强化学习：作业一

李丹悦 502024370013

October 14, 2024

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

2.1 专家数据采集与处理

根据 DAgger 算法的流程可知，我们需要在模型决策的过程中实时新增标注专家数据，但是经过实操发现由于模型训练所需步数较多，这样的训练方式需要玩家时刻关注游戏实况并根据画面逐次手动输入，过于耗时耗力，所以我们退而求其次，提前通过玩游戏的过程进行了专家知识库的建立，这样在算法运行过程中就不用再手动标注数据，而可以直接查询现成数据集。同时，为了尽可能还原 DAgger 算法的特性，我们不仅标注了常见的“得分路径”，也对一些较为罕见、极端的游戏状态进行了标注 (如agent卡在角落或走过头的情况)。为了加快后续的模型训练，我们参考https://github.com/zzhuncle/NJUAI-Master-Courses中的做法将$210 \times 160 \times 3$的RGB图压缩为 128×128的灰度图后再存入训练集。我们运行了多轮游戏，最终标注了2510条数据，样本与标签一并存入 `data_set.zip`。

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Figure 1: DAgger算法

2.2 DAgger算法的实现

我们通过 `main.py` 中的以下代码实现 DAgger 算法 (为突出算法主体对原代码进行了简化后展示)，其中专家问询部分我们使用提前收集的2510条样本训练了一个 k 值为 1 的 KNN 模型，这样就可以找到数据集中与当前游戏画面最相似的画面并给出对应的专家意见。这样的设置避免了遇到未标注数据时专家不知道给出何种建议的情况[1]。专家问询概率 `epsilon` 的初始值为 1，以每轮 0.95 倍的速度衰减。我们在agent的模型部分也选用了KNN，并同样将 k 值设置为 1，具体实现可见 `Dagger.py`。

```
epsilon = 1
for step in range(args.num_steps):
    new_label = expert_model.predict(obs)
    train_data_set['data'].append(obs)
    train_data_set['label'].append(new_label)
    if i == 0 or np.random.rand() < epsilon:
        action = action_map[new_label]
        query_cnt += 1
```

[1]此处感谢张少丹同学的建议。

```
    else:
        action_label = agent.select_action(obs)
        action = action_map[action_label]
    obs = obs_next
    if done:
        envs.reset()
epsilon *= 0.95
agent.update(train_data_set['data'],
                        train_data_set['label'])
```

2.3 参数设置

在进行实验时，我们对参数进行如下设置:

参数名称	值
–num-stacks	8
–num-steps	300
–num-frames	3000
–test-steps	1000
–log-interval	1

Table 1: 实验参数

3 复现方式

在主文件夹下运行 `python main.py`，完整代码可见https://github.com/bioLydia/RL-HW1。

4 实验效果

如图 2所示，由于我们设置的衰减速率较慢，所以从图像上来看，访问专家次数基本上正比于算法运行步数；随着访问专家次数的增加，agent 的表现也出现上升趋势，在算法运行 300 步后，就可以吃到第一个钥匙并到达第二块地图，在 1200 步后可以吃到第二个钥匙，在 1500 步后可以吃到第三个钥匙并到达第四块地图，证明了 DAgger 算法的有效性；但是同时我们也注意到，agent 性能在测试中不够稳定、波动较大，可能原因是样本训练量不足，尽管我们在数据标注中已经关注到了一些特殊状态，但在算法实际运行过程中，agent 经常走到一些出其不意的地点，专家没有能给出一个较好的决策建议，导致了 agent 的表现不佳。

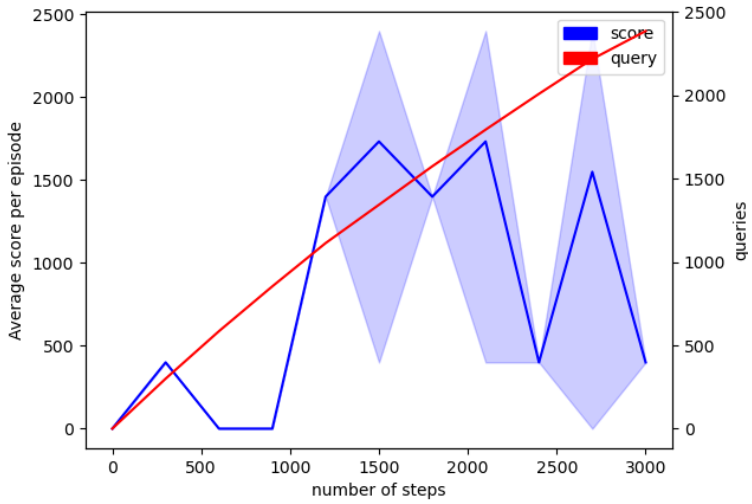

Figure 2: 实验效果图

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?

A：在玩游戏的过程中标注数据，并在标注完后利用这些数据训练模型，那么样本量往往就是固定的，我们无法在训练中新增标注数据。同时，真人玩家玩游戏时都具有较强的主观性，因此在玩游戏的过程中标注数据很容易导致数据集中于熟悉的游戏路径，而忽略了那些边缘的、难以遇到的状态。相比之下，DAgger算法能够“在线”实时增加标注数据，并且由于模型在过程中不断探索和产生错误，DAgger算法标注的数据会更加多样化，能够逐渐纠正专家的决策偏差，有助于模型在不同状态下学到更加稳健的策略。

6 小结

在这次实验中，我们成功在“蒙特祖马的复仇”环境中实现了 DAgger 算法。实验结果表明，随着算法的迭代，agent 的表现逐步提高，能够完成关键任务（如收集钥匙和到达新区域）。然而实验结果也暴露了一些问题，如样本不足导致的模型不稳定。总体而言，DAgger 算法在该任务中证明了其有效性，同时也启发了我们思考如何优化样本标注与模型训练过程。