Berkeley CS 285: Deep Reinforcement Learning, Decision Making, and Control
Fall 2023
Assignment 1 – Imitation Learning

Due September 11, 11:59 pm

1 Goal

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

Starter code:
`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.

2 Analysis

Consider imitation learning within a discrete MDP of horizon T and an expert policy π^*. We gather expert demonstrations from π^* and fit an imitation policy π_θ to these trajectories so that

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

That is, the expected likelihood that the learned policy π_θ disagrees with π^* on states drawn from the expert distribution p_{π^*} is at most ε.

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

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

Hint: In lecture we showed a similar inequality under the stronger assumption $\pi_\theta(a_t \neq \pi^*(s_t) \mid s_t) \le \varepsilon$ for every $s_t \in \mathrm{supp}(p_{\pi^*})$. Try converting the inequality above into an expectation over p_{π^*} and apply a union bound.

Solution. Assume
$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{s_t \sim p_{\pi^*}(s_t)}\left[\pi_\theta(a_t \neq \pi^*(s_t) \mid s_t)\right] \le \varepsilon, \qquad \text{i.e.,} \quad \mathbb{E}_{s\sim p_{\pi^*}}[\pi_\theta(a \neq \pi^*(s) \mid s)] \le \varepsilon.$$

For any policy π the state distribution satisfies
$$p_\pi(s_t) = \sum_{s_{t-1}} p_\pi(s_{t-1})\, P_\pi(s_t \mid s_{t-1}), \qquad P_\pi(s_t \mid s_{t-1}) = \sum_{a_{t-1}} P(s_t \mid s_{t-1}, a_{t-1})\, \pi(a_{t-1} \mid s_{t-1}).$$

Hence
$$\begin{aligned} p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t) = &\sum_{s_{t-1}} \big(p_{\pi_\theta}(s_{t-1}) - p_{\pi^*}(s_{t-1})\big)\, P_{\pi_\theta}(s_t \mid s_{t-1}) \\ &+ \sum_{s_{t-1}} p_{\pi^*}(s_{t-1})\big(P_{\pi_\theta}(s_t \mid s_{t-1}) - P_{\pi^*}(s_t \mid s_{t-1})\big). \end{aligned}$$

Taking ℓ_1 over s_t and using triangle inequality and $\sum_{s_t} P(\cdot) = 1$ gives
$$\sum_{s_t}\left|p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)\right| \le \sum_{s_{t-1}}\left|p_{\pi_\theta}(s_{t-1}) - p_{\pi^*}(s_{t-1})\right| + \sum_{s_{t-1}} p_{\pi^*}(s_{t-1})\,\Xi(s_{t-1}), \tag{2}$$

where
$$\Xi(s_{t-1}) := \sum_{s_t}\left|P_{\pi_\theta}(s_t \mid s_{t-1}) - P_{\pi^*}(s_t \mid s_{t-1})\right|.$$

Expand the transition difference:
$$P_{\pi_\theta}(s_t \mid s_{t-1}) - P_{\pi^*}(s_t \mid s_{t-1}) = \sum_{a_{t-1}} P(s_t \mid s_{t-1}, a_{t-1})\big(\pi_\theta(a_{t-1} \mid s_{t-1}) - \pi^*(a_{t-1} \mid s_{t-1})\big).$$

Therefore,
$$\Xi(s_{t-1}) \le \sum_{a_{t-1}} \left|\pi_\theta(a_{t-1} \mid s_{t-1}) - \pi^*(a_{t-1} \mid s_{t-1})\right|.$$

For any two distributions p, q over the same finite action set,
$$\sum_a |p(a) - q(a)| = 2\left(1 - \sum_a \min\{p(a), q(a)\}\right) \le 2\left(1 - \sum_a p(a)q(a)\right) = 2 \Pr_{a\sim p,\ a^*\sim q}[a \neq a^*].$$

Applying this with $p = \pi_\theta(\cdot \mid s_{t-1})$ and $q = \pi^*(\cdot \mid s_{t-1})$ and averaging over $s_{t-1} \sim p_{\pi^*}$ yields
$$\sum_{s_{t-1}} p_{\pi^*}(s_{t-1})\,\Xi(s_{t-1}) \le 2\,\mathbb{E}_{s_{t-1}\sim p_{\pi^*}}\left[\pi_\theta(a_{t-1} \neq \pi^*(s_{t-1}) \mid s_{t-1})\right] \le 2\varepsilon.$$

Let $\Delta_t := \sum_{s_t} |p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)|$. Plugging the bound above into (2) gives

$$\Delta_t \;\le\; \Delta_{t-1} + 2\varepsilon, \qquad \Delta_0 = 0.$$

By induction, $\Delta_t \le 2t\,\varepsilon$. In particular, for any $t \le T$,

$$\sum_{s_t} \big|p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)\big| \;\le\; 2t\varepsilon \;\le\; 2T\varepsilon.$$

2. Consider the expected return of the learned policy π_θ for a state-dependent reward $r(s_t)$, assume the reward is bounded, $|r(s_t)| \le R_{\max}$.

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

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

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

Solution. We define the expected return as

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

Then

$$J(\pi^*) - J(\pi_\theta) = \sum_{t=1}^{T} \Big(\mathbb{E}_{s_t \sim p_{\pi^*}}[r(s_t)] - \mathbb{E}_{s_t \sim p_{\pi_\theta}}[r(s_t)]\Big) = \sum_{t=1}^{T} \sum_{s_t} \big(p_{\pi^*}(s_t) - p_{\pi_\theta}(s_t)\big) r(s_t).$$

(a) When $r(s_t) = 0$ for all $t < T$, only the final step contributes:

$$J(\pi^*) - J(\pi_\theta) = \sum_{s_T} \big(p_{\pi^*}(s_T) - p_{\pi_\theta}(s_T)\big) r(s_T).$$

Taking the absolute value and using $|r(s_T)| \le R_{\max}$,

$$|J(\pi^*) - J(\pi_\theta)| \le R_{\max} \sum_{s_T} |p_{\pi^*}(s_T) - p_{\pi_\theta}(s_T)|.$$

From the previous result $\sum_{s_t} |p_{\pi^*}(s_t) - p_{\pi_\theta}(s_t)| \le 2T\varepsilon$,

$$|J(\pi^*) - J(\pi_\theta)| \le 2R_{\max} T\varepsilon = O(T\varepsilon).$$

(b) For an arbitrary reward, we have

$$J(\pi^*) - J(\pi_\theta) = \sum_{t=1}^{T} \sum_{s_t} \big(p_{\pi^*}(s_t) - p_{\pi_\theta}(s_t)\big) r(s_t).$$

Taking absolute values and using $|r(s_t)| \le R_{\max}$,

$$|J(\pi^*) - J(\pi_\theta)| \le R_{\max} \sum_{t=1}^{T} \sum_{s_t} |p_{\pi^*}(s_t) - p_{\pi_\theta}(s_t)|.$$

By the previous bound $\sum_{s_t} |p_{\pi^*}(s_t) - p_{\pi_\theta}(s_t)| \le 2t\varepsilon$,

$$|J(\pi^*) - J(\pi_\theta)| \le R_{\max} \sum_{t=1}^{T} 2t\varepsilon = 2R_{\max}\varepsilon \sum_{t=1}^{T} t = 2R_{\max}\varepsilon \frac{T(T+1)}{2} = R_{\max} T(T+1)\varepsilon = O(T^2\varepsilon).$$

3 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`: (run 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, important functionality is missing and 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 here).

4 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:

```
export PYTHONPATH="/media/zks/T7 Shield/2025上学期/深度强化学习/LAB/homework_fall2023:$PYTH
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` represent 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.

Solution Experiment on Ant-v4 environment.Initial parameters

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

Eval_AverageReturn : 907.4296264648438
Eval_StdReturn : 0.0
Eval_MaxReturn : 907.4296264648438
Eval_MinReturn : 907.4296264648438
Eval_AverageEpLen : 1000.0
Train_AverageReturn : 4681.891673935816
Train_StdReturn : 30.70862278765526
Train_MaxReturn : 4712.600296723471
Train_MinReturn : 4651.18305114816
Train_AverageEpLen : 1000.0
Training Loss : 0.03436638414859772
Train_EnvstepsSoFar : 0
TimeSinceStart : 1.130906105041504
Initial_DataCollection_AverageReturn : 4681.891673935816
```

Experiment on Ant-v4 environment 2000 num_agent_train_steps_per_iter

```
python cs285/scripts/run_hw1.py \
 --expert_policy_file cs285/policies/experts/Ant.pkl \
 --env_name Ant-v4 --exp_name bc_ant_2000 --n_iter 1 \
 --expert_data cs285/expert_data/expert_data_Ant-v4.pkl \
 --video_log_freq -1 --num_agent_train_steps_per_iter 2000

 Eval_AverageReturn : 2917.578857421875
 Eval_StdReturn : 1281.930908203125
 Eval_MaxReturn : 4199.509765625
 Eval_MinReturn : 1635.6478271484375
 Eval_AverageEpLen : 695.5
 Train_AverageReturn : 4681.891673935816
 Train_StdReturn : 30.70862278765526
 Train_MaxReturn : 4712.600296723471
 Train_MinReturn : 4651.18305114816
 Train_AverageEpLen : 1000.0
 Training Loss : 0.010485836304724216
 Train_EnvstepsSoFar : 0
 TimeSinceStart : 1.8920543193817139
 Initial_DataCollection_AverageReturn : 4681.891673935816
 Done logging...
```

Experiment on Ant-v4 environment.5000 num_agent_train_steps_per_iter

```
python cs285/scripts/run_hw1.py \
  --expert_policy_file cs285/policies/experts/Ant.pkl \
  --env_name Ant-v4 --exp_name bc_ant_5000 --n_iter 1 \
  --expert_data cs285/expert_data/expert_data_Ant-v4.pkl \
  --video_log_freq -1 --num_agent_train_steps_per_iter 5000

Eval_AverageReturn : 4674.00244140625
Eval_StdReturn : 0.0
Eval_MaxReturn : 4674.00244140625
Eval_MinReturn : 4674.00244140625
Eval_AverageEpLen : 1000.0
Train_AverageReturn : 4681.891673935816
Train_StdReturn : 30.70862278765526
Train_MaxReturn : 4712.600296723471
Train_MinReturn : 4651.18305114816
Train_AverageEpLen : 1000.0
Training Loss : 0.0019066202221438289
Train_EnvstepsSoFar : 0
TimeSinceStart : 4.066415071487427
Initial_DataCollection_AverageReturn : 4681.891673935816
Done logging...
```

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.

5 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).

Solution Experiment on Ant-v4 environment DAgger

```
python cs285/scripts/run_hw1.py \
 --expert_policy_file cs285/policies/experts/Ant.pkl \
 --env_name Ant-v4 --exp_name dagger_ant_original --n_iter 10 \
 --do_dagger --expert_data cs285/expert_data/expert_data_Ant-v4.pkl \
 --video_log_freq -1
 Eval_AverageReturn : 4523.068359375
 Eval_StdReturn : 0.0
 Eval_MaxReturn : 4523.068359375
 Eval_MinReturn : 4523.068359375
 Eval_AverageEpLen : 1000.0
 Train_AverageReturn : 4800.208984375
 Train_StdReturn : 0.0
 Train_MaxReturn : 4800.208984375
 Train_MinReturn : 4800.208984375
 Train_AverageEpLen : 1000.0
 Training Loss : 0.0004822885093744844
 Train_EnvstepsSoFar : 9000
 TimeSinceStart : 14.66175365447998
```

Experiment on Ant-v4 environment 2000 num_agent_train_steps_per_iter

```
python cs285/scripts/run_hw1.py \
  --expert_policy_file cs285/policies/experts/Ant.pkl \
  --env_name Ant-v4 --exp_name dagger_ant_2000 --n_iter 10 \
  --do_dagger --expert_data cs285/expert_data/expert_data_Ant-v4.pkl \
  --video_log_freq -1 --num_agent_train_steps_per_iter 2000
  Collecting data for eval...
  Eval_AverageReturn : 4674.890625
  Eval_StdReturn : 0.0
  Eval_MaxReturn : 4674.890625
  Eval_MinReturn : 4674.890625
  Eval_AverageEpLen : 1000.0
  Train_AverageReturn : 4737.00537109375
  Train_StdReturn : 0.0
  Train_MaxReturn : 4737.00537109375
  Train_MinReturn : 4737.00537109375
  Train_AverageEpLen : 1000.0
  Training Loss : 0.00027269450947642326
  Train_EnvstepsSoFar : 9442
  TimeSinceStart : 20.868728160858154
  Done logging...
```

6 Evaluation

Eval_AverageReturn: Average cumulative reward per episode during evaluation. This reflects the overall performance of the current policy. A higher value indicates better imitation or control ability.

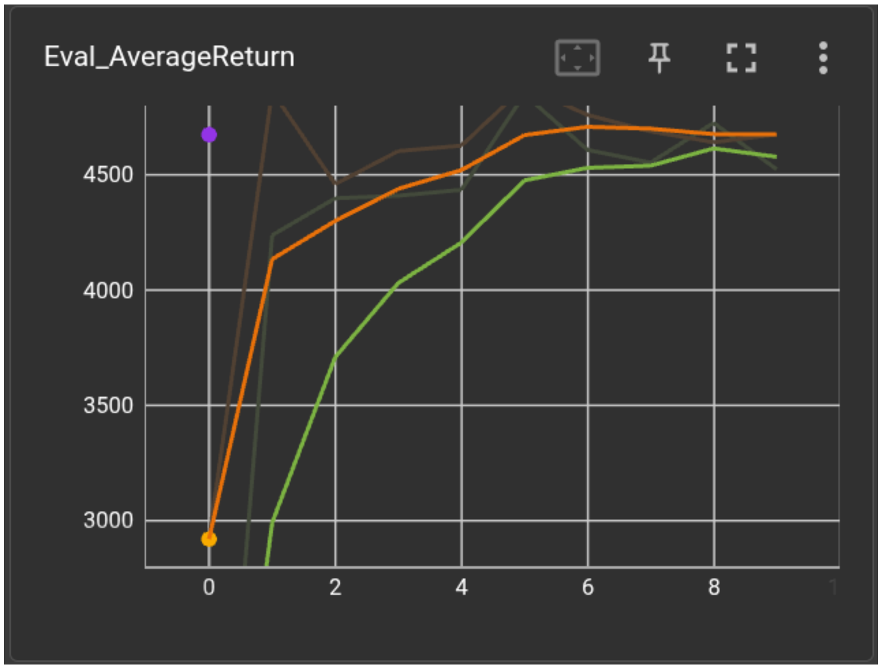

Figure 1: DAgger 在 Ant-v4 环境中的学习曲线。横轴为迭代次数，纵轴为平均回报。

Eval_StdReturn: Standard deviation of evaluation returns, measuring the variability of policy performance across episodes.

Eval_MaxReturn / Eval_MinReturn: Maximum and minimum total rewards obtained during evaluation. These values represent the best and worst case performances of the policy.

Eval_AverageEpLen: Average episode length during evaluation. A longer episode length implies that the agent can maintain stability for a longer time without failure.

Train_AverageReturn: Average cumulative reward from training rollouts stored in the replay buffer. Reflects the quality of training samples used for imitation learning.

Train_StdReturn: Standard deviation of returns in training trajectories, indicating how diverse or stable the sampled trajectories are.

Train_MaxReturn / Train_MinReturn: Maximum and minimum returns from the training data. These indicate the range of performance in collected rollouts.

Train_AverageEpLen: Average episode length in training data. It provides another measure of behavioral stability during training.

Training Loss (MSE): The supervised learning loss between predicted and expert actions:

$$L = \|\pi_\theta(o) - a_{\text{expert}}\|^2$$

A smaller value indicates the policy is better at imitating the expert' s behavior.

Train_EnvstepsSoFar: Total number of environment steps taken so far during training. Used to measure the scale of interaction data collected.

TimeSinceStart: Total elapsed wall-clock time since the beginning of training (in seconds). Useful for tracking computational efficiency.

7 (Extra Credit) SwitchDAgger

One of the shortcomings of the DAgger algorithm discussed in lecture is that 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 π^*. At each iteration $n = 1, \dots, N$ we have a policy π^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 π^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 π_1 for X steps, and then switches to running policy π_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:

$$\hat{\pi}^n \leftarrow \text{fit to expert actions } \pi^*(s) \text{ across } s \sim p_{\tilde{\pi}^{n-1}},$$

$$\tilde{\pi}^n \leftarrow S_{X_n}(\hat{\pi}^n, \tilde{\pi}^{n-1}) \quad \text{where } X_n + 1 \sim \text{Geom}(1-\alpha),$$

$$\pi^n \leftarrow S_{X_n}(\hat{\pi}^n, \pi^{n-1}), \quad \text{or equivalently } S_{X^*}(\tilde{\pi}^n, \pi^0) \text{ for } X^* = \sum_{i=1}^{n} X_i,$$

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, π^N. In the following parts, you will show that we can bound the cost $C(\pi^N)$ of this policy by $O(T\varepsilon \log(1/\varepsilon))$ for a suitable choice of N and α.

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

Solution 1. 取 $q = \alpha$，并注意交接步数 $X_n + 1 \sim \mathrm{Geom}(1-q)$。

于是有

$$C(\tilde{\pi}^n) = \sum_{k=1}^{T-1} q^k(1-q)\, C(\tilde{\pi}^n \mid X_n = k) + (1-q)\, C(\tilde{\pi}^n \mid X_n = 0). \tag{1}$$

其中，条件期望项可写作：

$$C(\tilde{\pi}^n \mid X_n = k) = \sum_{t=1}^{k} \mathbb{E}_{s_t \sim p_t^{(\hat{\pi}^n)}} \mathrm{err}_{\hat{\pi}^n}(s_t) + \sum_{t=k+1}^{T} \mathbb{E}_{s_t \sim p_{t-k}^{(\tilde{\pi}^{n-1});\,\mathrm{start}\sim p_k^{(\hat{\pi}^n)}}} \mathrm{err}_{\tilde{\pi}^{n-1}}(s_t), \tag{2}$$

即前 k 步由 $\hat{\pi}^n$ 控制，之后从状态分布 $s_k \sim p_k^{(\hat{\pi}^n)}$ 起转由 $\tilde{\pi}^{n-1}$ 控制。

由几何分布的性质，

$$\sum_{k=1}^{T-1} q^k(1-q) = 1 - q^{T-1},$$

并代入式 (1) 与 (2)，得到

$$\begin{aligned} C(\tilde{\pi}^n) = \sum_{k=1}^{T-1} q^k(1-q) &\left[\sum_{t=1}^{k} \mathbb{E}_{s_t \sim p_t^{(\hat{\pi}^n)}} \mathrm{err}_{\hat{\pi}^n}(s_t) + \sum_{t=k+1}^{T} \mathbb{E}_{s_t \sim p_{t-k}^{(\tilde{\pi}^{n-1});\,\mathrm{start}\sim p_k^{(\hat{\pi}^n)}}} \mathrm{err}_{\tilde{\pi}^{n-1}}(s_t) \right] \\ &+ (1-q)\sum_{t=1}^{T} \mathbb{E}_{s_t \sim p_t^{(\tilde{\pi}^{n-1})}} \mathrm{err}_{\tilde{\pi}^{n-1}}(s_t). \end{aligned} \tag{3}$$

取 $q = \alpha$，并按交接时刻分段（$X_n + 1 \sim \mathrm{Geom}(1-q)$）：

$$\boxed{\sum_{k=1}^{T-1} q^k(1-q) \left[\sum_{t=1}^{k} \mathbb{E}_{s_t \sim p_t^{(\hat{\pi}^n)}} \mathrm{err}_{\hat{\pi}^n}(s_t) \;+\; \sum_{t=k+1}^{T} \mathbb{E}_{s_t \sim p_{t-k}^{(\tilde{\pi}^{n-1})};\,\mathrm{start}\sim p_k^{(\hat{\pi}^n)}} \mathrm{err}_{\tilde{\pi}^{n-1}}(s_t) \right]} \tag{3}$$

把 q 提取出来（只重排权重，不改被积项），得

$$(3) \;=\; q\sum_{k=1}^{T-1} q^{k-1}(1-q)\; \underbrace{\left[\sum_{t=1}^{k} \mathbb{E}_{s_t \sim p_t^{(\hat{\pi}^n)}} \mathrm{err}_{\hat{\pi}^n}(s_t) + \sum_{t=k+1}^{T} \mathbb{E}_{s_t \sim p_{t-k}^{(\tilde{\pi}^{n-1})};\,\mathrm{start}\sim p_k^{(\hat{\pi}^n)}} \mathrm{err}_{\tilde{\pi}^{n-1}}(s_t) \right]}_{:=\ \boxed{(5)_k}} \tag{4}$$

Step (5): 拆第 $t=1$ 步　记事件 $E=\{\hat{\pi}^n(s_1)\neq\pi^*(s_1)\}$。则

$$(5)_k = \underbrace{\mathbb{E}\left[\mathbf{1}\{E\}\left(1+\sum_{t=2}^{k}\mathrm{err}_{\hat{\pi}^n}(s_t)\right)\right]}_{\text{第 1 步出错分支}} + \underbrace{\mathbb{E}\left[\mathbf{1}\{\neg E\}\left(\sum_{t=2}^{k}\mathrm{err}_{\hat{\pi}^n}(s_t)\right)\right]}_{\text{第 1 步正确：继续按 } \hat{\pi}^n}$$
$$+ \underbrace{\mathbb{E}\left[\mathbf{1}\{\neg E\}\left(\sum_{t=k+1}^{T}\mathrm{err}_{\tilde{\pi}^{n-1}}(s_t)\right)\right]}_{\text{从 } s_k\sim p_k^{(\hat{\pi}^n)} \text{ 起接 } \tilde{\pi}^{n-1}} \tag{5}$$

将三项分别写成条件期望：

$$(5)_k = \Pr(E)\,\mathbb{E}\left[1+\sum_{t=2}^{k}\mathrm{err}_{\hat{\pi}^n}(s_t)\;\middle|\;E\right] + \Pr(\neg E)\,\mathbb{E}\left[\sum_{t=2}^{k}\mathrm{err}_{\hat{\pi}^n}(s_t)\;\middle|\;\neg E\right]$$
$$+ \Pr(\neg E)\,\mathbb{E}\left[\sum_{t=k+1}^{T}\mathrm{err}_{\tilde{\pi}^{n-1}}(s_t)\;\middle|\;\neg E\right]. \tag{6}$$

把"相等分支"改写成 $A(T-1,n)$ 的同型（用马尔可夫性质）　在 $\neg E$ 下有 $\hat{\pi}^n(s_1)=\pi^*(s_1)$，由马尔可夫性质，从第 2 步起的状态分布只依赖于 (s_1,a_1)，而此时两者动作一致，故"去掉第 1 步"后得到的两段之和与把"总步数减 1、时间指标整体后移 1"同型。定义变元 $u=t-1$，得

$$\mathbb{E}\left[\sum_{t=2}^{k}\mathrm{err}_{\hat{\pi}^n}(s_t)\;\middle|\;\neg E\right] = \sum_{u=1}^{k-1}\mathbb{E}_{s_u\sim p_u^{(\hat{\pi}^n)}}\,\mathrm{err}_{\hat{\pi}^n}(s_u), \tag{7}$$
$$\mathbb{E}\left[\sum_{t=k+1}^{T}\mathrm{err}_{\tilde{\pi}^{n-1}}(s_t)\;\middle|\;\neg E\right] = \sum_{u=k}^{T-1}\mathbb{E}_{s_u\sim p_{u-k}^{(\tilde{\pi}^{n-1})};\ \mathrm{start}\sim p_{k-1}^{(\hat{\pi}^n)}}\,\mathrm{err}_{\tilde{\pi}^{n-1}}(s_u). \tag{8}$$

于是（把"步数整体减 1"），对每个 k 有

$$\boxed{\mathbb{E}\left[\sum_{t=2}^{k}\mathrm{err}_{\hat{\pi}^n}(s_t)+\sum_{t=k+1}^{T}\mathrm{err}_{\tilde{\pi}^{n-1}}(s_t)\;\middle|\;\neg E\right] \equiv (\text{与 } A(T-1,n) \text{ 同型的两段和，} k\mapsto k-1)} \tag{9}$$

把 (6) 代回 (4)，并把 $\{q^{k-1}(1-q)\}_{k=1}^{T-1}$ 视作对 $k-1\in\{0,\ldots,T-2\}$ 的几何权重，就得到

$$(3) \;=\; q\sum_{r=0}^{T-2}q^r(1-q)\Big(\;\Pr(E)\,\Xi_r \;+\; \Pr(\neg E)\;\underbrace{\Theta_r}_{\text{与 } A(T-1,n) \text{ 同型}}\;\Big),$$

其中 Ξ_r 是"第 1 步出错分支"的（上界）贡献，Θ_r 为 (9) 的那部分。

由此即可把 (3) 的 α 部分写成

$$q\Big(\;\Pr(E)\cdot[\text{出错项}] \;+\; \Pr(\neg E)\cdot A(T-1,n)\;\Big),$$

再与"首步交接"项 $(1-q)C(T,n-1)$ 合并，得到递推公式。.

2. 显然

3. When $n\geq T$ and $\alpha\leq 1/T$, we have

$$C(\pi^n)\leq C(\tilde{\pi}^n)+Te^{-\frac{n}{(1-\alpha)T}}.$$

Proof. Recall that

$$\pi^{n} = S_{X^*}(\tilde{\pi}^{n}, \pi^{0}), \qquad X^* = \sum_{i=1}^{n} X_i, \quad X_i + 1 \sim \mathrm{Geom}(1-\alpha),$$

that is, π^{n} executes $\tilde{\pi}^{n}$ for X^* steps and then switches to π^{0} if more steps remain.

Let

$$C(\pi) = \sum_{t=1}^{T} \Pr[\pi(s_t) \neq \pi^*(s_t)]$$

be the expected number of errors within a horizon of T.

We consider two cases for each trajectory:

- **Case 1:** $X^* \geq T$**.** No switch occurs within the T-step rollout. Then π^{n} and $\tilde{\pi}^{n}$ are identical on $\{1, \ldots, T\}$, hence their per-step errors coincide.
- **Case 2:** $X^* \leq T$**.** A switch may occur within the first T steps. In the worst case this can change at most T individual step errors (under 0 – 1 loss).

Therefore, trajectory-wise we have

$$\sum_{t=1}^{T} \mathbf{1}\{\pi^{n}(s_t) \neq \pi^*(s_t)\} \;\leq\; \sum_{t=1}^{T} \mathbf{1}\{\tilde{\pi}^{n}(s_t) \neq \pi^*(s_t)\} \;+\; T\,\mathbf{1}\{X^* \leq T\}.$$

Taking expectations over both the environment and algorithm randomness yields

$$C(\pi^{n}) \;\leq\; C(\tilde{\pi}^{n}) \;+\; T\,\Pr[X^* \leq T]. \qquad (\star)$$

By the Chernoff bound provided in the hint, when $n \geq T$ and $\alpha \leq 1/T$,

$$\Pr[X^* \leq T] \leq \exp\left(-\frac{n}{(1-\alpha)T}\right).$$

Substituting this into $(\star)$ gives

$$\boxed{C(\pi^{n}) \;\leq\; C(\tilde{\pi}^{n}) \;+\; T\,e^{-\frac{n}{(1-\alpha)T}}.}$$

□ 4.

Claim. There exist choices of α and N depending on ε and T such that

$$C(\pi^{N}) \;=\; \mathcal{O}\big(T\,\varepsilon\,\log(1/\varepsilon)\big).$$

Proof. Take

$$L \;=\; \left\lceil c\,\log\frac{1}{\varepsilon}\right\rceil, \qquad \alpha \;=\; 1 - \frac{1}{L}, \qquad N \;=\; c'\,T,$$

for absolute constants $c, c' > 0$ chosen below.

By $C_t(\tilde{\pi}^{n}) \leq (1-\alpha)\,C_t(\tilde{\pi}^{n-1}) + \alpha\big[\varepsilon t + (1-\varepsilon)\,C_{t-1}(\tilde{\pi}^{n})\big]$,

$$C(\tilde{\pi}^{N}) \;\leq\; \frac{\varepsilon\,T}{1-\alpha} \;=\; \varepsilon\,T\,L \;=\; \mathcal{O}\big(T\,\varepsilon\,\log(1/\varepsilon)\big).$$

For the switching tail, $(1-\alpha)T = T/L$ we get

$$T\,\exp\Big(-\tfrac{N}{(1-\alpha)T}\Big) \;=\; T\,\exp\Big(-\tfrac{N}{T/L}\Big) \;=\; T\,\exp\Big(-\tfrac{L\,N}{T}\Big) \;\leq\; T\,\exp(-c'L).$$

Taking c' large enough so that $e^{-c'L} \leq \varepsilon$ (e.g. $c' \geq 1$) yields

$$T\,\exp\Big(-\tfrac{N}{(1-\alpha)T}\Big) \;\leq\; T\,\varepsilon.$$

Combining the two bounds gives

$$C(\pi^N) \;\leq\; \underbrace{\mathcal{O}\big(T\,\varepsilon\,\log(1/\varepsilon)\big)}_{\text{from } C(\tilde{\pi}^N)} + \underbrace{\mathcal{O}(T\,\varepsilon)}_{\text{switching tail}} \;=\; \mathcal{O}\big(T\,\varepsilon\,\log(1/\varepsilon)\big).$$

8 Discussion

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

9 Submission

Submit a PDF report and a zip of code and logs on Gradescope. See the handout for expected directory structure and size limits.