强化学习
作业二

502022370062, 张含笑, zhanghx@lamda.nju.edu.cn

2022 年 11 月 7 日

1 作业内容

在 gridworld 环境中实现 Q-learning 算法。

2 实现过程

2.1 定义超参数

`args.env_mode`	选择 Q-learning 算法运行的环境, default=2
`args.num_stacks`	agent 每做一次动作间隔的帧数, default=4
`args.num_step`	每次更新 Agent 所需采集的步数, default=100
`args.num_frames`	一共需要采集的游戏画面的帧数, default=100000
`args.lr`	学习率, 对应 Q-learning 算法中的 α, default=0.01
`args.gamma`	折扣系数, 对应 Q-learning 算法中的 γ, default=0.8
`args.log_interval`	每次评估模型的间隔轮数, default = 1

2.2 实现 Q-learning 类

首先实现基本的 Q-learning 类。具体而言, Q-learning 的更新式为

$$Q^{new}(s_t, a_t) \leftarrow Q^{old}(s_t, a_t) + \alpha \left(r_t + \gamma \cdot \max_a Q^{old}(s_{t+1}, a) - Q^{old}(s_t, a_t) \right), \tag{2.1}$$

即

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) Q^{old}(s_t, a_t) + \alpha \left(r_t + \gamma \cdot \max_a Q^{old}(s_{t+1}, a) \right). \tag{2.2}$$

定义 myQAgent 类，用于维护 Q-table. Q-table 定义在某一个时刻的状态 s 下，采取动作 a 能够获得收益的期望，环境会根据 agent 的动作反馈相应的 reward 奖赏，初始状态下全部为 0.

select_action(ob): 用于选择动作。根据当前状态 `ob`, 查询 Q-table 对应状态下 Q 值（期望收益）最大的动作，并返回。

update(ob, action, reward, ob_next): 根据 Q-learning 更新规则2.1, 更新 Q-table.

最后在 `main.py` 中设定 myQAgent 类的实例化和 $\epsilon-$greedy 中 ϵ 的线性衰减:

$$\epsilon = 0.8 - \frac{i}{\texttt{num\_update}}, \tag{2.3}$$

其中 i 为当前更新次数，`num_update` 为总更新次数，具体定义为 $\frac{\texttt{args.num\_frames}}{\texttt{args.num\_steps}}$.

3 复现方式

由于 `gym-minigrid` 的名称已经被废弃, 改用 `minigrid`, 因此如报错可以在当前环境下运行 `pip install minigrid`.

在主文件夹下运行 `python main.py --env-mode 1` 可以查看算法在第一种环境下的运行结果；`python main.py --env-mode 2` 可以查看算法在第二种环境下的运行结果。

4 实验效果

两种环境下累计奖励和样本训练量之间的关系如图1所示, 其中左侧子图是 Q-learning 算法在第一种环境模式下的表现，右侧子图是 Q-learning 算法在第二种环境模式下的表现；横坐标为模型更新次数，纵坐标为累计奖励。

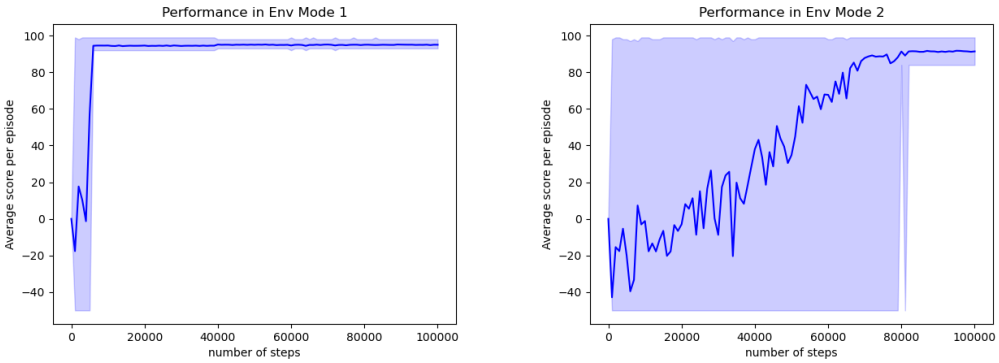

图 1: 在不同环境下的实验效果图

5 小结

在这次实验中，我发现在较简单的 gridworld 环境中基本的 Q-learning 算法就能取得不错的性能。这种确定性环境可以作为更大规模算法的验证性或说明性任务使用，相较复杂环境有更好的可解释性。