强化学习：作业四

李颖 MG20370022

2020 年 12 月 30 日

1 作业内容

实验内容：实现 Model-based Q-learning 算法。

游戏环境与目标: 本次作业使用和作业 2 一样的环境为网格世界 (grid-world)，玩家可以通过选择动作来移动人物，走到出口。唯一的区别在于输出的状态包括了额外的一维特征，表示 agent 是否拿到了钥匙。agent 需要先拿到钥匙（坐标在（0,7)），然后走到出口才算通关。

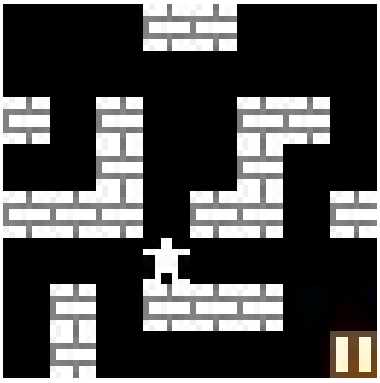

图 1: gridworld 环境

问题形式化：该问题设定可以被形式化为一个马尔可夫决策过程：

1. 状态集合 (S): 一个三维数组 (12,84,84)，表示 4 帧彩色图像 (3,84,84) 的复合。

2. 动作集合 (A): 6 个离散的动作，分别表示为 $\{0, 1, ..., 5\}$

3. 奖励 (R)：玩家在游戏中击中乒乓球有对应的奖励。
4. 转移概率 (P)：在每个位置，采取行动后都以概率 1 转移到对应的下一位置。
5. 折扣系数: γ

2 实现过程

在本次实验中需要实现 model-based Q-learning。对于 model-based 的算法来说，agent 除了需要对当前状态下的最佳动作进行预测（这部分通过 Q-learning 算法完成），还需要对环境的预测（也就是 model 的部分）。model 的部分在采样轨迹的时候需要存储：

1. 在当前状态 s 下采取动作 a，环境会发生什么样的变化，也即下一个状态 s' 是什么。

2. 在当前状态 s 下采取动作 a，环境给当前状态的即时奖励 r。

因为通过实验二和三，我们已经了解到 table-based 的 Q-learning 算法和 DQN 算法，与之类似，本次实验中，我们对 model-based 算法中也有基于 table 和基于神经网络两种算法。

2.1 实验一

在第一个实验中，需要实现 Dyna-Q 算法，Dyna-Q 算法的伪代码如下。由于在这个游戏场景中，环境的转移是确定性的，model 可以用 table 来进行记录和更新。model 中实现了三个接口：

1. store_transition: 存储 transition(s, a)

2. sample_state: 在出现过的 states 中采样

3. sample_action: 在该 state 上出现过的 actions 中采样

Algorithm 1 Dyna-Q algorithm

Initialize action-value function Q with random weights
Initialize Model(s,a) for all $s \in S$ and $a \in A$
s = env.reset()
while True do
  while not done do
    Select action a using $\epsilon - greedy$ strategy
    s', r, done $\leftarrow$ env.step(a)
    Update Q: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + (1 - done) * \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    Update Model: Model$(s,a) \leftarrow s'$
    $s \leftarrow s'$
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  end while
  for $t = 1$ to n do
    Select previously observed state s_m randomly
    Select action a_m previously taken in s_m
    $s'_m \leftarrow$ Model(s_m, a_m)
    $r_m \leftarrow R(s_m, a_m)$
    Update Q: $Q(s_m,a_m) \leftarrow Q(s_m,a_m) + \alpha[r_m + (1 - done) * \gamma \max_{a'} Q(s'_m, a'_m) - Q(s_m,a_m)]$
  end for
end while

(1) 尝试不同的 n，比如 n=0,500,1000 等，展示算法运行的性能图，也即每个 episode 的平均得分和时间随消耗样本的关系图，如图 2。
(2) 为了 n^* 经验性的估计，表示若 $n > n^*$ 算法消耗的样本量不再明显下降，于是从 0 开始，以 20 为步长增加 n，每次都记录**收敛时所消耗的样本量**。绘制收敛消耗样本量随 n 的关系图，如图 10

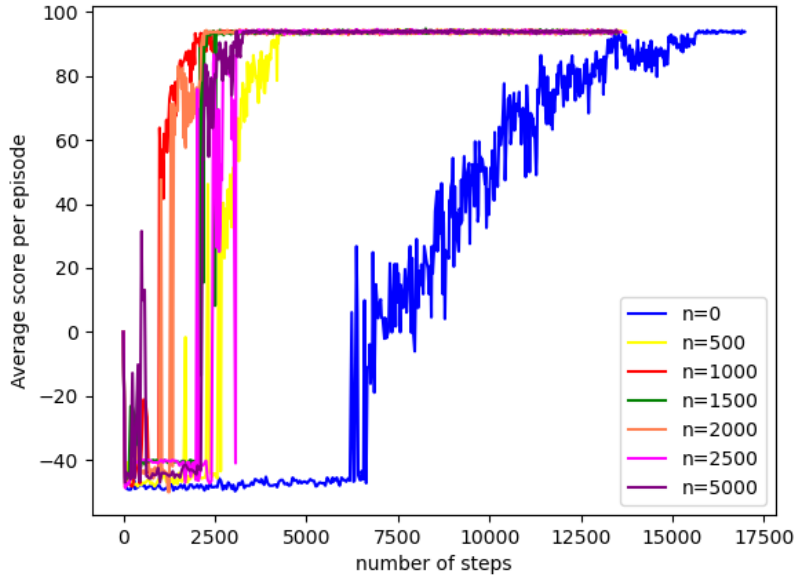

图 2: 每 episode 平均得分随消耗步数的关系

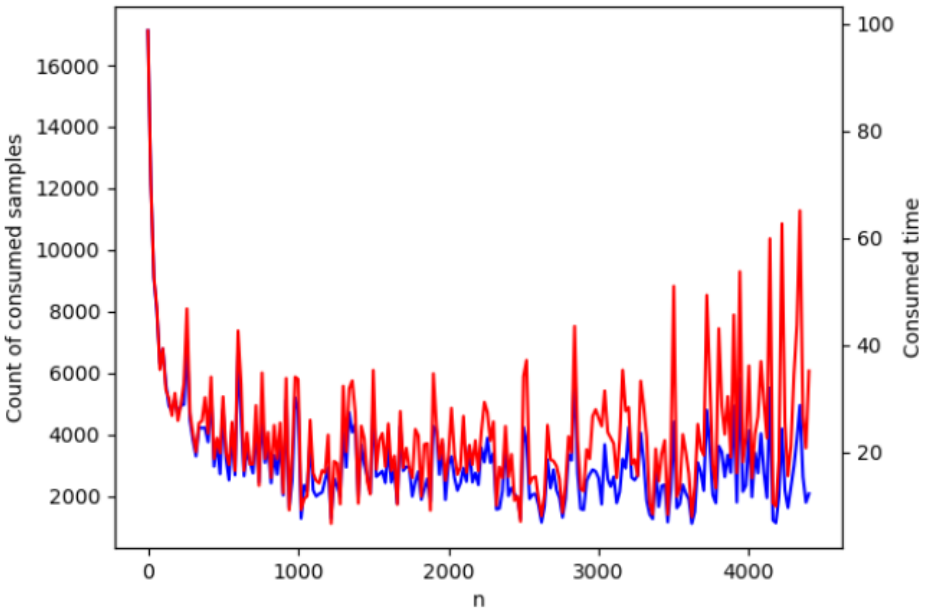

图 3: 收敛步数、时间随 n 的变化

对以上两张图进行分析：

1. 对比 n=0(model-free) 和 n=500 的收敛情况，n=0 时需要 17000 左右的样本才能收敛，而 n=50 时只需要 3000 左右，由此可知 model-based 的收敛速度要比 model-free 显著增加。

2. 从图 10发现，随着 n 不断增大，收敛时消耗样本不断减小，之后趋于一个值之后不变，在这个例子中，最少的消耗样本在 2000 左右。

3. 从图 10发现，随着 n 不断增大，消耗时间一开始减小，和消耗样本的变化类似，但随后，在消耗样本几乎保持不变的情况下，所用时间一路走高。

4. 从图 10发现，当 n 到达 2000 的时候，再增加 n 收敛消耗样本数不会有显著的下降，这个在图 2中也很明显，n 在 0、500、1000、1500、2000 收敛速度是逐步加快的，但再调大到 2500 甚至于 5000，他们的收敛是慢于 2000 的。所以在这个实验中，我们得到 n^* 的粗略估计是 1200 左右。

2.2 实验二

首先实现基于神经网络的 model-based 算法，伪代码如下所述。此处有四个超参数需要尝试：

1. n：采样的轨迹条数

2. start_planning: 开始使用 Model-based 提高样本利用率的点

3. h：一条轨迹执行的长度

4. m：转移训练的频率

Algorithm 2 Model-based neural network algorithm

Initialize action-value function Q with random weights
Initialize Model(s,a) for all $s \in S$ and $a \in A$
s = env.reset()
for t=1 to T do
  while not done do
    Select action a using $\epsilon - greedy$ strategy
    s', r, done $\leftarrow$ env.step(a)
    Update Q: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + (1 - done) * \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    Update Model: Model$(s,a) \leftarrow s'$
    $s \leftarrow s'$
  end while
  for i=1 to m do
    model.train_transition()
  end for
  if t > start_planning then
    for j=1 to n do
      Select previously observed state s_m randomly
      for k=1 to h do
        Select action a_m using $\epsilon - greedy$ strategy
        $s'_m \leftarrow$ Model(s_m, a_m)
        $r_m \leftarrow R(s_m, a_m)$
        Update Q: $Q(s_m,a_m) \leftarrow Q(s_m,a_m) + \alpha[r_m + (1 - done) * \gamma \max_{a'} Q(s'_m, a'_m) - Q(s_m,a_m)]$
        if done then
          break
        end if
      end for
    end for
  end if
end for

在调参开始之前，先观测一下两种方法在 model-free 的情形，如图 4、5，table-based 在 17000 左右个样本收敛，NN-based 在 18500 个样本左右收敛，NN-based 比 table-based 所消耗样本更多。

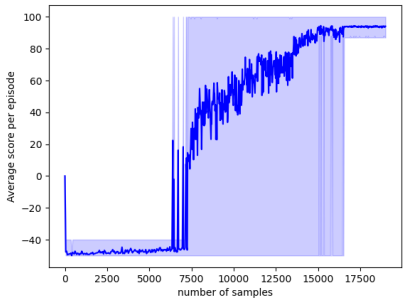

图 4: Table-based Model-free

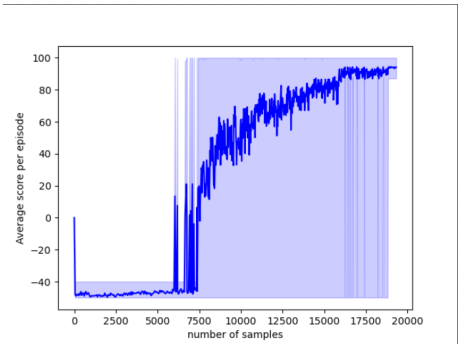

图 5: NN-based Model-free

在接下来的调参中随意尝试，有几点发现：

(1) 对于给定一组 start_planning, h 和 m 之后 n 对收敛 s 速度仍然有上述影响，也即随着 n 增大，收敛步数趋于稳定。比如图 6。在 n 较小的时候是否 n 越大收敛速度越快呢？这并不是绝对的，还跟其他参数的取值有关。

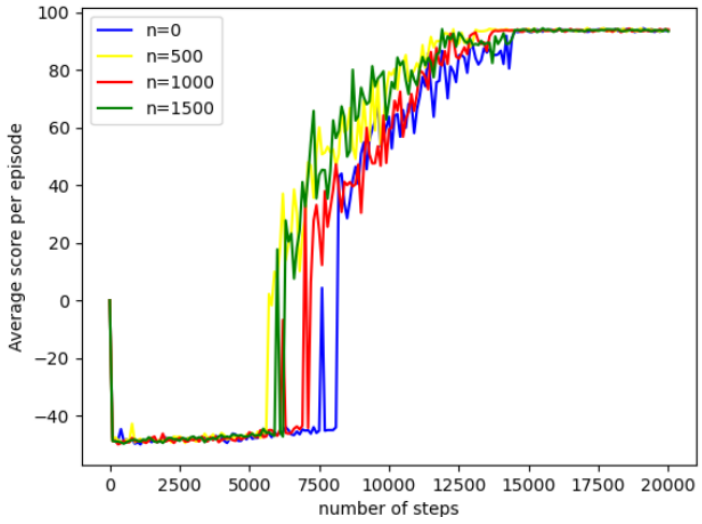

图 6: start_planning=20, h=10, m=500 情况下 n 的收敛消耗步数图

(2) start_planning 不能设置得太大，在迭代中越靠后使用 model-based 的方法收敛越慢。这点也从图 7 中可以得到佐证，从 s=10 开始，s 越大，同等条件下反而收敛越慢。

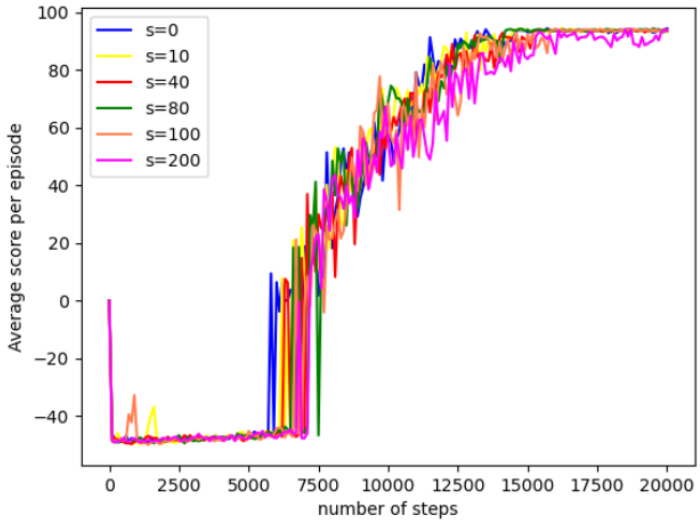

图 7: h=10, n=1, m=1000 情况下 start_planning 的收敛消耗步数图

最后通过尝试一些参数，在所有尝试中我得到最优的参数组合是 h=2, start_planning=20, m=1000,n=200，如图 8 所示，在 3500 步左右收敛。

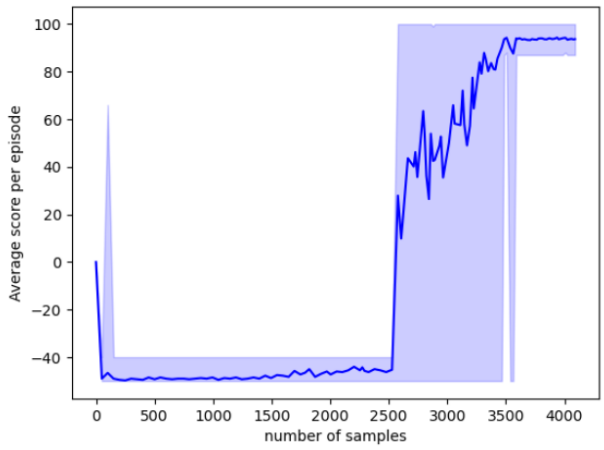

图 8: start_planning=20，h=2, n=200, m=1000

改进一：改进 Model 的学习流程，强化对系稀疏/奖励变化相关的数据的学习。再次尝试不同的参数组合，得到最优组合是 h=2,start_planning=20, m=1000,n=200，如图 9 所示，在 2600 步左右收敛。对算法进行优化后，最优的收敛步数要比没优化之前的少，且最优参数组合变化与之前得到的相近。

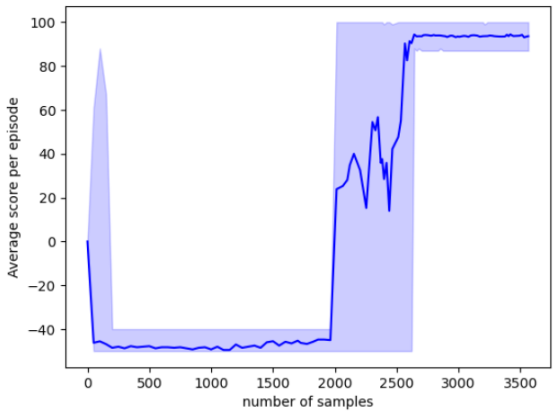

图 9: start_planning=20，h=2, n=100, m=1000

改进二：对策略的学习过程做额外的约束, 最优组合是 h=2,start_planning=20, m=1000,n=200，如图 9 所示，在 3200 步左右收敛。

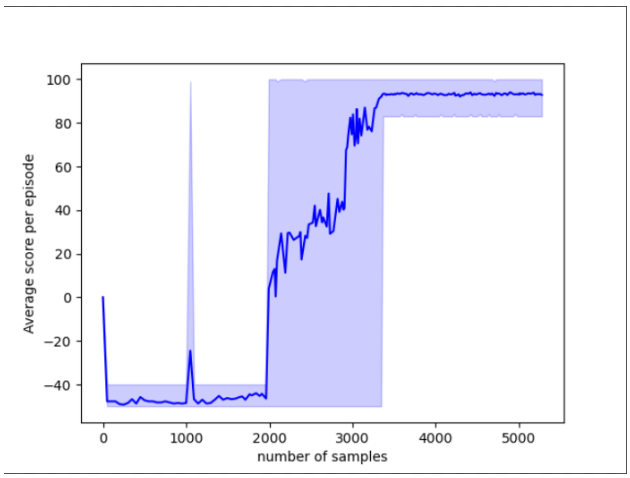

图 10: start_planning=20，h=2, n=200, m=1000

3 复现方式

1. 复现实验一: 在文件夹下将 line 68 的 dyna 设置为 True，并在运行时设置 n，例如 python main.py --n 1000.

2. 复现实验二: 在文件夹下将 line 68 的 dyna 设置为 False，并在运行时设置 s,h,m,n，例如 python main.py --s 10 --h 10 --m 1000 --n 100.

4 实验分析

1. 根据上面实验，讨论不同模型学习方式 (table 和 neural network)，不同参数对实验结果的影响和背后的原因，从而分析 model-based 的算法性能因素有哪里？

在上述实验一二中，已经对两种学习方式中参数的影响进行了分析。在此处做一个总结，对于 table-based 的学习来说，不断调高 n 的值会让收敛步数迅速减少，直到一个最优值不再减小，而收敛时间则是先随 n 减小再不断增大。对于基于 neural network 的学习来说，start_planning 和 h 的值不宜设得过大，尤其是 start_planning。n 和 m 对收敛速度的影响更大些。由此可见采样的轨迹条数和转移训练的频率对 model-based 算法性能影响比较大。

2. DQN 中的 replay buffer 设置和 Dyna-Q 实验，这两者有何关系？

DQN 中专门设置 replay buffer 和 Dyna-Q 实验是类似的手段去完成相同的目的。DQN 中启动 replay buffer 后能适度减缓 model-free 方法样本利用率不高的问题，而 Dyna-Q 中对样本的多次使用也是 model-based 方法样本利用率高的原因。除此之外，这样的操作还能减小样本之间的相关性，对于基于神经网络的学习来说，性能更好。

5 小结

在本次实验中，我实现了 table-based 和 NN-based 两种 model-based 的 Q-learning 算法，在网格环境中进行测试和调参，探究并分析了不同参数对 model-based 算法性能的影响。