强化学习：作业一

linze.H

2024 年 10 月 11 日

1 作业内容

在“蒙特祖马的复仇”环境中实现 Dagger 算法。

2 实现过程

1. 搭建环境：经过我一个上午不断的尝试和错误，我发现只有 Python 版本为 3.7，gym 版本在 0.21 以下，并且可能需要对环境做出一些修改后（如将 Python 库 importlib-metadata 降至 5 级以下），并安装 gym 独立的相关依赖，才可以运行框架代码. 具体见 ./requirements.txt
2. 重构代码：原框架代码并不是原算法 Dagger 的实现，只有 Dagger 的一层循环，因此需要重构代码，并构建基础设施. 我
   - 将所有 `train()` `val()` `main()` 等过程封装为函数，打包为模块 `train.py`
   - 并实现了对 `Env` `Agent` 的抽象，打包为模块 `Dagger`
   - 加入对专家的抽象接口 `Expert`，存储在 `Dagger` 中
   - 实现常用方法，如代码运行时动态绘画 `obs` 的函数 `Expert.draw()`；从终端读按键的函数 `Expert.label()` （为 `prompt_getkey(prompt: str)` 的封装）；保存、读取数据 `save_result(obs: list[np.ndarray], labels: list[int], conf)`，`load_data(dir_path: str)`
   - 将超参封装在 `conf` 中，并与 `args` 合并，可以通过 Python 和命令行调试超参.
3. 选择分类器：我先进行了几轮游戏，收集了一些数据，存储在 `./played_dir/` 中，并使用脚本 `./test_clf.py`（将 `obs` 的像素拼接为一维向量）进行不同分类器的效果检验，结果见表 1 和文件 clf_score.txt. 最终我选择决策树作为分类器.

相关改进

在超参 `conf.T` 为 `0xFFFFFFFF` 时，此局游戏 (epoch)，由专家决定什么时候结束；并且专家可以拒绝标注一个样本. 这些改进大大减小了专家数据标注的负担.

在本轮游戏 (epoch) 结束后，agent 开始更新前，专家可以在 `imgs` 文件夹 (`conf.save_dir`) 中重新查看已标注的数据，提高数据质量.

5 cross validation / performance / classifier	test score / fit time				
SVM(linear kernal)	0.72/184.49	0.75/192.74	0.77/208.16	0.77/943.09	0.81/95.42
SGD SVM(linear kernal)	0.58/100.11	0.75/83.80	0.68/98.73	0.76/74.03	0.74/127.71
SGD Logistic Regression	0.67/100.35	0.74/102.31	0.80/118.58	0.77/95.82	0.71/94.74
Decision Tree	0.73/7.12	0.79/7.84	0.81/7.50	0.78/8.05	0.81/8.57
RandomForest	0.78/18.54	0.84/18.64	0.86/20.03	0.80/17.20	0.87/17.41
Multinomial Naive Byasian	0.64/4.14	0.66/4.02	0.69/4.02	0.68/4.02	0.74/4.05

表 1: 分类器性能

3 复现方式

在主文件夹下运行 `python main.py` 或 `./main.py` (on *nix) 运行程序. 在终端中输入上下左右按键标注 agent 行走数据, 输入空格 ( ) 标注 agent 跳跃数据, 输入逗号 (`,`) 标注 agent 左跳跃数据, 输入句号 (`.`) 标注 agent 右跳跃数据, 输入斜线 (`/`) 标注 agent 禁止不动数据, 输入 `m` 拒绝标注此数据, 输入 `n` 结束提前本轮游戏, 进入下一轮.

在主文件夹下运行 `python main.py --play-game True --save-dir played_dir` 进行游戏游玩.

4 实验效果

4.1 结果

见图 1。

此图的超参设置为: env_name="MontezumaRevengeNoFrameskip-v0", num_fpstep=8, T=0xFFFFFFFF(expert dicide when to halt the game), num_steps=100000, epochs=30, val_T=2000, val_draw=True, epsilon=0.05, log_interval=2, save_img=True, save_dir="imgs", save_interval=10, play_game=False, draw_method="plt", log_file="log", agent=DtAgent, agent_init_dir="",

4.2 累计奖励、访问专家次数和样本训练量之间的关系

从图中可以发现, 随着样本训练量的增长, 访问专家的次数越来越多, 基本呈现线性增长. 因为框架代码中, agent 生成的数据都会被返回给专家标注, 因此呈现线性关系; 而累计奖励会不稳定地上升, 因为随着样本训练量的增长, 分类器的准确率越来越大; 但随着见过的情况越来越多, 分类器会做出权衡, 即舍弃一些游戏最开始的样本的准确率, 以应对新情况, 因此会出现累计得分不稳定的情况.

5 思考题

Q: 在玩游戏的过程中标注数据与 Dagger 算法中的标注数据方式有何不同? 这个不同会带来哪些影响?

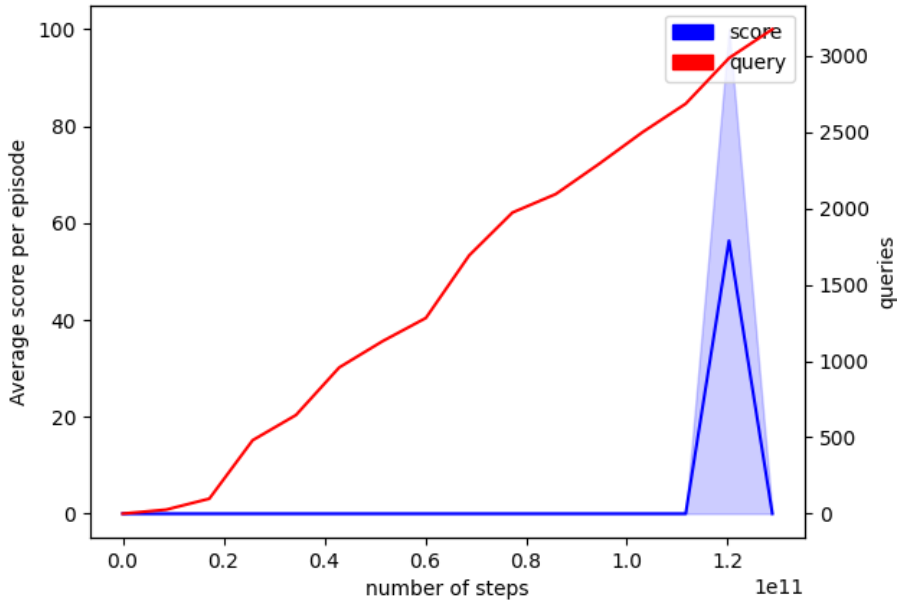

图 1: 决策树 Agent 性能

A: 玩游戏过程中标注数据，数据的分布和专家策略生成的数据分布一致；Dagger 算法中标注数据的分布和 Agent 生成数据的分布一致.

6 小结

在这次实验中，我发现...

1. 基础设施固然重要，权衡利弊也很重要，巧妙的设计更重要
2. Dagger 真的难训练，非常累人，非常消耗专家