CSE 571 - Robotics
Homework 3 - Reinforcement Learning

Due Monday June 5th @ 11:59pm

The key goal of this homework is to get an understanding of imitation learning and reinforcement learning methods - Behavior Cloning, DAgger and Policy gradient. Please refer to the git link for the assignment can be found at `https://github.com/jiafei1224/cse571-25sp`.

Collaboration: Students can discuss questions, but each student MUST write up their own solution, and code their own solution. We will be checking code/PDFs for plagiarism.

Late Policy: This assignment may be handed in up to 5 days late. If you have used up your 6 late days this quarter, there will be a penalty of 20% of the maximum grade per day.

1 Code Overview

The starter code is written in Python and depends on NumPy and Matplotlib as well as pytorch. If you are new to pytorch, please refer to the following tutorial. The README describes how to install packages in a conda environment to solve this assignment. We also provide a notebook which can be uploaded to Google Colab. We recommend either using a linux machine or using the colab, rather than windows. This section gives a brief overview and the README provides detailed instructions.

- `main.py` - overall launcher with hyperparameters and environment creation [DO NOT MODIFY]
- `environment.yml` - Conda env file to install dependencies [DO NOT MODIFY]
- `policy.py` - Code to load in expert policy [DO NOT MODIFY]
- `pytorch_utils.py` - Helper functions for pytorch [DO NOT MODIFY]
- `utils.py` - Helper functions for taking rollouts and collecting data [DO NOT MODIFY]
- `evaluate.py` - Evaluating learned policy reward and success rate [DO NOT MODIFY]
- `data` - Contains the expert data and interactive expert policy [DO NOT MODIFY]
- `bc.py` - Code for behavior cloning [FILL THIS IN]
- `dagger.py` - Code for DAgger [FILL THIS IN]
- `policy_gradient.py` - Code for REINFORCE [FILL THIS IN]
- `CSE571_HW3.ipynb` - Notebook for colab

2 Behavior Cloning [25 points]

2.1 Environment Details

You are provided with the Reacher environment, a 2D environment where a double-jointed arm aims to move its end effector to a target location. Details about the observation space and action space can be found at `https://www.gymlibrary.dev/environments/mujoco/reacher`.

2.2 Pseudo-code

This is approximate pseudocode for behavior cloning. Please make sure to convert to the appropriate pytorch commands. While it's anticipated that you will achieve a success rate exceeding 0.2, it's not strictly necessary for every run to surpass this threshold.

```
def behavior_cloning(expert_data, kwargs**):
    initialize policy
    for i in range(max_training_iters): // typical supervised learning loop
        s_batch, a_batch = sample_batch(expert_data)
        a_hat = policy(s_batch)
        loss = l2norm(a_hat, a_batch).mean() // MSE on the actions
        loss.backward()
    return
```

This serves as a reference for the expected output; please note that the success rate and reward may fluctuate:

```
$ python main.py --task behavior_cloning
using device cuda
Imported Expert data successfully
[0] loss: 0.06146659
[50] loss: 0.00734198
[100] loss: 0.00544504
[150] loss: 0.00429730
[200] loss: 0.00343579
[250] loss: 0.00285021
[300] loss: 0.00247725
[350] loss: 0.00232647
[400] loss: 0.00231698
[450] loss: 0.00318722
...
Success rate:  0.26
Average reward (success only):  -5.443988106540766
Average reward (all):  -10.3634220280316
```

2.3 Execution

1. Fill in the blanks in the code marked with TODO in the `simulate_policy_bc` function in `bc.py`.
2. Plot the loss during the training with default hyper-parameters. Report the success rate and average reward using the evaluate function that is provided to you.
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

To run the behavioral cloning assignment use:

```
python3 main.py --task behavior_cloning
```

3 DAgger [25 points]

3.1 Environment Details

You will use the same Reacher environment as in Sec. 2.

3.2 Pseudo-code

This is approximate pseudocode for DAgger. Please make sure to convert to the appropriate pytorch commands. You are expected to get a success rate high than 0.8.

```
def dagger(expert_data, expert_policy, kwargs**):
    initialize policy
    dataset = expert_data
    for i in range(max_dagger_iters):
        for i in range(max_training_iters): // run standard behavior cloning
            s_batch, a_batch = sample_batch(dataset)
            a_hat = policy(s_batch)
            loss = l2norm(a_hat, a_batch).mean()
            loss.backward()
        rollouts = rollout(policy) // roll out learned policy
        relabelled_rollouts = relabel_action(rollouts, expert_policy) // relabel actions with expert
        dataset += relabelled_rollouts // aggregate dataset
    return
```

When you execute the code, you may get similar outputs as below. Please note that the success rate and reward may fluctuate.

```
$ python main.py --task dagger
using device cuda
Imported Expert data successfully
Expert policy loaded
Average DAgger return is -8.43009561903899
Average DAgger return is -11.894969315160871
Average DAgger return is -6.679199372623998
Average DAgger return is -6.588657506211716
Average DAgger return is -6.287488364928622
Average DAgger return is -3.184413268215377
Average DAgger return is -4.609909559022624
Average DAgger return is -4.303682550954855
Average DAgger return is -3.6138667664531994
Average DAgger return is -5.071428497248543
```

```
...
Success rate:  1.0
Average reward (success only):  -4.018171099077643
Average reward (all):  -4.018171099077643
```

3.3 Execution

1. Fill in the blanks in the code marked with TODO in the `simulate_policy_dagger` function in `dagger.py`.
2. Plot the loss during the training with default hyper-parameters. Report the success rate and average reward using the evaluate function that is provided to you.
3. Compare the success rate and reward of the DAgger policy with the behavior cloning policy and explain why DAgger performs better.
4. Experiment with one set of hyperparameters that affects the performance of the 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 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 Policy Gradient [50 points]

4.1 Environment Details

The Inverted Pendulum environment is a simulation of a classic control problem called the inverted pendulum. The objective is to control the movement of an inverted pendulum by applying appropriate forces to keep it balanced. To learn about the observation space and action space, check out `https://www.gymlibrary.dev/environments/mujoco/inverted_pendulum`. You may also find `https://spinningup.openai.com/en/latest/algorithms/vpg.html` useful as a reference for the policy gradient method.

4.2 Pseudo-code

You are expected to get a success rate high than 0.8 and average reward (all) higher than 180.

```
def policy_gradient():
    initialize policy neural network
    instantiate baseline neural network
    for i in range(max_num_iters):
        trajectories = rollout(policy) // roll out current learned policy
        returns = compute_returns(policy) // compute return to go from observations
        returns = (returns - returns.mean())/(returns.std() + 1e-9) // normalize returns
        for i in range(baseline_training_iters): // train baseline via regression
            baseline_prediction = baseline(trajectories['observations'])
            loss_baseline = l2norm(baseline_prediction, returns).mean()
            loss_baseline.backward() // update baseline only

        baseline_prediction = baseline(trajectories['observations']) // compute final baseline
            prediction

        mean_predicted, std_predicted = policy(trajectories['observations'])
        log_probs = log_density(trajectories['actions'], mean_predicted, std_predicted)
```

```
        loss_policy = -log_probs*(returns - baseline_prediction)
        loss_policy.backward() // update policy only
    return
```

When you execute the code, you may get outputs similar to the one below. Please note that the success rate and reward may fluctuate.

```
$ python main.py --task policy_gradient
using device cuda
Episode: 0, reward: 8.48, max path length: 24
Episode: 10, reward: 10.08, max path length: 39
Episode: 20, reward: 15.85, max path length: 48
Episode: 30, reward: 28.73, max path length: 88
Episode: 40, reward: 52.75, max path length: 129
Episode: 50, reward: 109.34, max path length: 200
Episode: 60, reward: 152.19, max path length: 200
Episode: 70, reward: 173.68, max path length: 200
Episode: 80, reward: 189.03, max path length: 200
Episode: 90, reward: 195.31, max path length: 200
...
Success rate:  0.9
Average reward (success only):  200.0
Average reward (all):  197.45
```

4.3 Execution

1. Fill in the blanks in the code marked with TODO in the `train_model` function in `policy_gradient.py`.
2. Plot the loss during the training with default hyper-parameters. Report the success rate and average reward using the evaluate function that is provided to you.
3. Experiment with one set of hyperparameters that affects the performance of the 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 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 Submission

We will be using the Canvas for submission of the assignments. Please submit the written assignment answers as a PDF. For the code, submit a zip file of the entire working directory.