CS224R Spring 2023 Homework 3
Offline RL

Due 5/17/2023

SUNet ID:
Name:
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Overview

Goals: In this assignment you will implement two offline reinforcement learning algorithms: Implicit Q-Learning and Conservative Q-Learning. You'll be experimenting with different hyper-parameters and offline datasets that have been provided to you. Data collection is one of the most practical problems with applying Deep RL, and it's a common practice to try different exploration strategies for a given problem. In this assignment, we provide with you Random and Random Network Distillation based exploration strategies for you to compare the quality of the collected data.

You will then have the opportunity to train and tune these agents in a variety of environments in the MuJoCo physics simulator (`https://mujoco.org/`). We provide the code for generating the offline datasets as part of the assignment. Your objectives are as follows:

1. Implement and train the Conservative Q-Learning and the Implicit Q-Learning algorithms for Offline Reinforcement Learning.
2. Experiment with the key hyperparameters for each algorithm and explore how they affect performance of your RL agents.
3. Analyze differences between the quality of different offline data samples provided to you.

Submitting the PDF: Fill in your responses in the `answer{}` tags provided in the Tex template. Submit all the requested values in tables, and put in all requested plots/images as Tex figures. You should also include all your reasoning and text responses in the PDF.

Submitting the Code and Experiment Runs: In order to turn in your code and experiment logs, create a folder that contains the following:

- data/ folder with all logged runs corresponding to both problem 1 and 2.
- The cs224r folder with all the .py files, with the same names and directory structure as the original homework repository.

Gradescope: Submit both the PDF and the code and experiment runs in the appropriate assignment on Gradescope. An autograder will be provided to evaluate the performance of your policies from the generated tensorboard files.

Use of GPT/Codex/Copilot: For the sake of deeper understanding on implementing imitation learning methods, assistance from generative models to write code for this homework is prohibited.

Codebase

We provide you with offline transitions dataset $\mathcal{D}$. Assuming your replay buffer has been populated with the offline dataset trajectories, your goal is to implement the corresponding reinforcement learning algorithms.

For each problem, we provide you with the files that you need to complete. Sections that need to be filled have been marked with `TODO` tags.

For this assignment, you can use the **EC c4.4xlarge** AWS instance (the instance you used for homewor 2 part1). Please follow the AWS Guide for the set-up instructions. Here are the installation steps once you're on the instance. Instructions for creating the `cs224r` conda environment can be followed from homework 1.

```
conda activate cs224r
sudo apt-get install swig
pip install -r requirements.txt
pip install -e .
```

Preliminaries

Environments: In this assignment, we'll be working with the Pointmass environment variants.

The goal for the Pointmass environment is to navigate a gridworld of varying difficulties: `easy`, `medium` and `hard` to reach the 'goal' location. Sample environments for each difficulty have been visualized in Figure 1

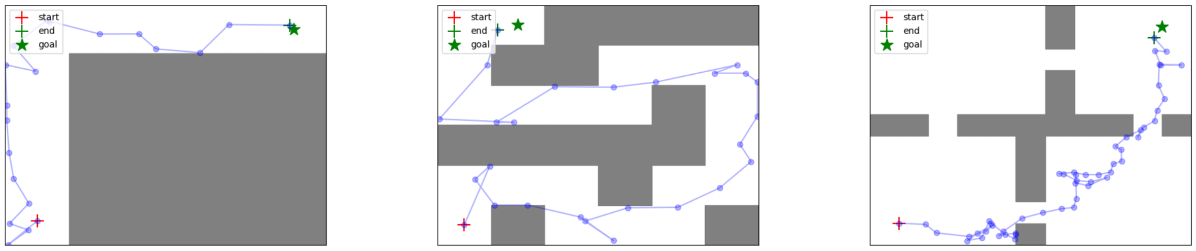

Figure 1: Visualization of the Point mass enivorment with varying difficulty levels for navigation and reaching the goal.

Offline datasets: For offline RL, we have provided you with a set of exploration strategies which are used to populate the replay buffer. These trajectories then serve as the training dataset for your offline RL algorithms. In this assignment, you'll experiment with two exploration strategies (i) a Random $\mathcal{E}$-Greedy strategy and (ii) Random Network Distillation (RND) algorithm.

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:

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

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. n 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.

Problem 1: Implicit Q-Learning

1. In this problem, you'll implement the Implicit Q-Learning (IQL) method for offline RL. The actor update for IQL incorporates the advantage function similar to the actor update in the Advantage-Weighted Actor Critic (AWAC) algorithm. This an advantage-weighted negative log likelihood loss function:

$$L_\pi(\psi) = -\mathbb{E}_{s,a\sim\mathcal{B}}\left[\log \pi_\psi(a \mid s) \exp\left(\frac{1}{\lambda}\mathcal{A}^{\pi_k}(s,a)\right)\right] \tag{2}$$

where $\mathcal{B}$ represents samples sampled from the dataset (behavior policy that was used to collect the data).

The actor update in AWAC corresponds to weighted maximum likelihood, where the targets are updated by reweighting the state-action pairs observed in the current dataset by the predicted advantages from the learned critic. The Q function is learnt with a Temporal Difference (TD) loss. The objective function is given below:

$$\mathbb{E}_D\left[\left(Q(s,a) - r(s,a) + \gamma\mathbb{E}_{s',a'}\left[Q_{\phi_{k-1}}(s',a')\right]\right)^2\right] \tag{3}$$

IQL modifies the actor critic update to use expectile regression. The expectile τ of a random variable X is defined as:

$$\arg\min_{m_\tau} \mathbb{E}_{x\sim X}\left[L_2^\tau(x - m_\tau)\right] \tag{4}$$

$$L_2^\tau(\mu) = |\tau - \mathbb{1}\{\mu \le 0\}|\mu^2 \tag{5}$$

That is for $\tau > 0.5$, this asymmetric loss function downweights the contributions of x values smaller than m_τ, while giving more weights to large values as visualized in Figure 2.

The offline dataset might contain different outcomes from similar states; in standard Q learning, we would want the critic to learn the expected value for any particular state-action pair. IQL, instead of learning the expected value, learns a given percentile. Our goal is to predict an upper expectile of the TD targets that approximate high values of $r(s,a) + \gamma * Q_\theta(s',a')$ which are present in the support of the offline dataset.

To perform this expectile regression, we need a separate parametric value function V_ϕ. This is allows the optimization process to be differentiable, since optimizing with a single parametric q-function implies incorporating the transition dynamics as $s' \sim p(\cdot|s,a)$. (Note the expectation over (s',a') in Equation 3. Finally, the critic is **only**

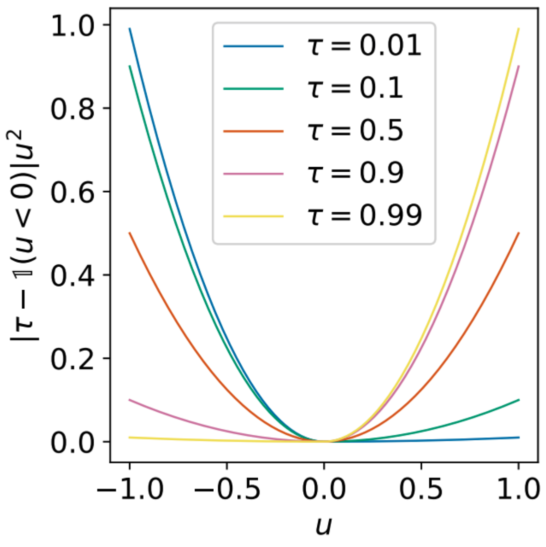

Figure 2: Expectile loss function visualized for different values of τ.

updated with actions that were seen in the offline dataset, and not on any out of distribution (unseen) sampled actions. This leads to the following loss functions:

$$L_V(\phi) = \mathbb{E}_{(s,a)\sim D}\left[L_2^\tau\left(Q_\theta(s,a) - V_\phi(s)\right)\right] \tag{6}$$

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim D}\left[\left(r(s,a) + \gamma V_\phi\left(s'\right) - Q_\theta(s,a)\right)^2\right] \tag{7}$$

Fill in the TO-DOs in:

- `cs224r/critics/iql_critic.py`
- `cs224r/agents/iql_agent.py`

Experiment with $\tau = 0.5, 0.9$ on the `PointmassEasy-v0` and report the eval average return and standard deviation for both cases. You can look at the final eval trajectories under the saved logs to qualitatively observe your agent's performance. The code would run for 50,000 iterations and would roughly take at most an hour to finish. Attach the `eval_last_traj.png` image denoting your agent's last trajectory for each run. Which τ value performs better and why?

```
python cs224r/scripts/run_iql.py --env_name Pointmass{}-v0 \
--exp_name iql_tau_{}_rnd --use_rnd \
--num_exploration_steps=20000 \
--unsupervised_exploration \
--awac_lambda=1 \
--iql_expectile={}
```

Answer:

Return values for both the experiment runs are provided in the table below:

Table 1: IQL on Pointmass Easy

τ	Eval Average Return	Eval Std. Dev
-	-	-

2. Compare the learnt policies for the RND and Random exploration on both the PointmassMedium-v0 environment using the best τ value from the previous runs. Report the eval average return and standard deviation. What does the performance gap between the two exploration strategies indicate about the collected data in case of offline RL?

   Remove the `-use_rnd` flag to run with random exploration:

```
python cs224r/scripts/run_iql.py --env_name Pointmass{}-v0 \
--exp_name iql_tau_{}_random \
--num_exploration_steps=20000 \
--unsupervised_exploration \
--awac_lambda=1 \
--iql_expectile={}
```

   Attach the `eval_last_traj.png` image denoting your agent's last trajectory for each run.

   Answer:
   ...

Problem 2: Conservative Q-Learning

1. In this problem, you'll be implementing the Conservative Q-Learning (CQL) algorithm. The goal of CQL is to prevent overestimation of the policy value. The overall CQL objective is given by the standard TD error objective augmented with the CQL regularizer weighted by $\alpha : \alpha \left[\frac{1}{N} \sum_{i=1}^{N} \left(\log\left(\sum_a \exp\left(Q\left(s_i, a\right)\right)\right) - Q\left(s_i, a_i\right)\right)\right]$.

   Fill in the TODOs in the following files:

   - `cs224r/critics/cql_critic.py`

   Once you've filled in all of the TODO commands, you should be able to run CQL with the following command -

   ```
   python cs224r/scripts/run_cql.py --env_name Pointmass{}-v0 \
       --exp_name cql_alpha_{}_rnd \
       --use_rnd --unsupervised_exploration \
       --offline_exploitation --cql_alpha={}
   ```

   You can look at the final eval trajectories under the saved logs to qualitatively observe your agent's performance. The code would run for 50,000 iterations and would roughly take about an hour to finish. Attach the `eval_last_traj.png` image denoting your agent's last trajectory for each run.

   Experiment with $\alpha = 0, 0.1$ on the `PointmassMedium-v0` and report the Eval average return and standard deviation for both cases on the PointmassEasy-v0 environment. What does $\alpha = 0.0$ signify in terms of the algorithm? Explain the difference in obtained performance gap based on α.

   Answer:

2. Compare the learnt policies for the RND and Random exploration on the `PointmassMedium-v0` environment with $\alpha = 0.1$ by reporting the eval average return and standard deviation. Remove the `-use_rnd` flag to run with random exploration:

   ```
   python cs224r/scripts/run_cql.py --env_name Pointmass{}-v0 \
       --exp_name cql_alpha_{}_random \
       --unsupervised_exploration \
       --offline_exploitation --cql_alpha={}
   ```

   Attach the `eval_last_traj.png` image denoting your agent's last trajectory for each run.

   Answer: