

实验 3. 强化学习实践

MF1733034, 李青坪, lqp19940918@163.com

2018 年 1 月 2 日

综述

与传统的监督、无监督学习不同，强化学习的过程需要与学习环境进行交互，利用环境反馈信息进行学习。因此在本次实验中，我们将脱离从训练数据集中获取学习模型的学习框架，转向与动态的学习环境打交道。本次实验，我们将从学习环境的安装、常用强化学习算法的实现，强化学习算法改进这些方面完整的体会一次强化学习研究的过程。

本次实验采用的强化学习环境为 OpenAI 开源的 Gym。Gym 中集成了很多强化学习的实验环境，并为测试强化学习提供了很多学习任务，如 CartPole 和 MountainCar 等。在 DQN 算法实现中会用到深度学习工具，本次实验采用 PyTorch，PyTorch 提供了神经网络的搭建模型，方便我们实现 DQN 算法。

实验二.

Gym 为我们提供了一些基础学习任务，但是在这些任务里面，状态空间都是连续的，而 Q-learning 算法的状态空间是离散的。所以本次实验的首要任务就是对状态空间离散化，通过 Gym 文档中提供的状态-动作空间的查询命令，我们知道 CartPole、MountainCar 和 Acrobot 三个任务的状态-动作空间分别为 (4, 2), (2, 3), (6, 3)，我们需要对这些空间进行离散化。本次实验采用的方法是将连续的空间分成等间隔的 n 份 (n 为每个维度上划分的块数)。具体算法如下：

设 $observation[i]$ 表示第 i 维状态观测值， $STATE_THRESHOLD[i]$ 表示第 i 维状态的下界， $bound_width$ 表示第 i 维状态的观测值的数据宽度， $STATE_NUM[i]$ 表示自定义的第 i 维状态的离散状态数目。则对于每一个观测值，求出其在离散化的状态空间中对应的下标即可。

$$state_index = (observation[i] - STATE_THRESHOLD[i]) / bound_width * STATE_NUM[i]$$

通过状态空间离散化，即将连续的观测空间状态转换到离散的空间上来，我们即可使用 Q-learning 算法进行强化学习。

Q-learning 算法实现的思路如下：

- 首先初始化一张 $Q(s, a)$ 表，里面的每一行表示在状态 s ，执行动作 a 所拥有的价值。

- 对每一个片段执行以下操作：
- 初始化状态 s
- 对每一个片段里的每一个步骤执行以下操作：
- 在状态 s 使用 ϵ 贪心策略选出动作 a
- 执行动作 a ，得到奖励 r 和下一个状态 s'
- 更新 Q 值： $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
- $s \leftarrow s'$

算法实现思路比较简单，而且代码也不是特别复杂，想要实验可以成功的完成任务，最重要的还是参数的调整。在本次实验中，探索率 ϵ 初始设置为 1.0，最小值设置为 0.01，按照指数级别递减；学习率 α 初始设置为 0.5，最小值设置为 0.1，按照指数级别递减；衰减值 γ 设置为 0.99 恒定不变。

针对 CartPole 任务，最大观测长度设置为 20000，片段数设置为 1000（考虑到接近收敛的时候，基本都能保证杆子不倒下来，故设置在连续 200 次成功解决 CartPole 问题后就认为已经收敛，则终止片段循环），成功解决 CartPole 问题的步骤数设置为 199。每个状态维度的可操作状态数设置为 (1, 1, 6, 5)，表示只考虑杆子的角度和角速度，忽略小车的位置和速度。虽然损失两个状态，但对解决 CartPole 问题不会产生影响。

针对 MountainCar 任务，最大观测长度设置为 2000，片段数设置为 1000（考虑到接近收敛的时候，基本都能保证小车到达旗杆处，故设置在连续 200 次成功解决 MountainCar 问题后就认为已经收敛，则终止片段循环）。每个状态维度的可操作状态数设置为 (8, 8)，表示小车的位置和速度各有 8 个可操作状态。

针对 Acrobot 任务，最大观测长度设置为 2000，片段数设置为 1000（考虑到接近收敛的时候，基本都能保证关节达到系统设定的高点，故设置在连续 200 次成功解决 Acrobot 问题后就认为已经收敛，则终止片段循环）。每个状态维度的可操作状态数设置为 (1, 1, 1, 1, 10, 10)，表示只考虑两个杆子的角速度，忽略它们的角度。虽然损失两个状态，但对解决 Acrobot 问题不会产生影响。

对三个强化学习任务分别进行 5 次实验，得到多条轨迹上 reward 的均值和标准差：

CartPole:

-	1	2	3	4	5
均值	7111.80	7376.67	7636.14	7219.83	7426.34
标准差	6009.24	5946.63	6190.02	6025.62	6020.15

MountainCar:

-	1	2	3	4	5
均值	-163.78	-178.46	-185.27	-161.381	-212.69
标准差	22.57	129.53	7.98	276.15	32.82

Acrobot:

-	1	2	3	4	5
均值	-144.01	-103.94	-105.65	-88.99	-261.28
标准差	29.49	193.17	122.38	63.22	271.97

实验三.

由于 Q-learning 需要穷举状态-动作对, 所以面对连续状态-动作空间, Q-learning 无法适用。但是值函数近似 Q-learning 方法能够解决此类问题。若将值函数近似模型选取为一个深度网络, 即为 Deep Q-learning (DQN) 算法。DQN 算法基于 Q-learning, 算法框架和 Q-learning 完全相同, 不同之处在于 Q 函数的表示和更新方式上。DQN 中 Q 函数用一个深度网络表示, 记为 $Q(s, a; \theta)$, 其中 s 表示状态, a 表示执行的动作, θ 表示网络参数。DQN 的目的是得到一个 Q 函数网络, 对于一组状态动作对 (s, a) , 使得 Q 函数网络的输出 $Q(s, a; \theta)$ 等于 (s, a) 下真实的 Q 值。具体到深度网络表示的 Q 函数模型上, Q 函数的学习问题转化为了如何学习一组深度网络参数 θ 使网络输出近似于真实的 Q 值。

本次实验中, 我们采用 PyTorch 提供的神经网络模型建立我们所需的 Q-target 网络和 Q-evaluation 网络, 输入层神经元数目为可观测的状态数目; 隐藏层有一层, 这里设置隐藏层有 50 个神经元; 输出层的神经元个数为动作的数目。新建一个 DQN 类, 表示我们将要实现的算法, 私有属性 memory 表示大小为 N 的重现内存, 这里设置 $N=2000$, 设置损失函数为 PyTorch 提供的 `nn.MSELoss()` 函数, 即平方损失函数, 公式如下:

$$loss(x, y) = \frac{1}{N} \sum_{i=1}^N |x - y|^2$$

使用 `torch.optim.Adam()` 作为学习的优化器, 它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率, 参数为 Q-evaluation 网络, 学习率 ALPHA 的初始值设为 0.008; 探索率 ϵ 初始设置为 1, 最小值设置为 0.01, 随片段 episode 的增加呈指数级别减小; 衰减值 γ 设置为 0.99 恒定不变; Q-target 更新频率设置为每学习 100 次更新一次; 算法中每次从 memory 中抽取的 MINI_BATCH 数目为 32。

DQN 算法实现思路如下:

- 初始化大小为 $(N \times (\text{状态数} + 2))$ 的 memory 矩阵。
- 新建 Q-target 网络和 Q-evaluation 网络, 即完成状态动作函数的初始化。
- 对每一个片段执行以下操作:
- 初始化状态 s
- 对每一个片段里的每一个步骤执行以下操作:
- 在状态 s 使用 ϵ 贪心策略选出动作 a_t
- 执行动作 a_t , 得到奖励 r_t 和下一个状态 s_{t+1}
- 将 (s_t, a_t, r_t, s_{t+1}) 存入 memory 中
- 从 memory 中随机抽取 MINI_BATCH 个数据
- 对抽取到的每个数据计算 $y_i = r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta)$
- 将 y_j 和 $Q(s_j, a; \theta)$ 带入损失函数, 修正 θ

实验四.

...