Assignment 1: Imitation Learning

Due Jan 30, 18:00

The goal of this assignment is to experiment with imitation learning, including direct behavior cloning and the DAgger algorithm. In lieu of a human demonstrator, demonstrations will be provided via an expert policy that we have trained for you. Your goals will be to set up behavior cloning and DAgger, and compare their performance on a few different continuous control tasks from the OpenAI Gym benchmark suite. Turn in your report and code as described in section 4.

The starter-code for this assignment can be found at

`https://github.com/milarobotlearningcourse/ift6163_homeworks_2023`

You have the option of running the code either on Google Colab (not completely supported) or on your own machine. Please refer to the `README` for more information on setup.

1 Behavioral Cloning

1. The starter code provides an expert policy for each of the MuJoCo tasks in OpenAI Gym. Fill in the blanks in the code marked with `Todo` to implement behavioral cloning. A command for running behavioral cloning is given in the Readme file.

   We recommend that you read the files in the following order. For some files, you will need to fill in blanks, labeled `TODO`.

   - `run_hw1.py`
   - `infrastructure/rl_trainer.py`
   - `agents/bc_agent.py` (another read-only file)
   - `policies/MLP_policy.py`
   - `infrastructure/replay_buffer.py`
   - `infrastructure/utils.py`
   - `infrastructure/pytorch_utils.py`

2. Run behavioral cloning (BC) and report results on two tasks: the Ant environment, where a behavioral cloning agent should achieve at least 30% of the performance of the expert, and one environment of your choosing where it does not. Here is how you can run the Ant task:

   ```
   python run_hw1.py
   ```

   This code uses hydra to organize experimental runs and data. There is a *YAML* file in *conf/config_hw1.yaml* that is used to control the parameters passed to the code for training. The YAML file looks like

   ```
   env:
     expert_policy_file: ./ift6163/policies/experts/Ant.pkl # Relative to where you're running this sc
     expert_data: ./ift6163/expert_data/expert_data_Ant-v2.pkl # Relative to where you're running this
     exp_name: "bob"
     env_name: Ant-v2 # choices are [Ant-v2, Humanoid-v2, Walker2d-v2, HalfCheetah-v2, Hopper-v2]
     max_episode_length: 100
     render: false

   alg:
     num_rollouts: 5
     do_dagger: false
   ```

```
  num_agent_train_steps_per_iter: 1000 # number of gradient steps for training policy (per iter in n
  n_iter: 1
  batch_size: 1000 # training data collected (in the env) during each iteration
  eval_batch_size: 1000 # eval data collected (in the env) for logging metrics
  train_batch_size: 100 # number of sampled data points to be used per gradient/train step
  n_layers: 2 # Network depths
  network_width: 64 # The width of the network layers
  learning_rate: 5e-3 # THe learning rate for BC
  max_replay_buffer_size: 1e5 ## Size of the replay buffer
  use_gpu: False
  which_gpu: 0 # The index for the GPU (the computer you use may have more than one)
  discrete: False
  ac_dim: 0 ## This will be overridden in the code
  ob_dim: 0 ## This will be overridden in the code

logging:
  video_log_freq: 5 # How often to generate a video to log/
  scalar_log_freq: 1 # How often to log training information and run evaluation during training.
  save_params: true # Should the parameters given to the script be saved? (Always...)
  random_seed: 1234
```

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. Submit your log file *data/.../log_file.log* on Gradescope as *ant1-2.log* for your Ant run and *custom1-2.log* for the run of your choosing.

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.

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 trajectories (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: Make sure to set both flags `video_log_freq` in *conf/config_hw1.yaml* to be greater than 1 and `render` to true when training your **final** agent for videos recording as requested in section 4.

3. 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. In the caption for the graph, state the hyperparameter and a brief rationale for why you chose it.

2 DAgger

1. Once you've filled in all of the `TODO` commands, you should be able to run DAgger by editing the *config.yaml* file here

```
alg:
  num_rollouts: 5
  do_dagger: true
```

and running again

```
python run_hw1.py
```

2. Run DAgger and report results on the two tasks you tested previously with behavioral cloning (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, with error bars to show the standard deviation. Include the performance of the expert policy and the behavioral cloning agent on the same plot (as horizontal lines that go across the plot). In the caption, state which task you used, and any details regarding network architecture, amount of data, etc. (as in the previous section). Submit the log file of your Ant run on Gradescope as *dagger_ant2-2.log*.

3 Bonus

1. Train the policy from images as well. You can look into how the video is recorded for your work and use that code to capture images and use that as input to the policy. You can use low resolution images.

4 Turning it in

1. **Submitting the PDF.** Make a PDF report containing: Table 1 for a table of results from Question 1.2, Figure 1 for Question 1.3. and Figure 2 with results from question 2.2.

   You do not need to write anything else in the report, just include the figures with captions as described in each question above. Full points for clear captions. See the handout at

   `http://rail.eecs.berkeley.edu/deeprlcourse/static/misc/viz.pdf`

   for notes on how to generate plots.

2. **Submitting log files on the autograder.** Make sure to submit all the log files that are requested on Gradescope you can find them in your log directory */data* by default.

3. **Submitting the code, experiment runs, and GIFs.** In order to turn in your code and experiment logs, create a folder that contains the following:

   - A folder named `run_logs` with experiments for both the behavioral cloning (part 2, not part 3) exercise and the DAgger exercise. Note that you can include multiple runs per exercise if you'd like, but you must include at least one run (of any task/environment) per exercise. These folders can be copied directly from the `output` folder into this new folder. You must also provide a GIF of your final policy for each question. To enable video logging, set both flags `video_log_freq` to greater than 1 and `render` to be true in *conf/config_hw1.yaml* and run your experiment. The GIFs will be located in the *outputs/path_to_experiment/videos/eval_step_x.gif* path.
   - The `ift6163` folder with all the `.py` files, with the same names and directory structure as the original homework repository. Also include the commands (with clear hyperparameters) and *conf/config_hw1.yaml* file that we need in order to run the code and produce the numbers that are in your figures/tables (e.g. run "python run_hw1.py –ep_len 200" to generate the numbers for Section 2 Question 2) in the form of a README file.

As an example, the unzipped version of your submission should result in the following file structure. **Make sure to include the prefix `q1_` and `q2_`.**

```
submit.zip
├── hw1
│   ├── run_logs
│   │   ├── q1_bc_ant
│   │   │   ├── events.out.tfevents.1567529456.e3a096ac8ff4
│   │   │   └── eval_step_x.gif
│   │   ├── q2_dagger_ant
│   │   │   ├── events.out.tfevents.1567529456.e3a096ac8ff4
│   │   │   └── eval_step_x.gif
│   │   └── ...
│   ├── roble
│   │   ├── agents
│   │   │   ├── bc_agent.py
│   │   │   └── ...
│   │   ├── policies
│   │   │   └── ...
│   │   └── ...
│   ├── conf
│   │   └── config_hw1.yaml
│   ├── README.md
│   └── ...
├── Dockerfile
├── Dockerfile_Ubuntu18.04
├── diff.txt
└── ...
```

You also need to include a diff of your code compared to the starter homework code. You can use the command

```
git diff cf9f83c040df586125e7d94dbe68ad5bbd6aa0c2 >> diff.txt
```

If you are a Mac user, **do not use the default "Compress" option to create the zip**. It creates artifacts that the autograder does not like. You may use `zip -vr submit.zip submit -x "*.DS_Store"` from your terminal.

Turn in your assignment on Gradescope. Upload the zip file with your code and log files to **HW1 Code**, and upload the PDF of your report to **HW1**.