CS224R Spring 2025 Homework 3
Offline RL

Due 5/16/2025

SUNet ID: agliang
Name: Agnes Liang
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 Simple PointMass physics simulator. 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.

Notice that Problem 1 may take longer time to finish than Problem 2.

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. **Note: Please remove any redundant run folders, and only keep your best run for each**

hyper-parameter configuration in the `data/` folder. Remove any empty/incomplete logs that correspond to interrupted/failed runs.

- `cs224r/` folder with all the .py files, with the same names and directory structure as the original homework repository.

Zip and submit the folder on Gradescope.

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 homework 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.

```
chmod 777 ./setup_env.sh
conda activate cs224r
pip install -r requirements.txt
pip install -e .
```

Preliminaries

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

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

`Homework3/figures/point_mass_envs.png`

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:

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

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. 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:

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

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

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

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

Homework3/figures/expectile_example.png

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

That is for $\zeta > 0.5$, this asymmetric loss function downweights the contributions of x values smaller than m_ζ, 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** 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:

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

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

Fill in the TO-DOs in:

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

Experiment with $\zeta = 0.2, 0.9$ 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 ζ value performs better and why?

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

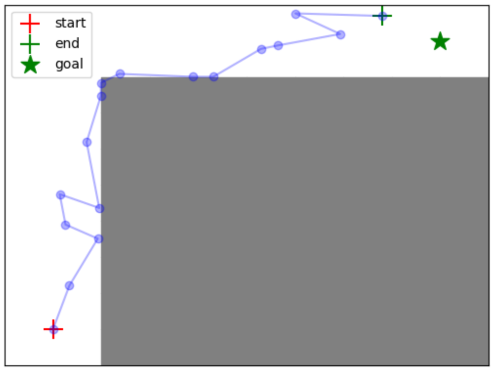

Figure 3: 0.2

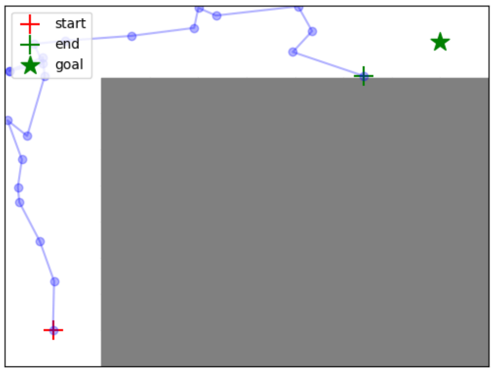

Figure 4: 0.9

While $\zeta = 0.2$ focused more on the highest-value actions in the dataset, the agent trained with $\zeta = 0.9$ performed slightly better overall. It also showed more consistent results, as seen from the lower standard deviation. This suggests that $\zeta = 0.9$ helped the agent make more stable decisions, even if they were a bit more cautious. Overall, $\zeta = 0.9$ gave better and more reliable performance in this environment.

Expectile (ζ)	**Eval Average Return**	**Eval Std. Deviation**
0.2	-20.8	8.13
0.9	-19.3	5.40

Table 1: Evaluation performance of IQL on `PointmassEasy-v0` for different expectile values after 50,000 iterations.

2. Compare the learnt policies for the RND and Random exploration on the `PointmassMedium-v0` environment using the best ζ 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 PointmassMedium-v0 \
--exp_name iql_zeta_{enter_your_zeta}_random \
--num_exploration_steps=20000 \
--unsupervised_exploration \
--awac_lambda=1 \
--iql_expectile={enter_your_zeta}
```

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

   Answer:

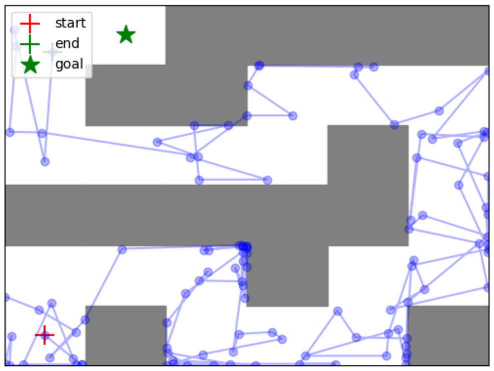

Figure 5: 0.9 Random Exploration

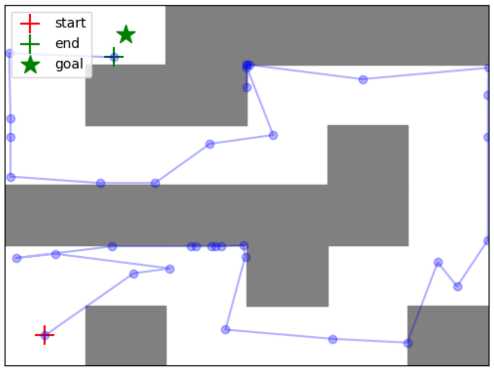

Figure 6: 0.9 RND

Exploration Strategy	Eval Average Return	Eval Std. Deviation
Random	-84.33	31.47
RND	-36.33	10.84

Table 2: Evaluation performance of IQL on `PointmassMedium-v0` using $\zeta = 0.9$ with different exploration strategies. RND leads to significantly better and more stable policy learning.

The performance gap between RND and random exploration indicates that the quality of the collected offline data has a significant impact on policy learning. RND exploration encourages the agent to visit novel states by adding a reward bonus based on prediction error, leading to a more diverse and informative dataset. As a result, the policy trained on RND data achieves a much higher average return and lower variance. In contrast, random exploration generates more uniform or redundant trajectories, which are less helpful for learning. This shows that smarter exploration strategies, even in offline settings, can greatly improve the effectiveness of reinforcement learning algorithms.

Problem 2: Conservative Q-Learning (Updated)

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 PointmassHard-v0 \
    --exp_name cql_alpha_{enter_your_alpha}_rnd \
    --use_rnd --unsupervised_exploration \
    --offline_exploitation --cql_alpha={enter_your_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 `PointmassHard-v0` and report the Eval average return and standard deviation for both cases on the PointmassHard-v0 environment. What does $\alpha = 0.0$ signify in terms of the algorithm? Explain the difference in obtained performance gap based on α (Performance may vary by different random seeds, please run the experiment 3 times and report all runs).

   Answer:

   $\alpha = 0.0$

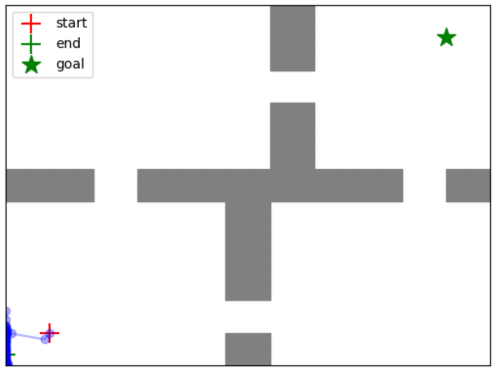

Figure 7: Run 1

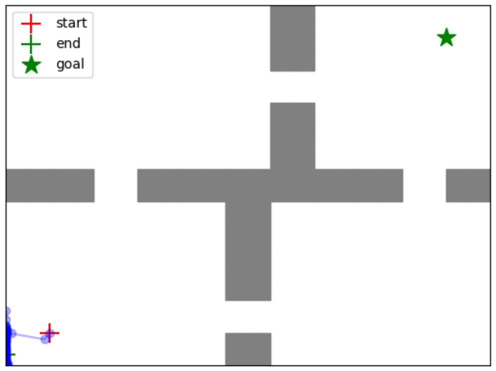

Figure 8: Run 2

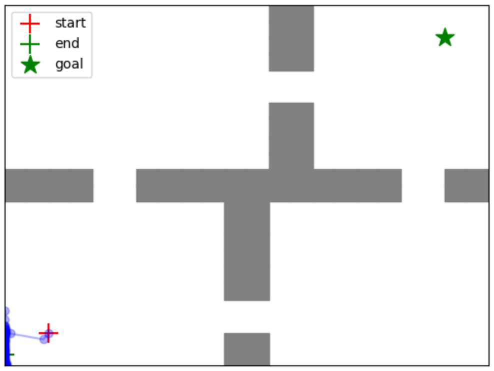

Figure 9: Run 3

$\alpha = 0.1$

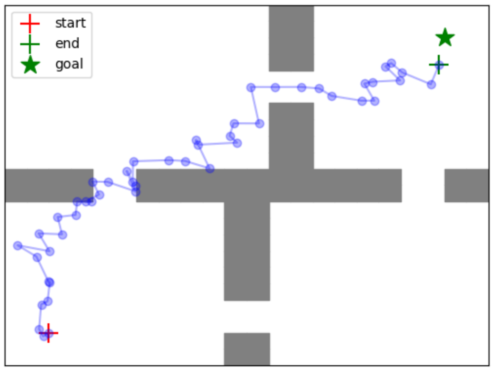

Figure 10: Run 1

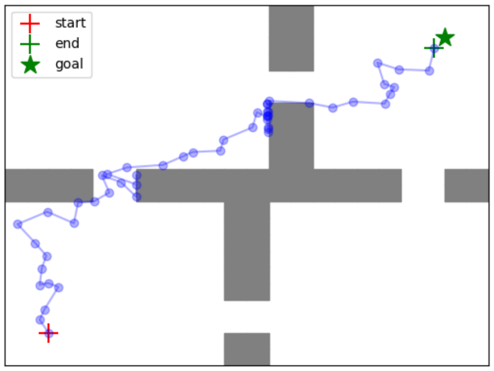

Figure 11: Run 2

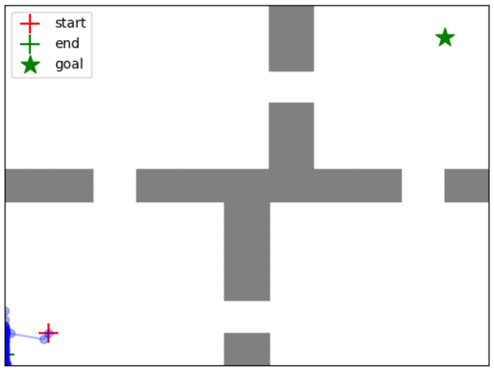

Figure 12: Run 3

α	Run	Eval Average Return	Eval Std. Deviation
0.0	1	-100.0	0.0
0.0	2	-100.0	0.0
0.0	3	-100.0	0.0
0.1	1	-37.9477	6.1379
0.1	2	-37.8807	6.0139
0.1	3	-36.9876	4.9751

Table 3: Evaluation results of CQL on `PointmassHard-v0` for different α values.

When we set $\alpha = 0.0$, the CQL algorithm becomes regular Q-learning without any penalty for overestimating Q-values. This means it can easily trust actions that look good under the learned Q-function, even if those actions were rarely seen in the dataset. Since the algorithm is offline and doesn't collect new data, this can lead to very poor performance—as we saw with returns of -100.0 in all runs. When we set $\alpha = 0.1$, the conservative term is added. This term pushes down the Q-values of actions that are not well supported by the data, helping the policy stay closer to what it has actually seen. As a result, the learned policy is safer and performs much better, with returns around -37 across runs. In conclusion, using a nonzero α helps prevent overestimation and improves offline performance.

2. Compare the learnt policies for the RND and Random exploration on the `PointmassHard-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 PointmassHard-v0 \
    --exp_name cql_alpha_0.1_random \
    --unsupervised_exploration \
    --offline_exploitation --cql_alpha=0.1
```

Attach the `eval_last_traj.png` image denoting your agent's last trajectory for each run (Performance may vary by different random seeds, please run the experiment 3 times and report all runs).

Answer:

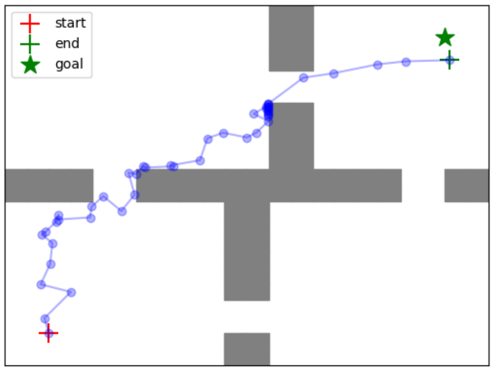

Figure 13: Random Run 1

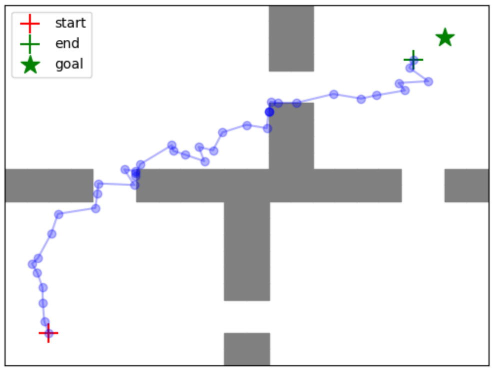

Figure 14: Random Run 2

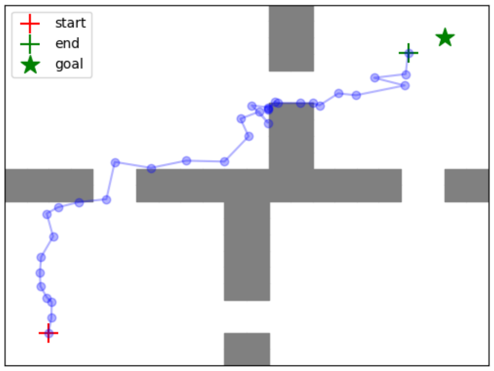

Figure 15: Random Run 3

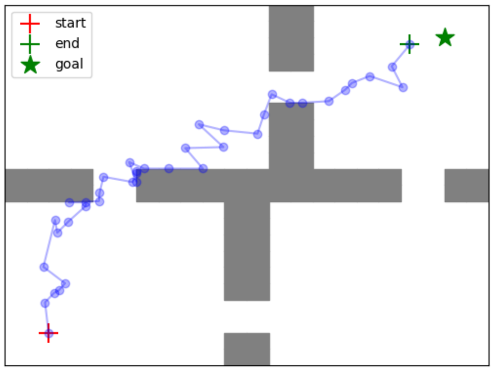

Figure 16: RND Run 1

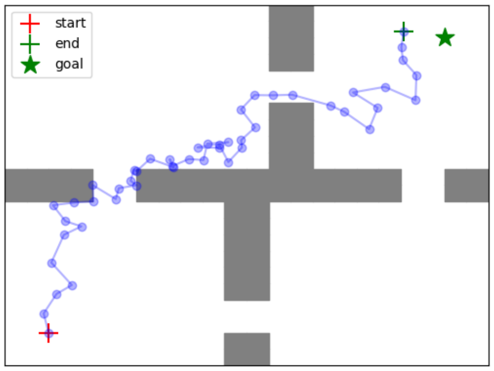

Figure 17: RND Run 2

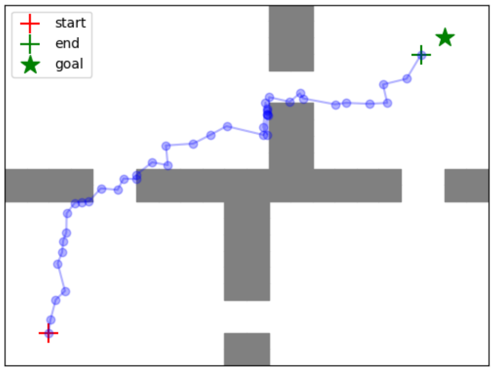

Figure 18: RND Run 3

α	Run	Eval Average Return	Eval Std. Deviation
0.1 (Random)	1	-39.8042	6.7143
0.1 (Random)	2	-41.5687	7.3598
0.1 (Random)	3	-39.368	6.3638
0.1 (RND)	1	-38.5314	6.3111
0.1 (RND)	2	-39.4184	5.645
0.1 (RND)	3	-38.2534	7.4485

Table 4: Evaluation results of CQL on `PointmassHard-v0` comparing Random vs RND exploration with $\alpha = 0.1$.