# AAI4160 Homework 5: Offline RL

[Kim KyungHwan]
[2022122076]

## 1 Introduction

In this assignment, you will implement one of the fundamental algorithms in offline reinforcement learning: Conservative Q-Learning (CQL). You will be experimenting with different hyperparameters and offline datasets that have been provided to you. In this assignment, we provide a random policy and an expert policy for you to compare the performance according to the quality of the collected data.

Your objectives are as follows:

1. Implement and train an offline reinforcement learning algorithm, CQL
2. Experiment with the key hyperparameters and explore how they affect performance of your RL agent
3. Analyze differences between the quality of different offline datasets, Random and Expert

**Writing the report**: Provide your responses for questions with `%TODO` tags in this LaTeX template. Submit all the requested values in tables, and put in all requested plots/images. You should also include all your reasoning and text responses in the PDF.

## 2 Codebase

In this assignment, you mainly work with the following codes:

- `aai4160/scripts/run_cql.py`: the main training loop for your CQL implementation.
- `aai4160/critics/cql_critic.py`: the CQL learner you will implement.

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

You can install dependencies with the following script:

```
conda activate [your_env]
pip install -r requirements.txt
pip install -e .
```

## 3 Tips for experiments

In this homework, our code is computationally light; if you use a RTX 3090 GPU, the GPU utilization will be up to 20%. To utilize the full power of GPUs, we recommend you to run multiple jobs simultaneously. You can do it by (1) executing multiple bash terminals and execute the training script for each bash terminal; (2) using "&" command to run the jobs in background; or (3) using tmux. We recommend trying tmux since it is one of the powerful utility tools that you can use inside linux servers. One of the advantages of

tmux is that your work status would be saved inside a tmux session, so it will keep running even if you loose the ssh connection.

# 4 Preliminaries

**Environments:** In this assignment, you will be working with the Pointmass environment. The goal for the Pointmass environment is to navigate a grid world of varying difficulties: `easy`, `medium`, and `hard` to reach the 'goal' location. Sample environments for each difficulty have been visualized in Figure 1. The observation consists of $(y, x)$-location of the point mass and the action space is $5$ discrete numbers indicating `NO MOVE`, `LEFT`, `RIGHT`, `UP`, and `DOWN`. The agent receives $1$ reward and terminates when the agent reaches the goal, otherwise receives reward of $0$.

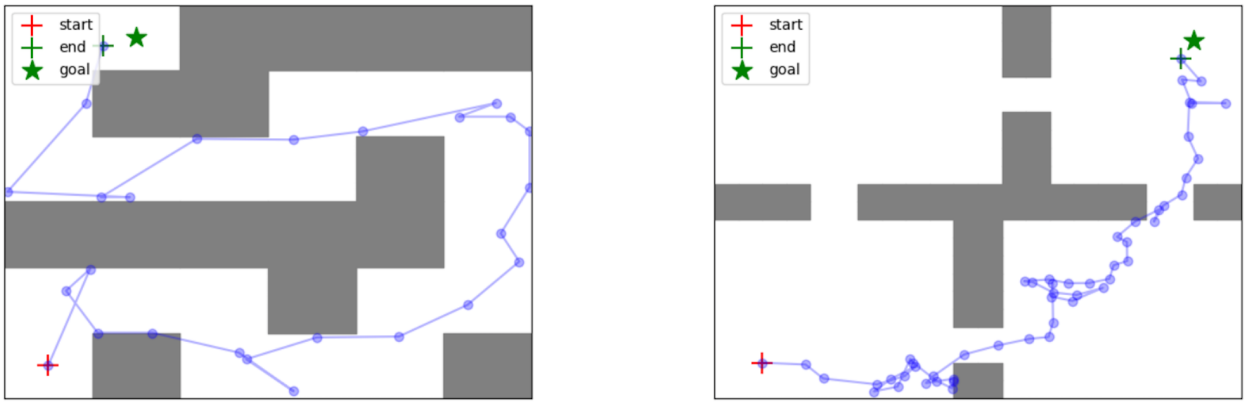


Figure 1: Visualization of the Pointmass environments with `Medium` (left) and `Hard` (right) difficulties for navigation and reaching the goal.

**Offline datasets:** For offline RL, we have provided you with a set of data collection strategies which are used to populate the replay buffer for the first $20000$ environment steps. These trajectories then serve as the training dataset for your offline RL algorithms. The offline transitions dataset $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})_i\}$ consists of tuples of the current state, action, reward, and future state. In this assignment, you will experiment with two datasets generated by (1) random policy and (2) expert policy (optimal actions with $\epsilon$-greedy, where $\epsilon = 0.3$).

# 5 Conservative Q-Learning

## 5.1 Implement CQL (5 points)

In this problem, you'll be implementing the Conservative Q-Learning (CQL) algorithm (a DQN version for the discrete environment). 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 $\alpha$:

$$L_Q(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\left(Q(\mathbf{s}_i, \mathbf{a}_i) - \left(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'}[Q(\mathbf{s}'_i, \mathbf{a}')]\right)\right)^2\right] + \alpha\frac{1}{N}\sum_{i=1}^{N}\left[\log\left(\sum_{\mathbf{a}}\exp\left(Q\left(\mathbf{s}_i, \mathbf{a}\right)\right)\right) - Q\left(\mathbf{s}_i, \mathbf{a}_i\right)\right]. \tag{1}$$

Fill in the TODOs in the following file:

- `aai4160/critics/cql_critic.py`

Once you've filled in all of the TODO commands, you should be able to run CQL on the **expert** dataset with the following command:

```
python aai4160/scripts/run_cql.py --env_name PointmassHard-v0 \
    --exp_name cql_alpha_{$ALPHA}_expert \
    --dataset_name expert --cql_alpha={$ALPHA}
```

Experiment with $\alpha = \{0.0, 0.1\}$ in `expert` dataset. You can look at the final eval trajectories under the saved logs to qualitatively observe your agent's performance. Your CQL implementation should achieve about at least $0.9$ in `expert` dataset with $\alpha = 0.1$ in `Hard` difficulty. The code will run for $50000$ iterations and it roughly takes about $30$ minutes to finish. Also, attach the `eval_last_traj.png` image, which visualizes your agent's last trajectory, and the *overestimation* value curve, $\frac{1}{N}\sum_{i=1}^{N}\left[\log\left(\sum_{\mathbf{a}}\exp\left(Q\left(\mathbf{s}_i, \mathbf{a}\right)\right)\right) - Q\left(\mathbf{s}_i, \mathbf{a}_i\right)\right]$.

**What does $\alpha = 0.0$ signify in terms of the algorithm (hint: think about an algorithm on top of which we developed CQL)? Explain the difference in the performance gap and the overestimation value with respect to $\alpha$.**

**Answer: Using cql loss gives better performance and lower overestimation values compare to naive loss. Since naive loss(alpha=0) is same as loss of naive Q learning in this hw, it will overestimate reward of OOD actions more often than CQL which encourages to choose undesired actions. On the other hand, CQL loss encourages to push down maximum of Q values while push up Q values for pair inside dataset which deals with overestimation and selecting action(given state) inside dataset.**

Table 1: Average Performance with different $\alpha$ in expert dataset. **Fill data and replace the caption.**

| | Mean | Standard Deviation |
|---|---|---|
| $\alpha = 0.0$ | **0.272** | **0.445** |
| $\alpha = 0.1$ | **1** | **0** |

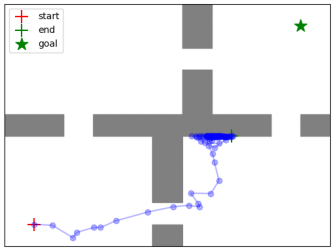


Figure 2: $\alpha = 0.0$

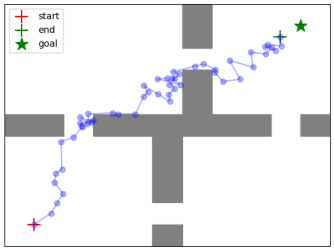


Figure 3: $\alpha = 0.1$

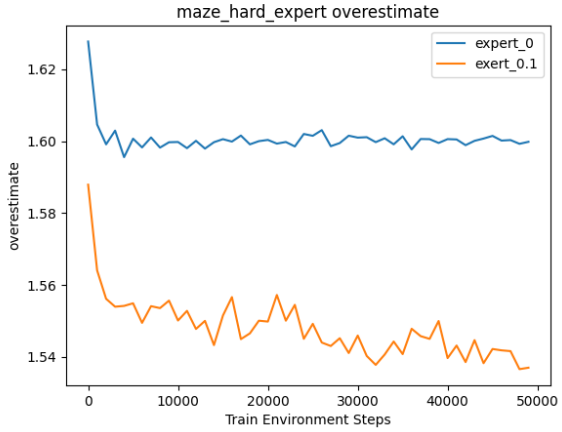


Figure 4: **overestimation of $Q$ values $\alpha = 0.0, 0.1$.**

## 5.2 Comparison between Expert dataset and Random dataset (5 points)

Compare the learned policies for the expert dataset and random dataset on the `PointmassHard-v0` environment with $\alpha = \{0.0, 0.1\}$ by reporting the eval average return and standard deviation. Execute the below command to execute CQL with **random** dataset:

```
python aai4160/scripts/run_cql.py --env_name PointmassHard-v0 \
    --exp_name cql_alpha_{$ALPHA}_random \
    --dataset_name random --cql_alpha={$ALPHA}
```

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

**You might have a result that contradicts with your intuition; specifically, $\alpha = 0.0$ works better than $\alpha = 0.1$ with the random dataset. Explain the difference in obtained performance gap on the dataset with the data-collecting policy.**

**Answer: CQL loss encourages Q function to have higher value on actions(given state) inside dataset which leads agent to choose those actions(say randomA). Now, since state space is large, Q function can't learn all the state,action pair. When Q function receives unseen state which is similiar to state it learned, then argmax of Q(s,a) will be randomA which is not desired action since it is random. In short, using cql loss with data coming from random policy prevents agent from learning expert and exploration(too pessimistic) while naive Q learning encourages exploration.**

Table 2: Average Performance with different datasets and $\alpha$. **Fill**

| | Mean | Standard Deviation |
|---|---|---|
| `Random;` $\alpha = 0.0$ | **0.947** | **0.223** |
| `Random;` $\alpha = 0.1$ | **0.769** | **0.421** |
| `Expert;` $\alpha = 0.0$ | **0.272** | **0.445** |
| `Expert;` $\alpha = 0.1$ | **1** | **0** |

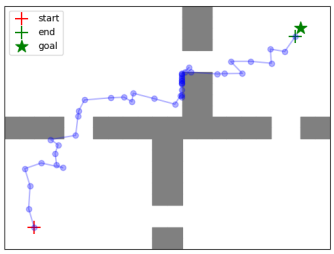


Figure 5: `random`$\alpha = 0.0$

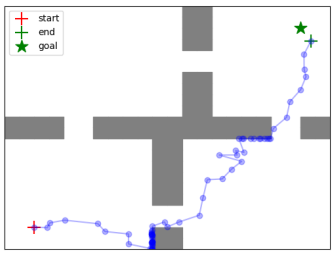


Figure 6: `random` $\alpha = 0.1$

## 6 Discussion

Please provide us a rough estimate, in hours, for each problem, how much time you spent. This will help us calibrate the difficulty for future homework.

- **Conservative Q-Learning**: **XX hours**

Feel free to share your feedback here, if any: **We would really appreciate your feedback to improve the reinforcement learning class.**

## 7 Submission

Please include the code, tensorboard logs data, and the **"report"** in a single zip file, `hw5_[YourStudentID].zip`. The zip file should be smaller than 20MB. The structure of the submission file should be:

```
hw5_[YourStudentID].zip
├── hw5_[YourStudentID].pdf
├── aai4160/
│   ├── critics/
│   │   ├── cql_critic.py
│   │   └── ...
│   └── ...
└── data/
    └── tensorboard log folders
```