

强化学习：作业三

傅浩敏 MG20370012

2020年12月28日

1 作业内容

本实验需要在gridworld环境中实现model-based Q-learning算法并评估算法性能。本实验的gradworld是一个8*8的迷宫，agent需要拿到位于(0,7)处的钥匙后到达终点才算通关，因此环境输出了状态除了当前坐标外还包含了是否获得钥匙这一额外特征。本次实验由三部分组成，首先我们实现Dyna-Q算法，并且调整超参数以尽可能达到算法可提升的性能极限。第二部分是使用神经网络来预测环境model，同样通过调整超参数来获得最佳的性能表现。第三部分是改进model学习过程以应对稀疏奖励的问题，并且分析这些改进对算法带来了哪些变化。最后需要分析不同的学习方式和超参数对model-based算法性能的影响，并且讨论产生这些影响的可能的原因。

2 实现过程

2.1 算法描述

首先需要在 `algo.py` 中实现Q-learning Agent。在本实验中我直接使用HW2中实现过的Q-learning算法，我以表格的形式记录Q值并且动态地向Q表中添加新的状态-动作对。

对于普通的Dyna-Q算法，我也采用表格的形式记录状态-动作-新状态之间的转移关系。每当在环境中采样后，需要将采样到的转移关系存在转移表中。我们需要同时使用采样到的转移和转移表中储存的转移来更新Q表。

在第二个实验中，我们使用神经网络来预测状态-动作-新状态之间的转移关系，在环境中采样后都要把采样到的转移关系存储在buffer中。每经过若干轮迭代，我们从buffer中抽取若干样本来训练我们的model。同样的，我们需要同时使用采样到的转移关系和神经网络模型预测的转移关系来更新Q表。

在第三个实验中，我们对model的学习过程进行了一些改进。首先，我们在采样的同时记录了那些获得钥匙的关键转移。在训练神经网络模型时，固定这些样本在训练样本中的比例，从而保证模型始终能够对“是否获得钥匙”做出正确的预测。第二个改进是在Q表中的值超过某个定义的上下界时进行截断。

2.2 代码实现

Algorithm 1 为Dyna-Q的伪代码：

Algorithm 1 Dyna-Q

```

1: Initial  $Q_\pi$  and Model. Given reward function  $R$ , and terminal function  $D$ .
   Given learning rate  $\alpha$ , discount rate  $\gamma$ , and repeat number  $n$ .
2: for  $1, 2, \dots$  do
3:    $s = env.reset()$ 
4:   for  $1, \dots, steps$  do
5:     epsilon-greedy:  $a = \pi_\epsilon(s)$ 
6:     interact with environment:  $s', r, done = env.step(a)$ 
7:     update Q:  $Q_\pi(s, a) += \alpha(r + \gamma(1 - done) \max_{a'} Q_\pi(s', a') - Q_\pi(s, a))$ 
8:     update Model:  $Model(s, a) \leftarrow r, s'$ 
9:      $s = s'$ 
10:    if  $done$  then
11:       $s = env.reset()$ 
12:    end if
13:  end for
14:  for  $1, 2, \dots, n$  do
15:     $s_m, a_m, s'_m = \text{random sample from observed transitions}$ 
16:    get reward and done:  $r_m = R(s_m, a_m, s'_m), d_m = D(s_m, a_m, s'_m)$ 
17:    update Q:  $Q_\pi(s_m, a_m) += \alpha(r_m + \gamma(1 - d_m) \max_{a'_m} Q_\pi(s'_m, a'_m) - Q_\pi(s_m, a_m))$ 
18:  end for
19: end for

```

Algorithm 2 为实验3的伪代码。可以通过删除与sensitive buffer和clip函数相关的语句来获得实验2的伪代码，因此实验2的代码不再单独列出：

Algorithm 2 Neural Network Algorithm with Improvement

```
1: Initial  $Q_\pi$ ,  $Model$ , transition buffer  $Buffer$ , and sensitive buffer  $Buffer_s$ .  
   Given reward function  $R$ , terminal function  $D$ , and clip function  $clip$ .  
   Given learning rate  $\alpha$ , discount rate  $\gamma$ , repeat time  $n$ , learning period  $m$ , step  
   length  $h$ , and start time  $start\_planning$ .  
2: for  $1, 2, \dots$  do  
3:    $s = env.reset()$   
4:   for  $1, \dots, steps$  do  
5:     epsilon-greedy:  $a = \pi_\epsilon(s)$   
6:     interact with environment:  $s', r, done = env.step(a)$   
7:     update Q:  $Q_\pi(s, a) + = \alpha(r + \gamma(1 - done) \max_{a'} Q_\pi(s', a') - Q_\pi(s, a))$   
8:     clip Q:  $Q_\pi(s, a) = clip(Q_\pi(s, a), -100, 100)$   
9:     update buffer:  $Buffer.append(s, a, r, s')$   
10:    if  $s[-1] \neq s'[-1]$  then  
11:      update sensitive buffer:  $Buffer_s.append(s, a, r, s')$   
12:    end if  
13:     $s = s'$   
14:    if  $done$  then  
15:       $s = env.reset()$   
16:    end if  
17:  end for  
18:  for  $1, 2, \dots, m$  do  
19:     $transition\_list = \text{sample transition from } Buffer \text{ and } Buffer_s$   
20:    train model:  $Model.train\_transition(transition\_list)$   
21:  end for  
22:  if  $iter > start\_planning$  then  
23:    for  $1, 2, \dots, n$  do  
24:       $s_m = \text{random sample from observed transitions}$   
25:      for  $1, 2, \dots, h$  do  
26:        epsilon-greedy:  $a_m = \pi_\epsilon(s_m)$   
27:        calculate next state:  $s'_m = Model(s_m, a_m)$   
28:        get reward and done:  $r_m = R(s_m, a_m, s'_m), d_m = D(s_m, a_m, s'_m)$   
29:        update Q:  $Q_\pi(s_m, a_m) + = \alpha(r_m + \gamma(1 - d_m) \max_{a'_m} Q_\pi(s'_m, a'_m) -$   
           $Q_\pi(s_m, a_m))$   
30:        clip Q:  $Q_\pi(s_m, a_m) = clip(Q_\pi(s_m, a_m), -100, 100)$   
31:        if  $d_m$  then  
32:           $break$   
33:        end if  
34:      end for  
35:    end for  
36:  end if  
37: end for
```

3 复现方式

在主文件夹下运行 `python main.py`.

3.1 参数介绍

实验中涉及的主要参数见表1, “*” 表示该参数的值会在实验中改变。

Table 1: 主要参数

名称	默认值	说明
<code>learning_rate</code>	0.2	学习率 α
<code>gamma</code>	0.99	折扣系数 γ
<code>epsilon</code>	0.2	探索概率 ϵ
<code>frames</code>	10000	训练总帧数
<code>n</code>	*	<code>model</code> 中采样次数
<code>m</code>	*	<code>model</code> 训练次数
<code>h</code>	*	<code>model</code> 采样时的步长
<code>start_planning</code>	*	开始从 <code>model</code> 中采样的帧数

4 实验效果

4.1 实验一

从`n=0`开始, 每次将`n`增大600, 找到大致的最优值范围, 然后进行细微调整, 找到的经验性 n^* 的估计大致在1500左右。实验中的其他主要参数设置为: `m=0`, `h=1`, `start_planning=0`。具体实验结果如下:

Table 2: n 与算法收敛时间和消耗样本量的大致关系

n 的取值	收敛所需样本量	收敛时间
0	9200	20s
600	1800	05s
1200	1600	05s
1400	1300	05s
1500	800	03s
1800	1000	04s
2400	900	04s
3000	1100	06s

部分实验中得分与使用样本量的关系如下：

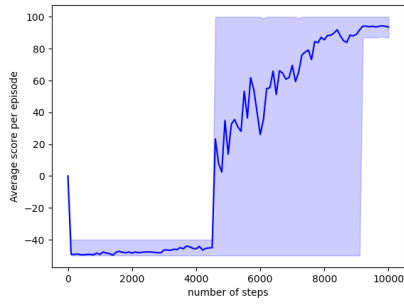


Figure 1: $n=0$

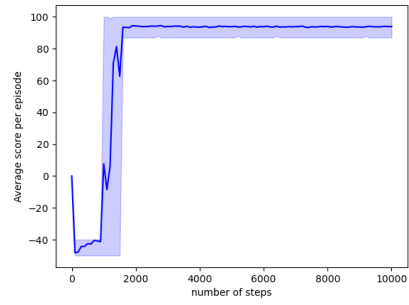


Figure 2: $n=1200$

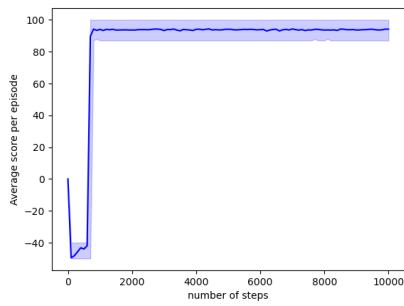


Figure 3: $n=1500$

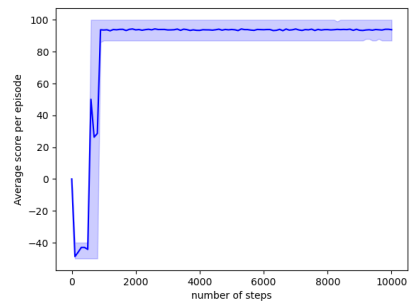


Figure 4: $n=2400$

4.2 实验二

我所找到的最好的参数组合是： $n=300$ ， $m=1000$ ， $h=1$ ， $start_planning=5$ 。在此参数设置下，模型大约能稳定地在5000步以内收敛，并且收敛所需时间在一分钟之内。同时，在最优参数设置的基础上，我分别改变了其他单一参数并进行了对比实验。最优参数和对比参数的实验结果如下：

4.3 实验三

在对算法进行改进后我找到的最优参数组合变为：。同时模型的收敛步数也提升到了xxx步之内，收敛时间大约为xxxx。在此参数设置下，改进后模型的性能曲线如下：

5 问题讨论

1. 根据上面实验，试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，从而分析影响model-based的算法的性能的因素由哪些？

使用table学习模型可以直接学习到正确的转移，因此影响算法性能的主要参数是在模型中循环采样的次数 n ，在 n 足够大时我们就能获得最好的收敛步数，在此基础上我们仅需要考虑算法耗时。但使用神经网络学习model时，对环境的学习会存在一定的误差，而model的误差会严重影响Q表的更新效率。在这种情况下，提升模型中的采样次数 n 只会让我们的策略更加不稳定。因此，我们必须首先保证神经网络能够较好的学习model，再利用由model产生的样本。

一个简单的方法是增加model的训练次数 m ，增强神经网络对环境的拟合度，但是过大的 m 会严重影响算法的效率。同时，注意到最初的几步迭代中，神经网络很有可能无法正确地学习到状态转移，因此可以通过延迟在model中采样的时间 h 来提升算法稳定性。我们还可以为环境设计针对性的改进以提升model的训练性能。比如在本实验中，改变钥匙特征的转移样本数量较少但十分重要，如果直接采样神经网络可能会遗忘这些关键样本，因此加强对这一部分样本的学习能够提升model的性能。同时我们也可以人为约束Q值的上下界来稳定Q表的训练过程。

在结合model和policy的训练过程时，我认为应该首先调整model的训练次数 m 和开始在model中的采样时间 h ，以免将model中的误差引入Q表的更新过程。在此基础上再调整 n 值获得性能和效率的平衡，从而获得最佳的参数设置。

2. 回顾HW3的DQN中的replay buffer设置和前面的Dyna-Q实验，你觉得这两者有什么联系？

DQN中的replay buffer和Dyna-Q都存储了遇到过的状态-动作-状态转移，并且都使用了存储的转移来训练模型。不同点在于replay buffer的目的是重放转移来加快模型的训练过程；而Dyna-Q则试图建模真实的环境，并用建模好的环境来训练模型。这使得在replay buffer中采样时，往往会采样到频繁出现的转移，而在Dyna-Q中，我们可以较为平均得采样到环境中不同状态的转移，从而更好地利用出现过的转移。

6 小结

6.1 关于算法本身

在这次实验中，我发现在能够训练好model的情况下，使用model based的方法可以极大地提高我们模型收敛时所需的样本数。但是在使用神经网络构建model时，model的训练效果对参数非常敏感，我们需要多次尝试才能较好地模拟环境，在多数情况下效果甚至不如model free的方法。此外，虽然model based的方法所需要的样本数量较少，但是由于训练model存在一定的时间成本，最后算法收敛的时间相较于model free的方法不一定能够有所缩短，在使用神经网络学习model时更是如此。因此，我认为是否使用model based的方法，取决对学习model的成本和与环境交互的成本之间权衡。当算法与环境的交互成本不高，但环境较为复杂的时候，model free的方法也许是更好的选择。当然，在某些情况下学习环境的model可以为我们带来额外的好处，比如我们可以迁移我们的model来训练其他agent，但这不是本实验探讨的主要内容。

6.2 关于实验过程

本实验的主要困难在于参数调整的过程。