值采取行动。

在训练结束时，保存训练好的 model，测试时调用即可。

3.4 算法对比

在同等参数条件下，对比 Q-learning、SARSA 和SARSA(λ)如图 6 前三幅所示，每张曲线中横轴表示 episode，纵轴为回报值，每 200 个 episode 计算一次，其中绿线为最大值，蓝线为平均值，橙线为最小值。可以看到，三者均在 4000episode 左右开始收敛，SARSA 的收敛速度略快于 Q-learning；在最小值的表现上，$SARSA(\lambda) > SARSA > Q - learning$，最大值上 Q-learning 较优，平均回报 SARSA 较优，这是因为 Q-learning 采用离线学习，每次行动采取最大值，勇于冒险，而 SARSA 则是在线学习，选择较为保守的策略。

DQN 的表现则更为优秀，在 1000 个 episode 左右已呈现收敛趋势，且最大值要明显高于前三者，体现了函数逼近方法的优越性。



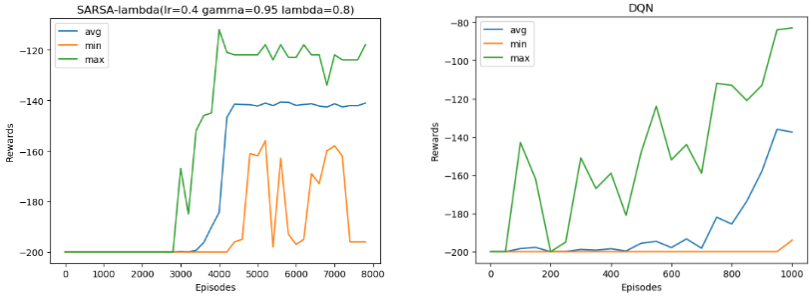


图 5 算法对比（lr = 0.4 γ = 0.95）

4 参数调整及优化[1]

• **学习率 lr：**更新 Q 表时，要将误差乘以一个系数α，这个系数即学习率，学习率的大小决定了算法从误差学习新知识的程度。在γ = 0.95时，分别取学习率为 0.7、0.4、0.3，从图 7 的对比来看，在学习率较高时收敛较快，但 lr=0.3 时回报值的整体表现优于 0.7，因此本次作业选取lr = 0.4.

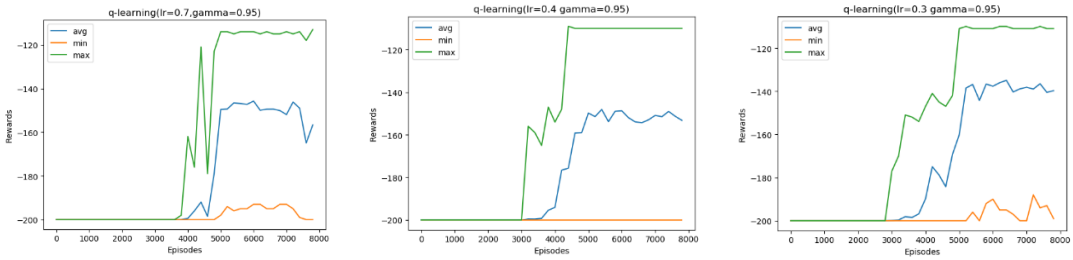


图 6 学习率对比

• **折扣因子γ：**

折扣因子(γ)体现了未来行动折算到当前的回报，在学习率lr = 0.4时，分别取折扣因子为 1、0.95、0.8，可以看到，在γ = 0.95时回报曲线表现最优，偏大或偏小均会出现不稳定波动[2]，本次作业选取γ = 0.95.

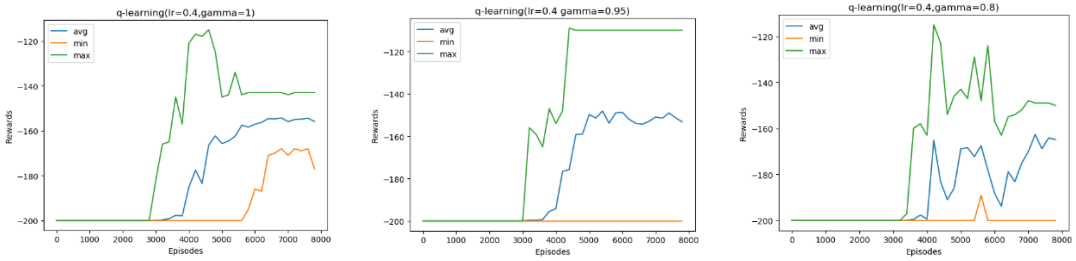


图 7 折扣因子对比

• **回报 reward：**

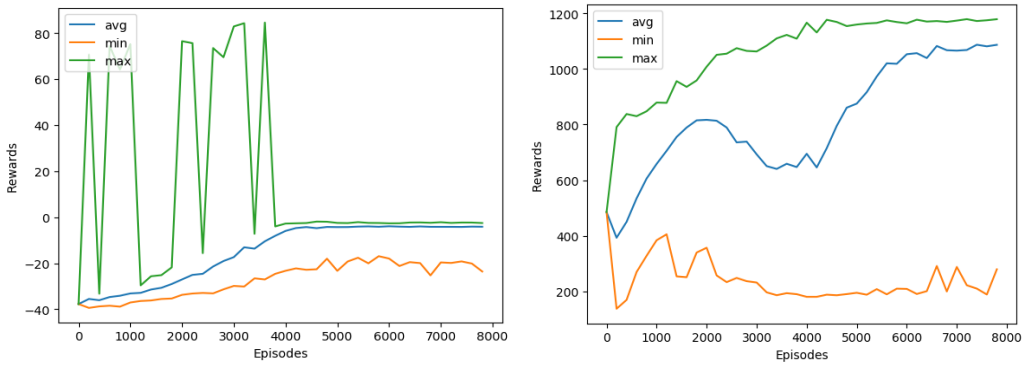
适当修改回报取值，会有效提升收敛速度。

在MountainCarContinues中，仅到达终点才会有 100 的回报，而每一步行动都会产生负

1 不做强调时，均采用 Q-learning 算法

2 不排除随机现象的可能

的回报，因此若没有在较少的 episode 内到达终点，小车最终会选择在静止在谷底。为了避免此种情况的发生，可以将每次的 reward 修改为reward = reward + abs(next_state[1] − abs(state[1])) ∗ β，其中state[1]为该状态下的速度，β为一个系数，即让小车尽量以较大的速度行动，尽快到达终点。



(a)原 reward　　(b)修改 reward

图 8 回报优化

• 离散化程度：

多次运行测试发现，对于MountainCarContinues来说，在一定裕度上加大行为空间离散程度可以提高算法性能；而状态空间离散程度对性能没有太大影响。

5 总结与反思

本次大作业为强化学习方面的程序设计，通过这次作业我加深了对 Q-learning 和 SARSA 两类算法的理解，用实践巩固了理论知识，并从程序结果更清晰地体会到了不同算法之间的差异，新学习了 SARSA(λ)和函数逼近的 DQN 算法，也是对之前监督学习的一个回顾，同时，我掌握并锻炼了 python 语言的编写，熟悉了 pycharm 平台的相关操作，有了上大作业的经验，这次配置环境、安装数据包等都熟练了很多。这是人工智能课程最后一个作业了，在这门课上真的收获很多，感谢老师和助教本学期以来的帮助！

6 参考文献和资料

[1] https://github.com/openai/gym/wiki/Environments
[2] https://github.com/openai/gym/wiki/Leaderboard
[3]zht007githubhttps://github.com/zht007/tensorflowpractice/tree/master/10_Renforcement_Learning_Moutain_Car
[4]莫凡 python　https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/4-1-A-DQN/
[5]《深入浅出强化学习 原理入门》郭宪，方勇纯著