# Assignment 1: Imitation Learning

**Due September 11, 11:59 pm**

The goal of this assignment is to gain familiarity with imitation learning, including direct behavioral 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 7.

The starter-code for this assignment can be found at

`https://github.com/berkeleydeeprlcourse/homework_fall2023/tree/main/hw1`

You have the option of running the code either on Google Colab or on your own machine. Please refer to the `README` for more information on setup.

**Note**: The Colab is only used as a source of GPU compute, so you will be editing the same code regardless of what option you choose. For this assignment, since GPU will not be necessary, we strongly recommend running the code locally to gain some familiarity with installing the necessary packages. This will be extremely beneficial for later homeworks, so you can run experiments in parallel.

If you are running locally we *strongly* recommend you use Conda to manage your Python environment and dependencies. Instructions for installing Conda and setting up an environment are included.

## 1 Analysis

Consider the problem of imitation learning within a discrete MDP with horizon $T$ and an expert policy $\pi^*$. We gather expert demonstrations from $\pi^*$ and fit an imitation policy $\pi_\theta$ to these trajectories so that

$$\mathbb{E}_{p_{\pi^*}(s)}\pi_\theta(a \neq \pi^*(s) \mid s) = \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{p_{\pi^*}(s_t)}\pi_\theta(a_t \neq \pi^*(s_t) \mid s_t) \leq \varepsilon,$$

i.e., the expected likelihood that the learned policy $\pi_\theta$ disagrees with the expert $\pi^*$ within the training distribution $p_{\pi^*}$ of states drawn from random expert trajectories is at most $\varepsilon$.

For convenience, the notation $p_\pi(s_t)$ indicates the state distribution under $\pi$ at time step $t$ while $p(s)$ indicates the state marginal of $\pi$ across time steps, unless indicated otherwise.

1. Show that $\sum_{s_t} |p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)| \leq 2T\varepsilon$.

   [Hint: In lecture, we showed a similar inequality under the stronger assumption $\pi_\theta(s_t \neq \pi^*(s_t) \mid s_t) \leq \varepsilon$ for every $s_t \in \text{supp}(p_{\pi^*})$. Try converting the inequality above into an expectation over $p_{\pi^*}$ and use a union bound ($\Pr[\bigcup_i E_i] \leq \sum_i \Pr[E_i]$) to get the desired result.]

2. Consider the expected return of the learned policy $\pi_\theta$ for a state-dependent reward $r(s_t)$, where we assume the reward is bounded with $|r(s_t)| \leq R_{\max}$:

   $$J(\pi) = \sum_{t=1}^{T}\mathbb{E}_{p_\pi(s_t)}r(s_t).$$

   (a) Show that $J(\pi^*) - J(\pi_\theta) = \mathcal{O}(T\varepsilon)$ when the reward only depends on the last state, i.e., $r(s_t) = 0$ for all $t < T$.

   (b) Show that $J(\pi^*) - J(\pi_\theta) = \mathcal{O}(T^2\varepsilon)$ for an arbitrary reward.

## 2 Editing Code

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.

- `scripts/run_hw1.py` (training loop)
- `policies/MLP_policy.py` (policy definition)
- `infrastructure/replay_buffer.py` (stores training trajectories)
- `infrastructure/utils.py` (utilities for sampling trajectories from a policy)
- `infrastructure/pytorch_utils.py` (utilities for converting between NumPy/Pytorch)

For some files, some important functionality is missing and are marked with `TODO`. Specifically, you are asked to implement parts of the following:

- `policies/MLP_policy.py`: `forward` and `update` functions
- `infrastructure/utils.py`: `sample_trajectory` function
- `scripts/run_hw1.py`: `run_training_loop` function (most of your code will be in here)

## 3 Behavioral Cloning

1. Run behavioral cloning (BC) and report results on two tasks: one 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 cs285/scripts/run_hw1.py \
     --expert_policy_file cs285/policies/experts/Ant.pkl \
     --env_name Ant-v4 --exp_name bc_ant --n_iter 1 \
     --expert_data cs285/expert_data/expert_data_Ant-v4.pkl \
     --video_log_freq -1
   ```

   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 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.

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

2. 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.

## 4 DAgger

1. Using the same code, you should be able to run DAgger by modifying the runtime parameters as follows:

```
python cs285/scripts/run_hw1.py \
  --expert_policy_file cs285/policies/experts/Ant.pkl \
  --env_name Ant-v4 --exp_name dagger_ant --n_iter 10 \
  --do_dagger --expert_data cs285/expert_data/expert_data_Ant-v4.pkl \
  --video_log_freq -1
```

2. Run DAgger and report results on the two tasks you tested previously with behavioral cloning. 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).

## 5 Extra Credit: SwitchDAgger

One of the shortcomings of the DAgger algorithm discussed in lecture is it requires the human expert to annotate optimal actions on states gathered by the robot. This may be counterintuitive, since humans usually select actions with continuous feedback from the environment.

In this question, you will analyze a variant of the DAgger algorithm that hands off control to the human expert at points during rollouts, allowing them to provide interactive demonstrations. We consider a discrete MDP with horizon $T$ and an expert policy $\pi^*$. At each iteration $n = 1, \dots, N$ we have a policy $\pi^n$. We roll out trajectories from this policy such that in each one we transfer control to the expert at some random time step $X^* + 1$ until the remainder of the trajectory. We denote the version of $\pi^n$ that hands off control with some probability as $\tilde{\pi}^n$.

Formally, define $S^X(\pi_1, \pi_2)$ to be the policy that executes policy $\pi_1$ for $X$ steps, and then switches to running policy $\pi_2$ from the current state for the remaining steps in the trajectory. We define our algorithm, SwitchDAgger, as follows. We set $\tilde{\pi}^0 \leftarrow \pi^*$ and $\pi^0 \leftarrow \hat{\pi}^1$ for convenience. At each step $n = 1, \dots, N$ we perform the following updates:

$$
\begin{aligned}
&\hat{\pi}^n \leftarrow \text{fit to expert actions } \pi^*(s) \text{ across } s \sim p_{\tilde{\pi}^{n-1}} \\
&\tilde{\pi}^n \leftarrow S^{X_n}(\hat{\pi}^n, \tilde{\pi}^{n-1}) \text{ where } X_n + 1 \sim \operatorname{Geom}(1-\alpha) \\
&\pi^n \leftarrow S^{X_n}(\hat{\pi}^n, \pi^{n-1}), \text{ or equivalently } S^{X^*}(\tilde{\pi}^n, \pi^0) \text{ for } X^* = \textstyle\sum_{i=1}^n X_i
\end{aligned}
$$

where we assume $\hat{\pi}^n$ is fit across the state marginal distribution of $\tilde{\pi}^{n-1}$ so that

$$\mathbb{E}_{s \sim p_{\tilde{\pi}_{n-1}}} \Pr[\hat{\pi}^n(s) \neq \pi^*(s)] \leq \varepsilon.$$

We define the cost as the expected number of errors made by a policy:

$$C(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim p_\pi} \Pr[\pi(s_t) \neq \pi^*(s_t)].$$

Our objective is to minimize the cost of the final policy produced by SwitchDAgger that does not use the expert, $\pi^N$. In the following parts, you will show that we can bound the cost $C(\pi^N)$ of this policy by $\mathcal{O}(T\varepsilon \log(1/\varepsilon))$ for a suitable choice of $N$ and $\alpha$.

1. Show we can bound $C(\tilde{\pi}^n) \leq A(T, n)$ for some $A(t, n)$ defined by the the following conditions:

$$
\begin{aligned}
A(0, n) &= 0, \\
A(t, 0) &= 0, \\
A(t, n) &= \alpha\varepsilon t + \alpha(1-\varepsilon)A(t-1, n) + (1-\alpha)A(t, n-1).
\end{aligned}
$$

2. Prove that $C(\tilde{\pi}^n) \leq Tn\alpha\varepsilon$.
3. Show that $C(\pi^n) \leq C(\tilde{\pi}^n) + Te^{\frac{-n}{(1-\alpha)T}}$ when $n \geq T$ and $\alpha \leq 1/T$. [Hint: a Chernoff bound gives $\Pr[X^* \leq T] \leq e^{\frac{-n}{(1-\alpha)T}}$.]
4. Conclude that for a choice of $\alpha$ and $N$ that depend on $\varepsilon$ and $T$ we can bound the cost of the final SwitchDAgger policy as $C(\pi^N) = \mathcal{O}(T\varepsilon \log(1/\varepsilon))$.

## 6 Discussion

Please answer the following short questions so we can improve future assignments.

1. How much time did you spend on each part of this assignment?
2. Any additional feedback?

## 7 Turning it in

1. **Submitting the PDF.** Make a PDF report containing: Responses for parts 1.1, 1.2, 6.1 and 6.2, Table 1 for a table of results from part 3.1, Figure 1 for Part 3.2, Figure 2 with results from question part 4.2, and any extra credit responses for parts 5.1 to 5.4.

   See the handout at

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

   for notes on how to generate plots.

2. **Submitting the code and experiment runs.** 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 `cs285/data` folder into this new folder. **Important: Disable video logging for the runs that you submit, otherwise the files size will be too large! You can do this by setting the flag `-video_log_freq -1`**
   - The `cs285` 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) 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_behavior_cloning.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 that the submit.zip file is below 15MB and that they include the prefix q1_ and q2_.**

```
submit.zip
├── run_logs
│   ├── q1_bc_ant
│   │   └── events.out.tfevents.1567529456.e3a096ac8ff4
│   ├── q1_dagger_ant
│   │   └── events.out.tfevents.1567529456.e3a096ac8ff4
│   └── ...
├── cs285
│   ├── scripts
│   │   ├── run_hw1.py
│   │   └── run_hw1.ipynb
│   ├── policies
│   │   └── ...
│   └── ...
├── README.md
└── ...
```

3. 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.
4. 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**.