Assignment 5: Offline Reinforcement Learning

Due December 1, 11:59 pm

1 Introduction

This assignment will require you to implement several algorithms for offline reinforcement learning. First, given a dataset of exploration data, you will implement conservative Q-learning (CQL). You will then implement several policy-based methods for offline RL, including reward-weighted regression (RWR), advantage-weighted regression (AWR), and advantage-weighted actor-critic (AWAC).

1.1 File overview

The starter code for this assignment can be found at

`https://github.com/berkeleydeeprlcourse/homework_fall2021/tree/master/hw5`

We will be building on the code that we have implemented in the first four assignments, primarily focusing on code from Homework 3. All files needed to run your code are in the `hw5` folder, but there will be some blanks you will fill with your previous solutions. Places that require previous code are marked with `# TODO` and are found in the following files:

- `infrastructure/utils.py`
- `infrastructure/rl_trainer.py`
- `policies/MLP_policy.py`
- `policies/argmax_policy.py`
- `critics/dqn_critic.py`

In order to complete this assignment, you will be writing new code in the following files:

- `critics/cql_critic.py`
- `agents/explore_or_exploit_agent.py`
- `agents/awac_agent.py`
- `policies/MLP_policy.py`

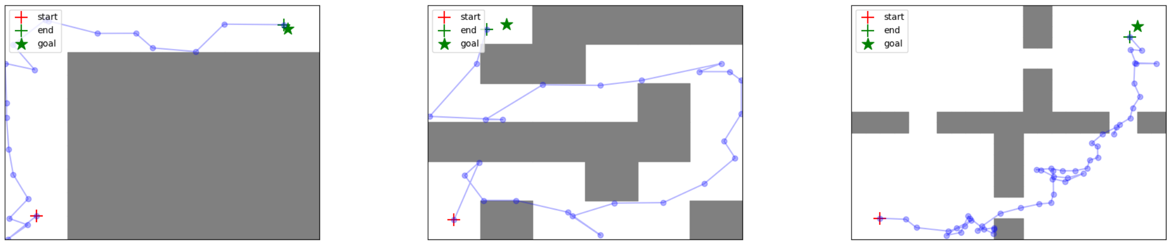

Figure 1: Figures depicting the `easy` (left), `medium` (middle) and `hard` (right) environments.

1.2 Environments

Unlike previous assignments, we will consider some stochastic dynamics, discrete-action gridworld environments in this assignment. The three gridworld environments you will need for the graded part of this assignment are of varying difficulty: `easy`, `medium` and `hard`. A picture of these environments is shown below. The easy environment requires following two hallways with a right turn in the middle. The medium environment is a maze requiring multiple turns. The hard environment is a four-rooms task which requires navigating between multiple rooms through narrow passages to reach the goal location. We also provide a very hard environment for the bonus (optional) part of this assignment.

1.3 Random Network Distillation (RND) Algorithm

You will not be required to implement RND for this assignment (it will be provided for you), but a description of the algorithm is included here for completeness.

A common way of doing exploration is to visit states with a large prediction error of some quantity, for instance, the TD error or even random functions. The RND algorithm aims at encouraging exploration by asking the exploration policy to more frequently undertake transitions where the prediction error of a random neural network function is high. Formally, let $f^*_\theta(s')$ be a randomly chosen vector-valued function represented by a neural network. RND trains another neural network, $\hat{f}_\phi(s')$ to match the predictions of $f^*_\theta(s')$ under the distribution of datapoints in the buffer, as shown below:

$$\phi^* = \arg\min_\phi \ \mathbb{E}_{s,a,s' \sim \mathcal{D}} \left[\underbrace{||\hat{f}_\phi(s') - f^*_\theta(s')||}_{\mathcal{E}_\phi(s')} \right]. \tag{1}$$

If a transition (s, a, s') is in the distribution of the data buffer, the prediction error $\mathcal{E}_\phi(s')$ is expected to be small. On the other hand, for all unseen state-action tuples it is expected to be large. To utilize this prediction error as a reward bonus for exploration, RND trains two critics – an *exploitation critic*, $Q_R(s, a)$, and an *exploration critic*, $Q_\mathcal{E}(s, a)$, where the exploitation critic estimates the return of the policy under the actual reward function and the exploration critic estimates the return of the policy under the reward bonus. In practice, we normalize error before passing it into the exploration critic, as this value can vary widely in magnitude across states leading to poor optimization dynamics.

In this problem, we represent the random functions utilized by RND, $f^*_\theta(s')$ and $\hat{f}_\phi(s')$ via random neural networks. To prevent the neural networks from having zero prediction error right from the beginning, we initialize the networks using two different initialization schemes marked as `init_method_1` and `init_method_2` in `exploration/rnd_model.py`.

In practice, we use a variant of RND exploration that will not utilize the exploration reward and the environment reward separately , but will use a combination of both rewards for exploration as compared to performing fully. That is, the exploration critic uses a weighted sum of the RND bonus and the environment reward of the form:

$$r_{\text{mixed}} = \text{explore\_weight} \times r_{\text{explore}} + \text{exploit\_weight} \times r_{\text{env}}$$

The weighting is controlled in `agents/explore_or_exploit_agent.py`. The exploitation critic is only trained on the environment reward and is used for evaluation.

1.4 Conservative Q-Learning (CQL) Algorithm

For the first portion of the offline RL part of this assignment, we will implement the conservative Q-learning (CQL) algorithm that augments the Q-function training with a regularizer that minimizes the soft-maximum of the Q-values $\log\left(\sum_a \exp(Q(s, a))\right)$ and maximizes the Q-value on the state-action pair seen in the dataset, $Q(s, a)$. The overall CQL objective is given by the standard TD error objective augmented with the CQL regularizer weighted by α: $\alpha \left[\frac{1}{N} \sum_{i=1}^N \left(\log\left(\sum_a \exp(Q(s_i, a))\right) - Q(s_i, a_i) \right) \right]$. You will tweak this value of α in later questions in this assignment.

1.5 RWR, AWR and AWAC Algorithms

For the second portion of the offline RL part of this assignment, we will implement several policy-based offline RL algorithms. These algorithms (RWR, AWR, AWAC) all have updates similar to policy gradient. The actor updates are given below:

RWR:

$$\theta \leftarrow \arg\max_{\theta} \ \mathbb{E}_{s,a\sim\mathcal{B}}\left[log\pi_\theta(a|s)exp(\frac{1}{\lambda}\mathcal{R}(s,a))\right]. \tag{2}$$

AWR:

$$\theta \leftarrow \arg\max_{\theta} \ \mathbb{E}_{s,a\sim\mathcal{B}}\left[log\pi_\theta(a|s)exp(\frac{1}{\lambda}[\mathcal{R}(s,a) - \mathcal{V}^{\pi_k}(s)])\right]. \tag{3}$$

AWAC:

$$\theta \leftarrow \arg\max_{\theta} \ \mathbb{E}_{s,a\sim\mathcal{B}}\left[log\pi_\theta(a|s)exp(\frac{1}{\lambda}\mathcal{A}^{\pi_k}(s,a))\right]. \tag{4}$$

Where $\mathcal{R}(s,a) = \sum_t[\gamma^t r_t]$ is the total return of the trajectory containing (s,a), and $\mathcal{A}^{\pi_k}(s,a)$ is the advantage of action a in state s.

This update is similar to weighted behavior cloning (which it resolves to if the Q function is degenerate). But with a well-formed Q estimate, we weight the policy towards only good actions. In the update above, the agent regresses onto high-advantage actions with a large weight, while almost ignoring low-advantage actions.

Note that for AWAC, the value and advantage estimates are computed using a Q-network trained to minimize the standard TD loss. Recall that:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s), V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s,a)] \tag{5}$$

1.6 Implementation

For this assignment, you will implement the algorithms as described above. Look for the `# TODO` markers in the files listed above for detailed implementation instructions. You may also want to add additional logging to understand the magnitude of Q-values, etc, to help debugging.

1.7 Evaluation

Once you have a working implementation of CQL, RWR, AWR, and AWAC, you should prepare a report. The report should consist of one figure for each question below (each part has multiple questions). You should turn in the report as one PDF and a zip file with your code. If your code requires special instructions or dependencies to run, please include these in a file called `README` inside the zip file.

1.8 Problems

Part 1: Offline Learning with CQL We have provided an implementation of RND for collecting exploration data that is (likely) useful for performing exploitation. You will perform offline RL on this dataset and see how close the resulting policy is to the optimal policy. To begin, you will implement the conservative Q-learning algorithm in this question which primarily needs to be added in `critic/cql_critic.py` and you need to use the CQL critic as the extrinsic critic in `agents/explore_or_exploit_agent.py`. Once CQL is implemented, you will evaluate it and compare it to a standard DQN critic. Run this part using:

```
python rob831/scripts/run_hw5_expl.py --env_name PointmassMedium-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=0.0 --offline_exploitation
--exp_name q1_medium_dqn

python rob831/scripts/run_hw5_expl.py --env_name PointmassMedium-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=1.0 --offline_exploitation
--exp_name q1_medium_cql
```

```
python rob831/scripts/run_hw5_expl.py --env_name PointmassHard-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=0.0 --offline_exploitation
--exp_name q1_hard_dqn
```

```
python rob831/scripts/run_hw5_expl.py --env_name PointmassHard-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=1.0 --offline_exploitation
--exp_name q1_hard_cql
```

Hint: Examine the performance of CQL when utilizing a transformed reward function for training the exploitation critic. Do not change any code in the environment class, instead make this change in `agents/explore_or_exploit_agent.py`. The transformed reward function is given by:

$$\tilde{r}(s,a) = (r(s,a) + \text{shift}) \times \text{scale}$$

The choice of shift and scale is up to you, but we used shift = 1, and scale = 100. On any one domain of your choice test the performance of CQL with this transformed reward. Is it better or worse? What do you think is the reason behind this difference in performance, if any?

Evaluate this part on the `medium` and `hard` environments. As a debugging hint, for the hard environment, with a reward transformation of scale = 100 and shift = 1, you should find that CQL is better than DQN. For these experiments, compare the performance of CQL to that of DQN. Include the learning curves for both DQN and CQL-based exploitation critics on these environments in your report.

Part 2: Offline Learning with RWR Similar to part 1 above, implement reward-weighted regression. The changes here primarily need to be added to `agents/awac_agent.py` and `policies/MLP_policy.py`. For simplicity, the methods in parts 2-4 share the same agent and update rule, but with different advantage estimators.

Once you have implemented RWR, we will test the algorithm on two Pointmass environments. We will also need to tune the λ value in the RWR update, which controls the conservatism of the algorithm. Consider what this value signifies and how the performance compares to BC and DQN given different λ values.

Below are some commands that you can use to test your code. You should expect to see a return of above -60 for the PointmassMedium task and above -30 for PointmassEasy.

```
python rob831/scripts/run_hw5_awac.py --env_name PointmassMedium-v0
--exp_name q2_rwr_medium_lam{0.1,1,2,10,20,50} --use_rnd
--offline_exploitation --awac_lambda={0.1,1,2,10,20,50} --num_exploration_steps=20000 --rwr

python rob831/scripts/run_hw5_awac.py --env_name PointmassEasy-v0
--exp_name q2_rwr_easy_lam{0.1,1,2,10,20,50} --use_rnd
--offline_exploitation --awac_lambda={0.1,1,2,10,20,50} --num_exploration_steps=20000 --rwr
```

Part 3: Offline Learning with AWR Similar to part 2, implement and run AWR in the two Pointmass environments:

```
python rob831/scripts/run_hw5_awac.py --env_name PointmassMedium-v0
--exp_name q3_awr_medium_lam{0.1,1,2,10,20,50} --use_rnd
--offline_exploitation --awac_lambda={0.1,1,2,10,20,50} --num_exploration_steps=20000 --awr

python rob831/scripts/run_hw5_awac.py --env_name PointmassEasy-v0
--exp_name q3_awr_easy_lam{0.1,1,2,10,20,50} --use_rnd
--offline_exploitation --awac_lambda={0.1,1,2,10,20,50} --num_exploration_steps=20000 --awr
```

Part 4: Offline Learning with AWAC Similar to part 2, implement and run AWAC in the two Pointmass environments:

```
python rob831/scripts/run_hw5_awac.py --env_name PointmassMedium-v0
--exp_name q4_awac_medium_lam{0.1,1,2,10,20,50} --use_rnd
--offline_exploitation --awac_lambda={0.1,1,2,10,20,50} --num_exploration_steps=20000
```

```
python rob831/scripts/run_hw5_awac.py --env_name PointmassEasy-v0
--exp_name q4_awac_easy_lam{0.1,1,2,10,20,50} --use_rnd
--offline_exploitation --awac_lambda={0.1,1,2,10,20,50} --num_exploration_steps=20000
```

In your report, please report your learning curves for each of these parts and report the optimal λ for each algorithm. Please also describe how the λ parameter should theoretically affect the performance of these algorithms and whether this effect is observed in practice. Also please consider lambda values outside of the range suggested above and consider how it may affect performance both empirically and theoretically. In addition, compare the performance of the offline learning algorithms (CQL, RWR, AWR, AWAC).

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 `data` with all the experiment runs from this assignment. **Do not change the names originally assigned to the folders, as specified by `exp_name` in the instructions. Video logging is not utilized in this assignment, as visualizations are provided through plots, which are outputted during training.**
- The `rob831` folder with all the `.py` files, with the same names and directory structure as the original homework repository (excluding the `data` folder). Also include any special instructions we need to run in order to produce each of your figures or tables (e.g. "run python myassignment.py -sec2q1" to generate the result for Section 2 Question 1) 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_`, `q2_`, `q3_`, etc.**

```
submit.zip
├─ data
│  ├─ q1...
│  │  └─ events.out.tfevents.1567529456.e3a096ac8ff4
│  ├─ q2...
│  │  └─ events.out.tfevents.1567529456.e3a096ac8ff4
│  └─ ...
├─ rob831
│  ├─ agents
│  │  ├─ explore_or_exploit_agent.py
│  │  ├─ awac_agent.py
│  │  └─ ...
│  ├─ policies
│  │  └─ ...
│  └─ ...
├─ README.md
└─ ...
```

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 **HW5 Code**, and upload the PDF of your report to **HW5**.