

Assignment 5: Exploration and Offline Reinforcement Learning

Due November 23, 11:59 pm

1 Introduction

This assignment requires you to implement and evaluate a pipeline for exploration and offline learning. You will first implement an exploration method called random network distillation (RND) and collect data using this exploration procedure, then perform offline training on the data collected via RND using conservative Q-learning (CQL) and finally finetune the resulting policy in the environment. You will also explore with variants of exploration bonuses – where a bonus is provided alongside the actual environment reward for exploration. This assignment would be easier to run on a CPU as we will be using gridworld domains of varying difficulties to train our agents.

1.1 File overview

The starter code for this assignment can be found at

https://github.com/berkeleydeeprlcourse/homework_fall2020/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 solutions from homework 3. These locations are marked with `# TODO: get this from hw3` and are found in the following files:

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

In order to implement RND and CQL, you will be writing new code in the following files:

- `critics/cql_critic.py`
- `exploration/rnd_model.py`
- `agents/explore_or_exploit_agent.py`

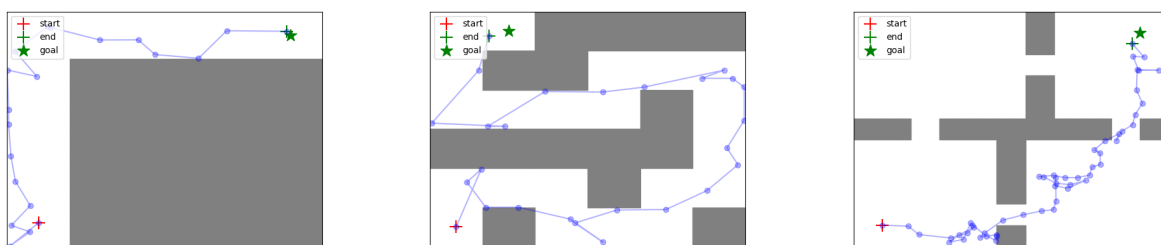


Figure 1: Figures depicting the **easy** (left), **medium** (middle) and **hard** (right) environments.

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

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, as covered in Lecture 13, 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:

$$\phi^* = \arg \min_{\phi} \mathbb{E}_{s,a,s' \sim \mathcal{D}} \left[\underbrace{\|\hat{f}_{\phi}(s') - f_{\theta}^*(s')\|^2}_{\mathcal{E}_{\phi}(s,a)} \right]. \quad (1)$$

If a transition (s, a, s') is in the distribution of the data buffer, the prediction error $\mathcal{E}_{\phi}(s, a)$ 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`.

1.4 Conservative Q-Learning

For 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 $\log(\sum_a \exp(Q(s, a)))$ 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 : $\alpha \left[\sum_{i=1}^N (\log(\sum_a \exp(Q(s_i, a))) - Q(s_i, a_i)) \right]$. You will tweak this value of α in later questions in this assignment.

1.5 Implementation

The first part in this assignment is to implement a working version of Random Network Distillation. The default code will run the `easy` environment with reasonable hyperparameter settings. Look for the `# TODO` markers in the files listed above for detailed implementation instructions.

Once you implement RND, answering some of the questions will require changing hyperparameters, which should be done by changing the command line arguments passed to `run_hw5_expl.py` or by modifying the parameters of the `Args` class from within the Colab notebook.

For the second part of this assignment, you will implement the conservative Q-learning algorithm 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. Finally, you will also need to implement the logic for switching between exploration and exploitation, and controlling for the number of offline-only training steps in the `agents/explore_or_exploit_agent.py` as we will discuss in problems 2 and 3.

1.6 Evaluation

Once you have a working implementation of RND and CQL, 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.

1.7 Problems

Part 1: “Unsupervised” RND and exploration performance. Implement the RND algorithm and use the argmax policy with respect to the exploration critic to generate state-action tuples to populate the replay buffer for the algorithm. In the code, this happens before the number of iterations crosses `num_exploration_steps`, which is set to 10k by default. You need to collect data using the `ArgmaxPolicy` policy which chooses to perform actions that maximize the exploration critic value.

In experiment log directories, you will find heatmap plots visualizing the state density in the replay buffer, as well as other helpful visuals. You will find these in the experiment log directory, as they are output during training. **Pick two of the three environments and compare RND exploration to random (epsilon-greedy) exploration.** Include all the state density plots and a comparative evaluation of the learning curves obtained via RND and random exploration in your report.

```
python cs285/scripts/run_hw5_expl.py --env_name *Chosen Env 1* --use_rnd
--unsupervised_exploration --exp_name q1_env1_rnd

python cs285/scripts/run_hw5_expl.py --env_name *Chosen Env 1*
--unsupervised_exploration --exp_name q1_env1_random

python cs285/scripts/run_hw5_expl.py --env_name *Chosen Env 2* --use_rnd
--unsupervised_exploration --exp_name q1_env2_rnd

python cs285/scripts/run_hw5_expl.py --env_name *Chosen Env 2*
--unsupervised_exploration --exp_name q1_env2_random
```

For debugging this problem, note that on the `easy` environment we would expect to obtain a return of -25 within 4000 iterations. The density of the state-action pairs on this `easy` environment should be, as expected, more uniformly spread over the reachable parts of the environment (that are not occupied by walls) with RND as compared to random exploration where most of the density would be concentrated around the starting state.

For the second sub-part of this problem, you need to implement a separate exploration strategy of your choice. This can be an existing method, but feel free to design one of your own. To provide some starting ideas, you could try out count-based exploration methods (such as pseudo counts and EX2) or prediction error based approaches (such as exploring states with high TD error) or approaches that maximize marginal state entropy. Compare and contrast the chosen scheme with respect to RND, and specify possible reasons for the trends you see in performance. The heatmaps and trajectory visualizations will likely be helpful in understanding the behavior here.

```
python cs285/scripts/run_hw5_expl.py --env_name PointmassMedium-v0
--unsupervised_exploration <add arguments for your method> --exp_name q1_alg_med

python cs285/scripts/run_hw5_expl.py --env_name PointmassHard-v0
--unsupervised_exploration <add arguments for your method> --exp_name q1_alg_hard
```

Part 2: Offline learning on exploration data. Now that we have implemented RND for collecting exploration data that is (likely) useful for performing exploitation, we 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.

For the first sub-part of this problem, you will write down the logic for disabling data collection in `agents/explore_or_exploit_agent.py` after exploitation begins and only evaluate the performance of the extrinsic critic after training on the data collected by the RND critic. To begin, run offline training at the default value of `num_exploration_steps` which is set to 10000. Compare DQN to CQL on the medium environment.

```
# cql_alpha = 0 => DQN, cql_alpha = 0.1 => CQL

python cs285/scripts/run_hw5_expl.py --env_name PointmassMedium-v0 --exp_name q2_dqn
--use_rnd --unsupervised_exploration --offline_exploitation --cql_alpha=0

python cs285/scripts/run_hw5_expl.py --env_name PointmassMedium-v0 --exp_name q2_cql
--use_rnd --unsupervised_exploration --offline_exploitation --cql_alpha=0.1
```

Examine the difference between the Q-values on state-action tuples in the dataset learned by CQL vs DQN. Does CQL give rise to Q-values that underestimate the Q-values learned via a standard DQN? If not, why? To answer this question, first you might find it illuminating to try the experiment shown below, marked as a hint and then reason about a common cause behind both of these phenomena.

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?

For the second sub-part of this problem, perform an ablation study on the performance of the offline algorithm as a function of the amount of exploration data. In particular vary the amount of exploration data for atleast two values of the variable `num_exploration_steps` in the offline setting and report a table of performance of DQN and CQL as a function of this amount. You need to do it on the medium or hard environment. Feel free to utilize the scaled and shifted rewards if they work better with CQL for you.

```
python cs285/scripts/run_hw5_expl.py --env_name *Chosen Env* --use_rnd
--num_exploration_steps=[5000, 15000] --offline_exploitation --cql_alpha=0.1
--unsupervised_exploration --exp_name q2_cql_numsteps_[num_exploration_steps]

python cs285/scripts/run_hw5_expl.py --env_name *Chosen Env* --use_rnd
--num_exploration_steps=[5000, 15000] --offline_exploitation --cql_alpha=0.0
--unsupervised_exploration --exp_name q2_dqn_numsteps_[num_exploration_steps]
```

For the third sub-part of this problem, perform a sweep over two informative values of the hyperparameter α besides the one you have already tried (denoted as `cql_alpha` in the code; some potential values shown in the run command below) to find the best value of α for CQL. Report the results for these values in your report and compare it to CQL with the previous α and DQN on the medium environment. Feel free to utilize the scaled and shifted rewards if they work better for CQL.

```
python cs285/scripts/run_hw5_expl.py --env_name PointmassMedium-v0 --use_rnd
--unsupervised_exploration --offline_exploitation --cql_alpha=[0.02, 0.5]
--exp_name q2_alpha[cql_alpha]
```

Interpret your results for each part. Why or why not do you expect one algorithm to be better than the other? Do the results align with this expectation? If not, why?

Part 3: “Supervised” exploration with mixed reward bonuses. So far we have looked at an “unsupervised” exploration procedure – where we just train the exploration critic. In this part, we will implement a different variant of RND exploration that will not utilize the exploration reward and the environment reward separately (as you did in Part 1), but will use a combination of both rewards for exploration as compared to performing fully “supervised” exploration via the RND critic and then finetune the resulting exploitation policy in the environment. To do so, you will modify the `exploration_critic` to utilize 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. Once you have implemented this mechanism, run this part using:

```
python cs285/scripts/run_hw5_expl.py --env_name PointmassMedium-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=0.0 --exp_name q3_medium_dqn
```

```
python cs285/scripts/run_hw5_expl.py --env_name PointmassMedium-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=1.0 --exp_name q3_medium_cql
```

```
python cs285/scripts/run_hw5_expl.py --env_name PointmassHard-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=0.0 --exp_name q3_hard_dqn
```

```
python cs285/scripts/run_hw5_expl.py --env_name PointmassHard-v0 --use_rnd
--num_exploration_steps=20000 --cql_alpha=1.0 --exp_name q3_hard_cql
```

Feel free to utilize the scaled and shifted rewards if they work better with CQL for you. For these experiments, compare the performance of this part to the second sub-part of Part 2 (i.e. results obtained via purely offline learning in Part 2) for a given number of `num_exploration_steps`. Include the learning curves for both DQN and CQL-based exploitation critics on these environments in your report.

Further, how do the results compare to Part 1, for the default value of `num_exploration_steps`? How effective is (supervised) exploration with a combination of both rewards as compared to purely RND based (unsupervised) exploration and why?

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.

Part 4: Bonus problem (ungraded, but counts towards extra credits and we encourage you to try it). We also provide a harder layout of the gridworld environment in the environment file located at `envs/pointmass/pointmass.py`. Try out your exploration methods on one or more of these domains, and explore how offline training and online finetuning work in these domains. Report any interesting findings including state-visitation heatmaps, and policy performance.

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 `cs285` 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
│   └── ...
├── cs285
│   ├── agents
│   │   ├── explore_or_exploit_agent.py
│   │   └── ...
│   ├── policies
│   │   └── ...
│   └── ...
├── README.md
└── ...
```

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**.