AAI4160 Homework 1: Imitation Learning

Jungwoo Ahn
2021147584

1 Introduction

The goal of this assignment is to make you familiar with **(1) Yonsei AI Teaching Cluster (VESSL AI)**, **(2) Reinforcement Learning Environments**, and **(3) Imitation Learning Algorithms**, including behavioral cloning (BC) and DAgger. Here is the link to this homework report template in Overleaf.

2 (Optional) VESSL AI

If you have your own GPUs, you can skip this section.

If you need a GPU for this assignment, we provide VESSL AI (AI-LEC-1 group), a GPU cluster for lectures in the Yonsei AI department. You can use an RTX3090 GPU for 300 hours per month. Here is the instruction about how to launch and use a workspace (i.e. a virtual machine with one GPU assigned to you) for this homework.

Warnings:

- Always stop the workspace when not in use to save the computing resources.
- The **VESSL Workspace** provides a maximum allocation of 72 hours of runtime. Exceeding this limit will automatically stop your workspace. Monitor your usage regularly to manage your available hours, and restart the workspace if it has been automatically stopped.

3 Play with Reinforcement Learning Environments

We provide a tutorial (`GymTutorial.ipynb`) inside the homework material to help you familiarize yourself to Reinforcement Learning (RL) environments. We encourage you to review and run this tutorial to understand how to interact with gym environments.

After completing the tutorial, please answer the following questions to demonstrate your understanding of the basic principles of gym environments.

3.1 Questions:

1. Describe the role of the `step()` function. What kind of information does it return?
   Answer:
   The role of the `step()` function is to run one timestep of the environment's dynamics, and get the new observation space.

   The `step()` function returns:
   `observation`, which is an element of the environment's `observation_space` attribute.
   `reward`, which is the amount of reward returned as a result of taking the action.

`terminated`, boolean that tells whether a `terminal state` is reached.
`info`, a dictionary containing auxiliary diagnostic information, which could be felpful for debugging, learning, and logging.

2. Describe the role of the `reset()` function in a gym environment. What is the return value of this function?
   Answer:
   The role of the `reset()` function is to reset the environment to an initial state and return the initial observation.

   The `reset()` function returns:
   `observation`, which is observation of the initial state.

3. How can you figure out if some gym environment has a discrete action space or continuous action space?
   Answer:
   We can query the `action_space` attribute of the `Env` class to figure out whether the actions space of the environment is discrete or continuous.

4 Behavioral Cloning

4.1 Implement Behavioral Cloning

Your task is to fill in sections marked with `TODO` in the code. In particular, see the following files:

- `aai4160/infrastructure/bc_trainer.py`, except for `do_relabel_with_expert` function, which is for the next section, DAgger.
- `aai4160/policies/MLP_policy.py`
- `aai4160/infrastructure/replay_buffer.py`
- `aai4160/infrastructure/utils.py`
- `aai4160/infrastructure/pytorch_util.py`

Run behavioral cloning (BC) and report results on **two tasks**: (1) the Ant environment (`Ant-v4`), where a behavioral cloning agent should achieve at least 30% of the performance of the expert, and (2) any one environment among `Walker2d-v4`, `HalfCheetah-v4`, and `Hopper-v4`, where the expert data is also provided.

The performance of the expert policy can be found in `Initial_DataCollection_AverageReturn` in the log output.

Once you implement `TODO` above, you can train a BC policy for the Ant task as follows:

```
python aai4160/scripts/run_hw1.py \
    --expert_policy_file aai4160/policies/experts/Ant.pkl \
    --env_name Ant-v4 --exp_name bc_ant --n_iter 1 \
    --expert_data aai4160/expert_data/expert_data_Ant-v4.pkl \
    --video_log_freq -1
```

If your run succeeds, you will be able to find your tensorboard log data in
`hw1_starter_code/data/q1_[--exp_name]_[--env_name]_[current_time]/`.

When providing results, report the **mean and standard deviation** of your policy's return **over multiple rollouts** in a table, and state which task was used. When comparing one that is working versus one that is not working, be sure to set up a fair comparison in terms of network size, amount of data, and number of training iterations. **Provide these details** (and any others you feel are appropriate) in the table caption.

Note: What "report the mean and standard deviation" means is that your `eval_batch_size` should be greater than `ep_len`, such that you're collecting multiple rollouts when evaluating the performance of your trained policy. For example, if `ep_len` is 1000 and `eval_batch_size` is 5000, then you'll be collecting approximately 5 episodes (maybe more if any of them terminate early), and the logged `Eval_AverageReturn` and `Eval_StdReturn` represents the mean/std of your policy over these 5 rollouts. Make sure you include these parameters in the table caption as well.

Note: To generate videos of the policy rollouts, remove the flag "`--video_log_freq -1`". However, this is slower, and so you probably want to keep this flag on while debugging.

4.1.1 BC Result

Table 1: `eval_batch_size=100000, L1Loss()` **for both experiments**

Environment	**Performance (Mean Return ± Std)**
`Ant-v4`	1348 ± 780.2
`HalfCheetah-v4`	2949 ± 153.9

4.2 Hyperparameter Tuning of Behavioral Cloning

Experiment with **one set of hyperparameters** that affects the performance of the behavioral cloning agent, such as the amount of training steps, the amount of expert data provided, or something that you come up with yourself. For one of the tasks used in the previous question, show a graph of how the BC agent's performance varies with the value of this hyperparameter. State the hyperparameter and a brief rationale for why you chose it.

You should include at least **4 different** settings for the hyperparameter you have chosen, including the default setting you used in the previous part.

Note: There are some default hyperparameters you can specify using the command line arguments. You may want to choose one of the hyperparameters listed below:

- Number of gradient steps for training policy (`--num_agent_train_steps_per_iter`, default: 1000)
- The amount of training data (`--batch_size`, default: 1000)
- Training batch size (`--train_batch_size`, default: 100)
- Depth of the policy neural net (`--n_layers`, default: 2)
- Width of the policy neural net (`--size`, default: 64)
- Learning rate for supervised learning (`--learning_rate`, default: $5e-3$)

You can specify the hyperparameter in the command line when you execute the script. For example, if you run the command like this, you can train the policy for 500 gradient steps:

```
python aai4160/scripts/run_hw1.py \
    --num_agent_train_steps_per_iter 500 \
    --some other arguments...,
```

Note: Use `matplotlib` for drawing the plots. If you are not familiar with `matplotlib`, you can refer to its official tutorial.

4.2.1 Hyperparameter Tuning Results

Hyperparameter: `learning_rate`
Trained the model with 8 different learning rate values, `learning_rate = [5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1]`

Plot:

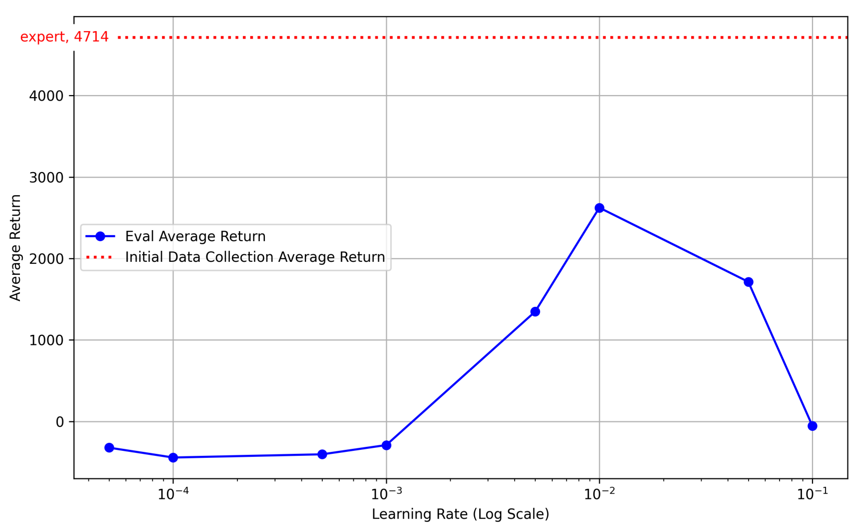

Figure 1: Average return values across different learning rates

Rationale:

When the learning rate is small, it can be assumed that sufficient learning has not occurred. Since same amount of training steps were done in all experiments, it is highly likely that the global minimum (or local minima that shows decent performance) was not reached in the loss landscape.

The highest return is shown when the learning rate is 1e-2, which can be considered to show good performance due to the appropriate learning rate and appropriate number of training steps.

Lastly, when the learning rate is large, the return value decreases rapidly. This can be thought of as a case of overshooting due to a learning rate value that is too large. In other words, the learning rate is not elaborate enough for the model to be trained well.

5 DAgger

5.1 Implement DAgger

Now your task is to implement the DAgger algorithm. If you implemented the BC part correctly, you can just implement the `TODO` in the following file.

- `do_relabel_with_expert` function in `aai4160/infrastructure/bc_trainer.py`

Once you have filled in all of the instructions specified with `TODO` comments in the code, you should be able to train DAgger with the following command:

```
python aai4160/scripts/run_hw1.py \
  --expert_policy_file aai4160/policies/experts/Ant.pkl \
  --env_name Ant-v4 --exp_name dagger_ant --n_iter 10 \
  --do_dagger \
  --expert_data aai4160/expert_data/expert_data_Ant-v4.pkl \
  --video_log_freq -1
```

5.2 Compare BC and DAgger

Run DAgger and report results on the two tasks you tested previously with BC (i.e., Ant + another environment). Report your results in the form of a learning curve, plotting the number of DAgger iterations vs. the policy's mean return. In the caption, state which task you used, and any details regarding network architecture, amount of data, etc. (as in the previous section).

Note: You can use the example helper script (`aai4160/scripts/parse_tensorboard.py`) to parse the data from the tensorboard logs and plot the figure. Here's an example usage that saves the figure as `output_plot.png`:

```
python aai4160/scripts/parse_tensorboard.py \
  --input_log_files data/[replace_here_with_the_name_of_log_folder] \
  --data_key "Eval_AverageReturn" \
  --title "DAgger: Ant-v4" \
  --x_label_name "DAgger iterations" \
  --y_label_name "Mean Return" \
  --output_file "output_plot.png"
```

You may also want to plot the performances of BC and expert policy as a horizontal line and plot the standard deviations as the error bars. Feel free to modify the example parsing script as you want.

5.2.1 DAgger Result

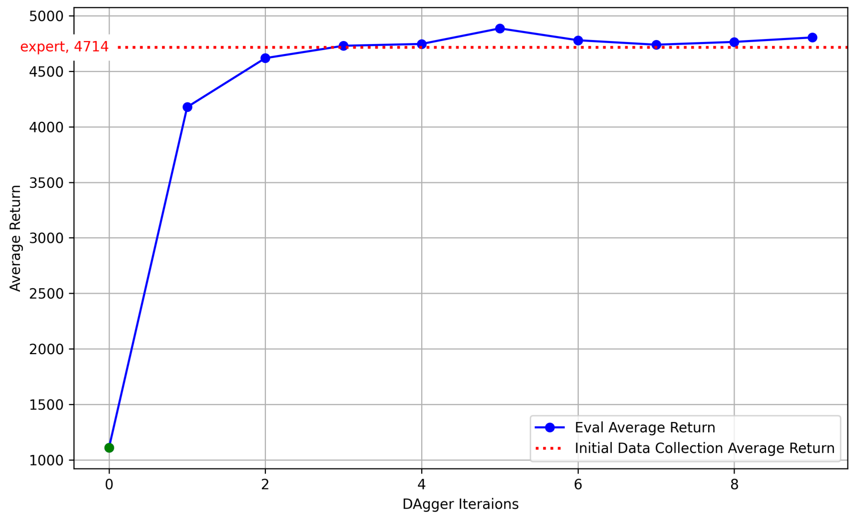

Figure 2: **Ant-v4, 2-layer MLP, eval_batch_size=1000, n_iter=10**

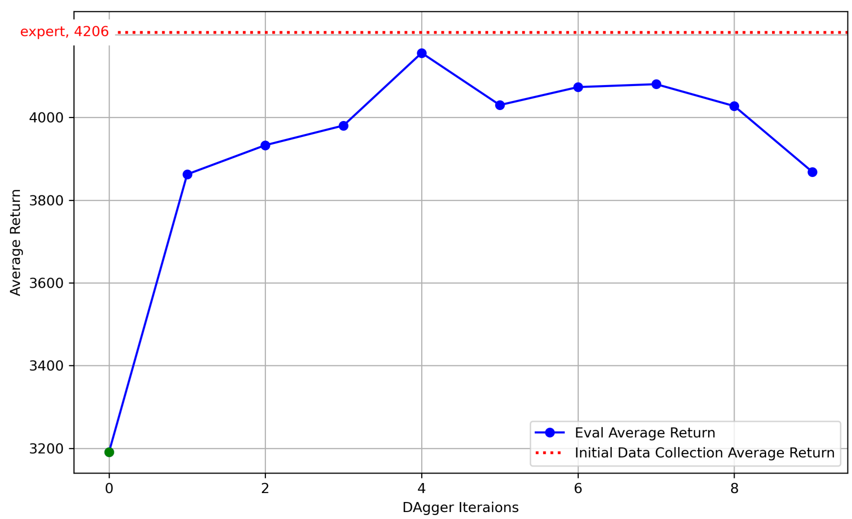

Figure 3: **HalfCheetah-v4, 2-layer MLP, eval_batch_size=1000, n_iter=10**

5.2.2 Justification of the Result

BC(Behavioral Cloning) is a supervised learning method which aims to mimic expert behavior. It is simple and straightforward to implement since it relies of standard supervised learning algorithms.

However, it is vulnerable to the distribution shift problem, wherein the agent must operate in states that the expert has never encountered.

DAgger(Dataset Aggregation) algorithm collects corrective behavior data online, and updates the policy given these data. It rolls out learned (current) policy, queries expert action at visited states, aggregate correction with existing data, and updates policy at every iteration. DAgger can be efficient when given small amounts of data.

For Ant-v4 environment[Fig.2], the initial average return value shows significantly lower value compared to the expert performance. But going through multiple iterations, return value continues to improve, reaches peak average return at 5th iteration, and shows slight fluctuation but maintains around expert's average return value.

Similar to Ant-v4 environment, for HalfCheetah-v4 environment[Fig.3], the agent starts with a lower average return that the expert, but improves as going through the iterations. But unlike Ant-v4 environment, the performance slightly decreases in the last few iterations, which could appear due to exploration or overfitting, etc.

For both environments, BC method shows lower performance than the expert. It could be due to small amount of existing expert data. Distribution shift could also be the problem, as mentioned above. DAgger can mitigate this problem, by collecting corrective behavior online, and using the collected data to improve the performance. It can be confirmed in both figure 2 and 3, where the average return value increases as the number of DAgger iteration increases.

6 Discussion

Please provide us a rough estimate, in hours, for each problem, how much time you spent. This will help us calibrate the difficulty for future homework.

- **Behavioral Cloning**: **20 hours**
- **DAgger**: **2 hours**

Feel free to share your feedback here, if any: **We would really appreciate your feedback to improve the reinforcement learning class.**

7 Submission

Please include the code, tensorboard logs data, and the report. Zip it to `hw1_[YourStudentID].zip`. The structure of the submission file should be:

```
hw1_YourStudentId.zip
├─ hw1_YourStudentId.pdf
├─ aai4160/
│   └─ ...codes
└─ data/
    └─ ...tensorboard log folders
```

Note: Do NOT include the videos (.mp4 files) in your submission. Your submission file size should be less than 15MB.