OFFLINE REINFORCEMENT LEARNING HANDS-ON

Jakub Kmec[*]
InstaDeep
j.kmec[†]

Louis Monier[*]
InstaDeep
louis.monier[‡]

Alexandre Laterre
InstaDeep
a.laterre[†]

Thomas Pierrot
InstaDeep
t.pierrot[†]

Valentin Courgeau
InstaDeep
v.courgeau[†]

Olivier Sigaud
Sorbonne Université
olivier.sigaud[§]

Karim Beguir
InstaDeep
kb[†]

ABSTRACT

Offline Reinforcement Learning (RL) aims to turn large datasets into powerful decision-making engines without any online interactions with the environment. This great promise has motivated a large amount of research that hopes to replicate the success RL has experienced in simulation settings. This work ambitions to reflect upon these efforts from a practitioner viewpoint. We start by discussing the dataset properties that we hypothesise can characterise the type of offline methods that will be the most successful. We then verify these claims through a set of experiments and designed datasets generated from environments with both discrete and continuous action spaces. We experimentally validate that diversity and high-return examples in the data are crucial to the success of offline RL and show that behavioural cloning remains a strong contender compared to its contemporaries. Overall, this work stands as a tutorial to help people build their intuition on today's offline RL methods and their applicability.

1 INTRODUCTION

Offline Reinforcement Learning holds the promise of bridging the gap between reinforcement learning algorithms and real-world applications. By taking advantage of large pre-collected datasets, it can mitigate the technical concerns associated with online data acquisition, as it is often expensive or dangerous to interact with real environments. These promises have triggered a surge of interest for research around this problematic and yielded notable improvements (Kumar et al., 2020; Wang et al., 2020; Nair et al., 2020). However, the fast publication pace ends up being confusing for non-experts, as it is not straightforward to know which offline algorithm should be used to address a particular application. Consequently, this might prevent its wide adoption, similar to what supervised learning has experienced with the availability of large datasets. Recent attempts were made to overcome this obstacle. Levine et al. (2020) provides the reader with the conceptual tools needed to get started on research on offline RL. Fu et al. (2020) and Gulcehre et al. (2020) try to facilitate the measure of progress of research on offline RL by introducing benchmarks specifically designed for the offline setting. Despite these efforts, a hands-on up-to-date comparison of recent methods still ought to be carried out. The objective of this article is to provide the reader with a ”cooking recipe” where we first focus on the ingredients, namely the offline methods and the dataset specifications. We then apply these algorithms on various datasets designed to highlight which method performs best in each setting. Finally, we discuss these findings so as to help the reader to build intuition on what might be the most appropriate method for their use case and start off the right foot.

[*]Equal contribution.
[†]@instadeep.com
[‡]@mines-paristech.fr
[§]@upmc.fr

2 Background

We consider a classic Reinforcement Learning (RL) framework with a Markov Decision Process (MDP), defined by the tuple ($\mathcal{S}$, $\mathcal{A}$, p, r, γ, ρ_0). $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively, and $\gamma \in (0, 1]$ is the discount factor. The dynamic or transition distribution is denoted as $p(s'|s, a)$, the initial state distribution as $\rho_0(s)$, and the reward function as $r(s, a)$. RL aims to maximize the expected sum of (discounted) rewards over trajectories τ by finding the optimal policy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right].$$

Distributional shift: the key difficulty in offline RL. RL algorithms, even the ones that can be trained from off-policy data, can not usually learn an optimal behaviour without additional on-policy interactions. This well-known issue, referred to as *distributional shift* (also described as extrapolation/bootstrapping error accumulation (Fujimoto et al., 2019b; Kumar et al., 2019)) occurs when the used function approximator is trained under one distribution, i.e. the offline dataset, but evaluated on a different one – the learned policy induces a different visited state-action distribution. Indeed, when entering states that are far outside the training distribution, even with strong inductive bases, we can hardly provide any convergence guarantee on the agent's behaviour. Over the last few years, a considerable amount of algorithmic solutions has been published to address this problem. So far, the different approaches can be split into three different groups: explicit policy constraint methods, Q-values regularisation approaches and implicit policy constraint methods.

Solution 1: Explicit Policy Constraint Methods. A first solution to address the distributional shift is to constrain how much the learned policy might differ from the behaviour policy having created the dataset so that the distributional shift is bounded. Practically, one can force the parametrised policy being learned π_ϕ to take actions close to the behavioural distribution π_β through a divergence measure D_m:

$$\arg\max_{\phi} \mathbb{E}_{s\sim\mathcal{D}} \left[\mathbb{E}_{a\sim\pi_\phi(.|s)} \left[Q_\theta(s, a) \right] \right] \quad s.t. \quad D_m(\pi_\phi, \pi_\beta) \leq \epsilon.$$

This type of method fundamentally favors pessimism over risky exploration (Fujimoto et al., 2019b; Kumar et al., 2019). This approach is mostly effective on expert demonstrations as it operates more like an imitation process rather than an offline algorithm (Fujimoto et al., 2019a). They also suffer from some limitations as (i) the estimated Q-values are often too conservative; (ii) the (unknown) behavioural policy π_β has to be modeled.

Two categories of solutions appeared to overcome those limitations. We chose to focus on the latest, most promising and best representative methods from each, namely Critic Regularised Regression (CRR) (Wang et al., 2020; Nair et al., 2020) and Conservative Q-Learning (CQL) (Kumar et al., 2020). Neither of the two require modelling the prior policy, nor enforce an explicit constraint to avoid out-of-distribution actions.

Solution 2: Q-values Regularisation. The CQL algorithm inserts an additional regularisation term on top of standard policy evaluation steps to learn a conservative Q-function and avoids over-estimation issues, highly detrimental when boostrapping:

$$\arg\min_{\theta} \ \alpha \mathbb{E}_{s\sim\mathcal{D}} \left[\log \sum_{a} \exp Q_\theta(s, a) - \mathbb{E}_{a\sim\pi_\beta(.|s)} \left[Q_\theta(s, a) \right] \right] + \frac{1}{2} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[(Q_\theta - \mathcal{B}^{\pi_\phi} Q)^2 \right] \tag{1}$$

Intuitively, the *LogSumExp* term squashes down the Q-values, especially the overestimated ones. The second term compensates the ”push down” effect by maximizing the Q-values of (state-action) pairs sampled from the dataset. The resulting modified objective prevents overestimation without constraining the policy to stay close the behavioural distribution. This new critic update yields strong theoretical guarantees: the expected Q-value under π is actually a lower bound of its true value. In Section 4, we provide further practical implementation remarks for CQL.

Solution 3: Implicit Policy Constraint Methods. Another approach is to implicitly apply a constraint on the policy to dissuade it from selecting out-of-distribution actions. The idea is to create a weighted behavioural cloning objective where bad actions are discarded and good ones are used to train the agent. In practice, the policy update re-weights the state-action pairs from the buffer leveraging the advantage estimations coming from the critic:

$$\arg\max_{\phi} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\log\left(\pi_\phi(a \mid s)\right) f(Q_\theta, \pi_\phi, s, a)\right],$$

where the so-called *filtered* function pushes the policy towards the most reward-promising transitions sampled from the buffer. Note that in the past years, several methods tried to implicitly induce constraints in the RL objective (Ghasemipour et al., 2020; Siegel et al., 2020; Wu et al., 2019). We focus on the latest up-and-coming method CRR which was published approximately at the same time as AWAC (Nair et al., 2020). Both algorithms implement the implicit constraint similarly. See Appendix A.2 for the definitions used in practice for the *filtered* function and the advantage.

3 Dataset characteristics

In this section, we explore the main characteristics of offline RL datasets. The experimental dataset setup is further detailed in Appendix A.1 for our experiments presented in Section 4.1 and the characteristics of each dataset are presented in Appendix B.

Online RL is effectively a feedback loop: the chosen actions determine the training data. In offline RL, the dataset and amount of exploratory behaviour are kept fixed. We are therefore limited to finding the best policy for the MDP defined by the dataset rather than solving the true MDP that exists in the real world. Therefore, having a limited dataset is detrimental to the performance of an offline agent.

Trajectories quality vs state-action coverage? This trade-off underpins offline RL analogously to the exploration vs. exploitation dilemma in online RL. On one hand, the dataset must include *high-quality* actions i.e. leading to high-reward episodes. On the other hand, learning a policy that outperforms the behavioural policy requires the dataset to include ”*bad*” (exploratory) actions that the behavioural policy would not have taken. That is, a dataset with a higher state-action *coverage*. See the first experiment in Section 4.1 as well as Figures 7–9, App. B and the interpretation therein.

At one extreme lies a **fully-random** behavioural policy which collects a dataset with very good action space coverage but only for the states that are reachable by such policy. Therefore, in environments requiring elaborate exploration strategies, the dataset might not contain enough high-reward transitions to learn a satisfactory policy. Furthermore, datasets collected by random policies are not very common in practice.

At the other end is a **fully-deterministic** behavioural policy, e.g. logs generated by a deterministic heuristic. It is very difficult to learn policies from offline data that outpreform such heuristics: each state is only paired with a single action leading to a poor action-state coverage. Current approaches rely on the generalization, or extrapolation capability of the function approximation. This provides few guarantees and usually leads to overestimation errors that accumulate (*distributional shift*). The field of Causal Inference provides a more extensive treatment of such scenarios with more underlying assumptions that provide additional guarantees. It introduces the concept of *counterfactual queries*, which attempt to estimate the value—or so-called *effect*—of unobserved actions (Pearl, 2009).

To summarise, neither a fully random nor a fully deterministic behavioural policy are suitable for collecting offline RL datasets. There are two key properties the behavioural policy should have to obtain an actionable dataset: 1) *Sufficient quality* to reach all high-reward regions (Fig. 1). 2) *Sufficient coverage* to allow for fruitful exploration (Section 4.1 on outperforming the dataset).

This being said, quantifying the level of exploration in an offline dataset is a difficult problem in itself. Proxies can be used to better understand what type of behaviour is available within a dataset. Some examples include: (i) reward distribution, (ii) actions distribution (e.g. policy entropy), (iii) episode length distribution, (iv) maximum reward in dataset / maximum attainable reward, (v) state coverage, (vi) state-action coverage. We plot the first five metrics for each of the datasets used in our experiments in Appendix B. It is important to note that visualizing the last two metrics is not always trivial in high dimensional environments.

4 Experiments and Results

The goal of this section is to help the reader better understand how current state-of-the-art methods (CQL and CRR) perform as a function of the dataset quality and coverage ; we also highlight some common failure modes. We first show how the chosen offline algorithms perform in a simple interpretable grid world environment with discrete actions. We then run the same experiments on more realistic tasks which include higher dimensional environments with continuous action spaces. We also compare CQL and CRR approaches with naive off-policy baseline algorithms (DQN/SAC) and Behaviour Cloning (BC). Baseline implementation details can be found in Appendix A.2.

4.1 Exploring dataset edge cases

In this section, we run three experiments on a simple MiniGrid environment (Chevalier-Boisvert et al., 2018) to help the reader develop a better understanding of the core issues specific to offline RL. The goal is to strip away any confounding factor such as the need perception in pixel observations or a stochasticity in the environment that could make the results more convoluted.

How much does performance vary with dataset quality? As mentioned in the previous section, dataset quality can be described in many ways. Here, we use an ϵ-greedy expert with 3 different values of ϵ as described in Appendix A.1. As ϵ gets larger, the action selection becomes more stochastic and deviates from the expert baseline.

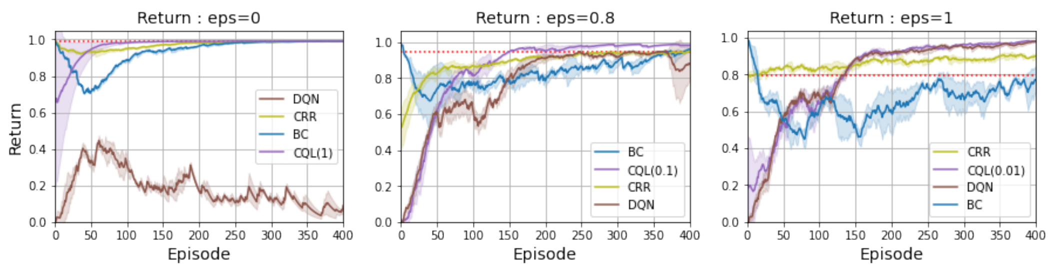


Figure 1: Comparison of CQL, CRR (with exponential filter), DQN and BC on datasets of different quality. The red dotted line shows the average episode return for each dataset (0.991, 0.947, 0.796 respectively).

We carried out experiments across datasets of varying quality and a grid search over the key hyper-parameters for each algorithm. All the results can be found in Appendix C. Figure 1 depicts the results obtained when training each agent on an expert, medium-quality and fully random dataset, respectively. CQL consistently achieves the highest returns. However, this comes at the cost of a high sensitivity to the value of alpha which had to be tuned precisely for each dataset. CQL with small alpha ($\alpha = 0.01$) performs better on datasets that contain more randomness, but fails on expert data (see Fig. 12). The opposite is true for large values of alpha, i.e. for $\alpha = 1$. This is expected, since alpha determines how much weight is put on the regularising term (Eq. 1) which implicitly constrains the policy to lie close to the dataset distribution. CRR_{exp} (CRR with an exponential filter) achieves lower returns but is more robust to the choice of hyper-parameters, such that the same value performs well across all experiments. Finally, the baselines also behave as expected, the performance of BC is upper-bounded by the average return of the trajectories in each dataset: only preforms well on the expert dataset. DQN, on the other hand, fails in the expert settings due to distributional shift (discussed in sec. 2), but performs well on a fully random dataset thanks to its high state-action space coverage.

How much can we improve upon the dataset performance? To test this, we construct a highly stochastic dataset with very sparse rewards. It is essential to employ RL methods in such settings, since other approaches, such as behavioural cloning, are unable to exploit the sequential nature of the problem. We collect a dataset by a random agent acting in the Lava environment shown in Figure 4.

A detailed visualization of the dataset characteristics can be found in Figure 10, Appendix B. Note that it only includes 3.1% of positive trajectories with an average episode return of 0.024. Figure 2 on the right shows the performance of our selected methods on this highly sparse dataset. We can see that CRR$_{\text{exp}}$ exceeds the maximum episode return in the dataset (blue dotted line), which means it must piece together parts of different trajectories. This behaviour has been previously referred to as stitching (Fu et al., 2020). CQL, with the right alpha still greatly improves upon the average dataset behaviour, but does not exceed the maximum. BC and DQN fail in this settings which is why we didn't include them.

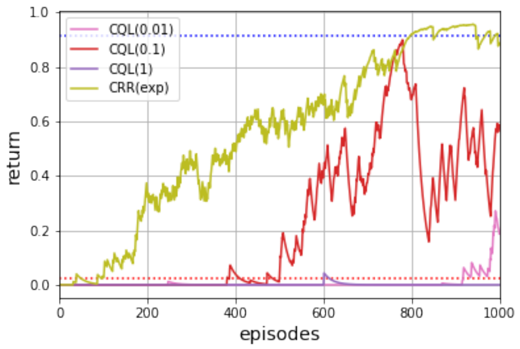


Figure 2: Red and blue dotted lines show the average and maximum episode return respectively.

Can existing methods extract optimal policies from multi-modal datasets? The experiment described in Appendix A.1 aims to test the agents ability to recover the optimal behaviour from a dataset containing trajectories of policies of various qualities. The most straightforward way to tackle out-of-distribution actions is to constrain the policy to lie close to the behaviour policy. Methods that use such approach e.g. CQL, are expected to fail in this setting. Indeed, we can see in Table 1 that CQL with a high value of alpha converges to the incorrect sub-optimal policy. On the other hand, if we decrease alpha too much, it fails to learn at all and suffers from the distributional shift. There is a value of alpha for which CQL converges to the correct policy, but the algorithm is highly sensitive and was difficult to tune correctly. BC and CRR with an exponential filter fail to learn the correct policy. We tested a variant of CRR by incorporating a conservative critic that uses the CQL penalty (CCRR). The conservative critic made the advantage estimates more accurate and successfully learned the optimal policy, but introduced the additional complexity of tuning the value of alpha.

Method	Hyperparameters	Ep. length
CQL	$\alpha = 0.001$	500
	$\alpha = 0.01$	13
	$\alpha = 0.1, \alpha = 1$	17
CRR$_{\text{exp}}$	$\beta = 1$	17
	$\beta = 0.01$	500
CCRR	$\alpha = 0.01, \beta = 1$	17
	$\alpha = 0.01, \beta = 0.01$	13
BC		17

Table 1: Final episode length on the multi-modal dataset described in Appendix A.1.

From these experiments, practitioners might want to keep in mind that the quality of the offline dataset is crucial to train efficient decision-making agents, even more than in the supervised setting. In scenarios where the dataset contains highly stochastic behaviours, it is worth trying an off-policy online method such as DQN, which directly optimizes the true objective without imposing any constraints. In most practical scenarios however, the dataset will have gaps in its state-action coverage and online methods will fail due to distributional shift. This is where CQL and CRR prove useful. Properly tuned CQL usually yields the best results, but CRR is more robust to hyper-parameter selection and works better on datasets with sparse rewards. Lastly, if we have a dataset that was collected by a highly deterministic policy, BC is going to be very difficult to outperform. Understanding how these methods perform on environments with increasing complexity is what we deal with in the next section.

4.2 Making the most out of offline RL for continuous control

We present here experiments on challenging continuous control tasks for people looking to leverage their dataset in more realistic settings. Similarly to the previous experiments with discrete action spaces, we created three generic datasets to cover a wide range of real-world scenarios while keeping it as simple as possible: high quality data (expert policy behaviour), medium quality (average performance) and purely random data. We intentionally selected environments with an increasing complexity to evaluate the algorithms in various contexts: a simple 2D grid (`PointMaze-v0`) and two classic MuJoCo benchmark tasks (`HalfCheetah-v2` and `Humanoid-v2`). Additional details on the datasets creation protocol and the environments can be found in Appendix A.1 and A.5 respectively. The experiments below depict how the offline methods actually perform when they are implemented following the instructions from the papers, as well as when using the suggested hyper-parameter values.

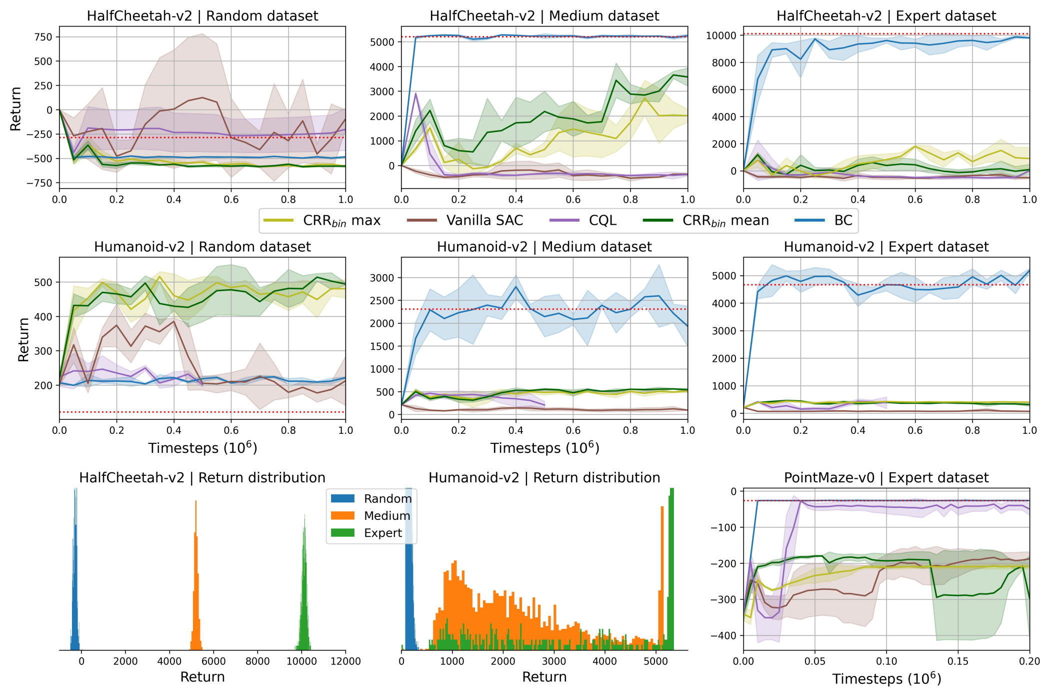


Figure 3: Comparison of CQL, CRR, SAC and BC on different environments and datasets of various qualities. The red dotted line indicates the average episode return and the normalized return distributions are plotted in the bottom left for each dataset.

What is the best offline training strategy when dealing with continuous control tasks? The experiments depicted in Figure 3 first show that naive application of an off-policy algorithm like SAC fails when applied in the offline setting. As for BC, it reaches a near-optimal strategy with medium and high-quality data but automatically fails when a large part of random or low-performing behaviours is present in the dataset.

As shown in Figure 3 top-center, CRR is the only offline method able to partly retrieve the performance contained in the medium quality dataset on `HalfCheetah-v2` but does not improve upon it. By definition, CRR is only able to mimic a subset of good action decisions. Indeed, the agent is implicitly constrained to lie in the vicinity of the best performing trajectory in the static dataset through the use of the *filtered* function. Therefore, it cannot extract information across trajectories and therefore cannot outperform the dataset performance. Note that it fails when the action distribution is narrow or when its entropy is high (no useful information to extract). In practice, the exponential *filtered* function can cause numerical instabilities leading to crashes during the training. Therefore, we mainly use the binary function (CRR$_{bin}$) with both *mean* and *max* advantage estimates.

As far as CQL is concerned, it remains unstable and is unable to learn robustly across all the dataset-task pairs. Moreover, we found it difficult to implement: the authors recommend using the *LogSumExp* function provided by Torch or TensorFlow libraries for the discrete actions setting and suggest otherwise for the continuous case, while the official repository seems to use it in either case. They also use an additional term in their code for the Q-value regulariser: $\sum_{a_{t+1}^i \sim \pi(\mathbf{a}_{t+1}|s_{t+1})}^{N} \exp\left\{Q(s_{t+1}, a_{t+1}^i) - \log \pi(a_{t+1}^i \mid s_{t+1})\right\}$, which is not mentioned in the paper. Intuitively, they likely include these additional samples (from the region of interest where we might find higher Q-values) to improve the accuracy of the regularisation term[*]. Finally, they also use a temperature hyper-parameter to make the soft-maximum estimate more accurate and remove the non-deterministic part of the Q-value target (additional entropy term) which are not mentioned in the paper. Overall, applying the CQL method is appealing but even after fixing all these discrepancies, we were not able to replicate the results. We believe the fragile hyper-parameters equilibrium makes practical implementations difficult and subject to high variance. Finally, in a simpler continuous environment like `PointMaze-v0`, CQL and BC interestingly perform best and retrieve the expert behaviour whereas CRR gets stuck in a local minimum.

Overall, offline RL is undeniably a tough problem. The takeaway is that spending a lot of resources on data collection and/or algorithm customization does not guarantee a robust strong performance for complex continuous control tasks as the latest methods are unable to generalize well on most new dataset-task pairs. A safe bet is BC which remains a strong and robust baseline for any continuous settings as long as the dataset contains qualitative data. We provide further fine-tuning experiments in Appendix C to evaluate whether it is possible to get further gains of performance with a limited additional exploration after offline training when a simulator can be used.

5 Conclusion and Discussion

In this paper, after presenting three methods from recent offline RL contributions, we take a step back from the literature to return to practical considerations: data and algorithms. For the former, we discuss various dataset properties that may play a role in a successful deployment of offline RL. Both the quality of the trajectories therein contained and their coverage of the state-action space offer a key trade-off when it comes to data. We devise a collection of experiments to document their interface through the lens of three algorithms (BC, CRR and CQL) and three dataset quality levels (random, medium and expert). Since the problems we tackle are undocumented, we provide foundations to help the reader better grasp the issues and suitability of different offline RL methods. We hypothesise that datasets that are of interest in offline RL are generated by medium quality policies, ideally with high stochasticity which lead to a dataset in which we can expect to be able to outperform the data generating process. Overall, BC strengthens its position as baseline with a robust performance across datasets with qualitative data by matching the near-optimal strategy contained in the buffer. On discrete action spaces, CQL exhibits good recovery properties for medium quality data and CRR occasionally outperforms the dataset baseline. Furthermore, when testing the methods on more complex environments, we found it even harder to beat the baselines. Even though CQL was shown to preform well on very complex environments, it can be difficult and expensive to tune because of the fragile hyper-parameter equilibrium.

The selection of the offline RL method to use and its success cannot be made without the aforementioned considerations, that is, no method performs uniformly better independently of the use case. However, dealing with custom datasets uncovers many practical pitfalls that have not been addressed in the literature such as algorithms' fitness for different levels of data quality. Given the pace at which the field of offline RL is evolving, we expect it to become a central question in the near future.

[*]After contacting the authors, they confirmed that all the available sample sources are needed to evaluate the regulariser more accurately. They also suggest a higher number of samples ($N \approx$ 50–100) instead of the paper-recommended value ($N \approx$ 10).

References

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Geoffrey Cideron, Thomas Pierrot, N. Perrin, Karim Beguir, and Olivier Sigaud. Qd-rl: Efficient mixing of quality and diversity in reinforcement learning. *ArXiv*, abs/2006.08505, 2020.

Justin Fu, Aviral Kumar, Ofir Nachum, G. Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *ArXiv*, abs/2004.07219, 2020.

Scott Fujimoto, H. V. Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *ArXiv*, abs/1910.01708, 2019a.

Scott Fujimoto, D. Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019b.

Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. *ArXiv*, abs/2007.11091, 2020.

Caglar Gulcehre, Ziyu Wang, A. Novikov, T. L. Paine, Sergio Gomez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel J. Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, J. Li, Mohammad Norouzi, Matt Hoffman, Ofir Nachum, G. Tucker, Nicolas Heess, and N. D. Freitas. Rl unplugged: Benchmarks for offline reinforcement learning. *ArXiv*, abs/2006.13888, 2020.

T. Haarnoja, Aurick Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

A. Kumar, Justin Fu, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *ArXiv*, abs/1906.00949, 2019.

Aviral Kumar, Aurick Zhou, G. Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *ArXiv*, abs/2006.04779, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *ArXiv*, abs/2006.09359, 2020.

Judea Pearl. *Causality*. Cambridge university press, 2009.

Noah Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, T. Lampe, Roland Hafner, and Martin A. Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *ArXiv*, abs/2002.08396, 2020.

Ziyu Wang, A. Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, B. Shahriari, N. Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, and N. D. Freitas. Critic regularized regression. *ArXiv*, abs/2006.15134, 2020.

Y. Wu, G. Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *ArXiv*, abs/1911.11361, 2019.

A Specs and protocols

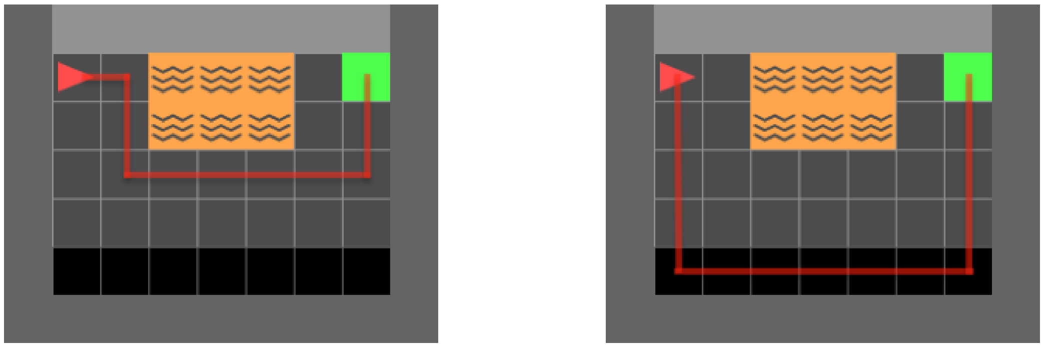
A.1 Dataset Creation Protocol

Discrete action spaces We used a simple double-DQN to train an expert agent on each environment. The fully trained expert is then used to collect datasets for each experiment.

EXPERIMENT 1: we use an ϵ-greedy expert policy with increasing ϵ values to interact with a simple empty 6x6 MiniGrid environment (Chevalier-Boisvert et al., 2018). The state in this environemnt is defined by cell position and orientation and there are 3 actions: turn left/right and move forward. The dataset used in the experiment are also visualized in Appendix B.

EXPERIMENT 2: we use a fully random agent to collect data in the Lava environment shown in Figure 4. Most random walks in this environment end up in Lava which terminates the episode with zero reward. This results in a dataset which is very sparse.

EXPERIMENT 3: the multi-modal dataset used in the third experiment contains data collected by agents with two different policies.



(a) Expert trained with greedy Q-learning updates to learn the shortest path to the goal (13 steps).

(b) Expert trained with SARSA updates with high epsilon to converge to a safe path (17 steps).

Figure 4: The two expert trajectories used for data collection.

To collect the final dataset, we generate 20% of the trajectories using expert A and the remaining 80% of the trajectories using expert B. Both experts are epsilon-greedy with $\epsilon = 0.1$. As usual, the final dataset can be visualized in Appendix B, see Figure 11. The goal is to create a dataset that contains a majority of relatively high reward, but sub-optimal trajectories, with a smaller proportion of optimal trajectories.

Continuous action spaces We used QD-RL (Cideron et al., 2020), TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018) to generate homemade offline datasets on `PointMaze-v0`, `HalfCheetah-v2` and `Humanoid-v2` benchmarks respectively. Each dataset contains 10^6 transitions. We chose three generic settings in which most of real-world situations can be classified:

- ”Random”:
  To collect the dataset, we simply let a random agent wandering around in the environment. Allegedly, random datasets show the largest state-action coverage even though it is not enough to ensure deep exploration. In a realistic setting, it corresponds to situations where the logged data is meaningless but still represents a possible physical state of the system under consideration. For example, let’s say we want to train a decision making agent to optimize a power plant operational efficiency. Arbitrary data does not feature any relevant operation strategy, yet it outlines possible values.
- ”Medium”:
  The RL agent is trained online until an average return of approximately 5000 is reached on `HalfCheetah-v2` and 2500 on `Humanoid-v2`. Then, the transitions are saved. This setting

is the most realistic as it represents a wide range of possible real-word scenarios where datasets with some sub-optimal strategies can be easily collected (hand-engineered policies or human demonstrations for instance). Going back to the power plant example, let's consider that human operators handle the energy production. Logs of the relevant plant parameters are collected and may reflect poor production performance every now and then (non-expert policy).

In general, medium quality datasets are also interesting to investigate because this setting makes it ultimately easier to detect potential improvements of an offline agent upon the best performing trajectory of the static data.

- ”Expert”:
  Same data acquisition protocol as ”medium” but with a return around 10000 on `HalfCheetah-v2`, 5000 on `Humanoid-v2` and -26 on `PointMaze-v0`. This setting highlights near-expert behaviours. A successful offline RL agent is expected to reach at least the same level of performance as contained in the buffer. But mostly it should be able to generalize to new situations and perform similarly.

The histograms of the return distributions are represented at the bottom left of Figure 3. The return profiles for `HalfCheetah-v2` show sharp and narrow distributions. For `Humanoid-v2`, the distribution contains multi-modal transitions because of the larger return coverage in the medium and expert settings.

A.2 Baselines Implementation Details

- **Naive off-policy algorithms in the offline setting:**
  As a reference baseline, we naively apply off-the-shelf off-policy RL algorithm in the offline setting (DQN (Mnih et al., 2013) and SAC (Haarnoja et al., 2018) for discrete and continuous control tasks respectively) to show its failure for every dataset-algorithm-task configurations caused by the distributional shift problem.
- **Behavioural Cloning (BC):**
  This is the simplest form of imitation learning as it does not require environment interactions. In practice, it is the easiest method to implement as it just copies/mimics actions from the batch, so no RL process is involved. BC only controls in-distribution errors and does not address the out-of-distribution actions problem:

  $$\arg\min_{\phi} \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\mathbb{E}_{a'\sim\pi_\phi(.|s)}\left[D_m\left(a',a\right)\right]\right]$$

  where D_m is a divergence measure. We use the cross-entropy and the mean square error loss for discrete and continuous control domains respectively.
- **Critic Regularised Regression (CRR):**
  The CRR algorithm acts more like a smart BC method as it cannot pull information across the best performing trajectory in the buffer. The *filtered* function which exploits the best actions and discards the bad ones has several definitions in practice:

  $$f(Q_\theta,\pi_\phi,s,a)=\mathbb{1}\left[A^{\pi_\phi}(s,a)>0\right] \text{ or } \exp\left(A^{\pi_\phi}(s,a)/\lambda\right).$$

  Moreover, CRR mainly uses two definitions for the advantage A^π.
  - The *mean* advantage: $\hat{A}_{\text{mean}}(s_t,a_t)=Q_\theta(s_t,a_t)-m^{-1}\sum_{j=1}^{m}Q_\theta(s_t,a_j)$.
  - the *max* advantage: $\hat{A}_{\text{max}}(s_t,a_t)=Q_\theta(s_t,a_t)-\max_{j=1}^{m}Q_\theta(s_t,a_j)$

  where $a_j\sim\pi_\phi(\cdot\mid s_t)$. We perform experiments with both in Section 4. Additionally, we provide a batch size sensitivity analysis for CRR in Figure 15 : the key message is that training is faster with a larger batch size. This results does not modify our conclusions Section 4.2.
- **Conservative Q-Learning (CQL):**
  We limited at 500k timesteps the runs on the `Humanoid-v2` environment as it is sufficient to clearly conclude its failing. We used the recommended MuJoCo hyper-parameters from the paper. The dual gradient ascent trick to tune the weight of the Q-value regulariser α in the policy evaluation step didn't bring about better results. We believe the robustness of CQL is largely affected by this hyper-parameter and may therefore be the reason why CQL fails in our high dimensional continuous tasks.

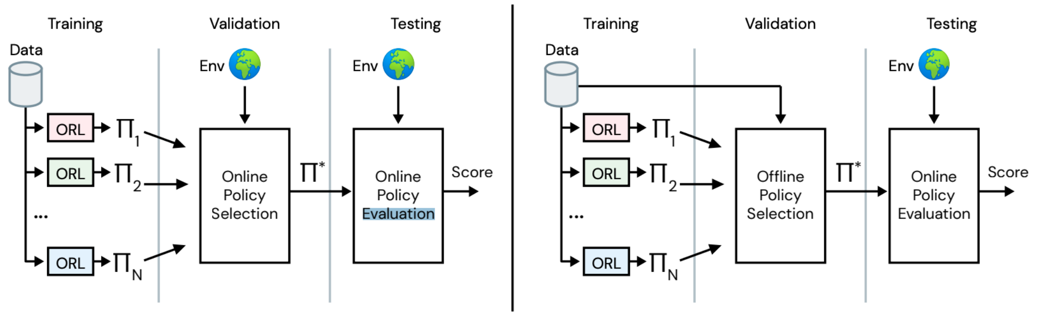


Figure 5: Comparison of evaluation protocols taken from (Gulcehre et al., 2020). (left) online policy selection - gives overly optimistic results as it allows perfect policy selection. (right) offline policy selection - closest to a real world use case where it is too expensive to query the environment.

The different methods were implemented using TensorFlow and Pytorch in the discrete and continuous setting respectively.

A.3 Evaluation Protocol

Within the scope of this work, we make a simplifying assumption that we have access to the environment for evaluation. It is important to note that this is usually not the case in practice, since the very motivation for doing offline reinforcement learning is that interacting with the environment is expensive or unsafe. The key difference is highlighted in the Figure 5, which shows the two most common evaluation protocols in offline reinforcement learning literature.

Due to the additional complexity that Offline Policy Evaluation (OPE) methods entail, we decided to use an online policy evaluation protocol within this work. More specifically, the performance is measured as the average trajectory return across the workers every 5000 steps. Our figures show the mean and the variance over 3 random seeds for each run.

A.4 Hyper-parameter Choices

Hyper-parameter	Value
RL batch size	256
Discount factor	0.99
Reward scaling	1
Replay buffer size	10^6
Actor hidden layers	[256, 256]
Actor hidden activation	ReLu
Actor learning rate	$3 \cdot 10^{-4}$
Critic hidden layers	[256, 256]
Critic hidden activation	ReLu
Critic learning rate	$3 \cdot 10^{-4}$
Soft target τ	$5 \cdot 10^{-2}$
Offline evaluation frequency	$5 \cdot 10^3$

Table 2: Default SAC hyper-parameters used for continuous control tasks.

CRR mostly relies on default SAC hyper-parameters shown in table 2. However, CQL introduces and modifies heavily the hyper-parameters, notably the policy learning rate ($3 \cdot 10^{-5}$) and the weight coefficient α of the Q-value regulariser.

A.5 Environment-Specific Details

Environments	**Observation space dimension**	**Number of actions**
MiniGrid-Empty-Random-6x6-v0	108	3
MiniGrid-DistShift1-v0	189	3

Table 3: Environments with discrete observation and action spaces

Environments	**Observation space dimension**	**Action space dimension**
PointMaze-v0	2	2
HalfCheetah-v2	17	6
Humanoid-v2	376	17

Table 4: Environments with continuous action spaces.

Specific Details about the PointMaze Environment

PointMaze is a simple 2D maze where the agent controls a 2D material point initialized randomly in $[-0.1, 0.1] \times [-0.1, -0.7]$. The goal is to exit from the three corridors maze as fast as possible. The states correspond to the agent's position $(x_t, y_t) \in [-1, 1]$ at time t. The two continuous actions are mini-step increments: (δx, δy) $\in$ [-0.1, 0.1]. The episode ends once the agent reaches the exit square or the If the length of an episode exceeds 200 time steps (unsuccessful episode). At last, the reward is computed as: $r_t = -(x_t - x_{goal})^2 - (y_t - y_{goal})^2$.

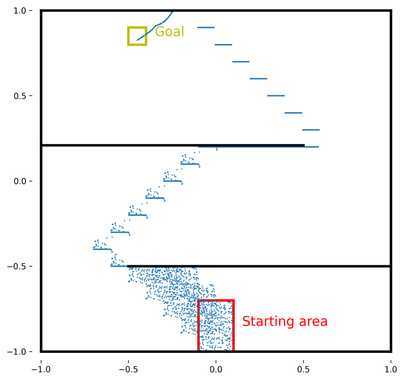


Figure 6: `PointMaze-v0` environment and 10^4 random states samples from the expert dataset

B Dataset Visualisation

In this section, we present the dataset characteristics for the discrete action spaces experiments from Section 4.1. Each 3×3 figure consists of heatmaps and histograms organised as follows: the heatmaps at the center of each side are state counts conditional on the agent facing the direction given by the position of the plot in the figure. The middle heatmap is a sum of all the directions. The total count of transitions in the dataset is in the title of each *combined* heatmap (center). Histograms are placed in the corners: distribution of rewards (Upper LHS), distribution of *positive* rewards (Upper RHS), episode length distribution (Lower LHS) and the proportion of actions within the dataset (Lower RHS).

The state-action coverage conditional on each facing direction can be assessed using those heatmaps: green/browning squares are for highly visited states whilst the lavender colour is for states sparsely visited.

By comparing the heatmaps and action count histogram of Figures 7, 8 and 9, we notice that the state coverage gets higher across all facing directions and the action counts is closer a uniform distribution, the mean reward goes down from 0.991 to 0.796. Also, the episode length increases with epsilon: we can see that while the expert dataset only contains 6k transitions, the fully random dataset has more than 113k for 1,000 episodes.

B.1 EXPERIMENT 1

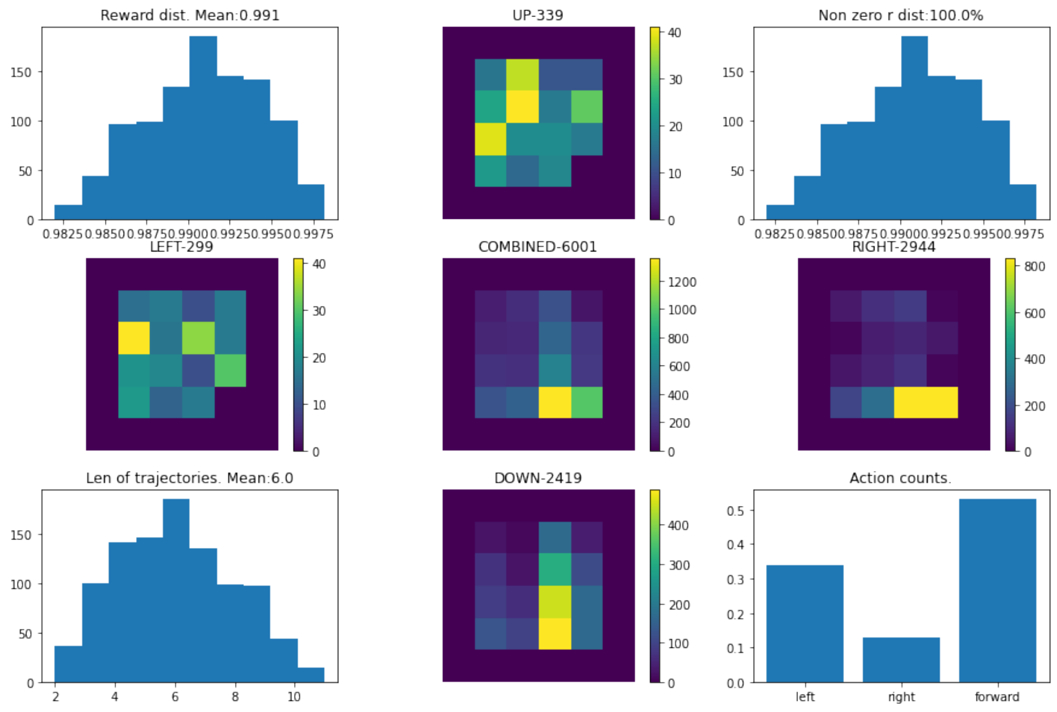


Figure 7: Expert dataset with with 1000 episodes and $\epsilon = 0$.

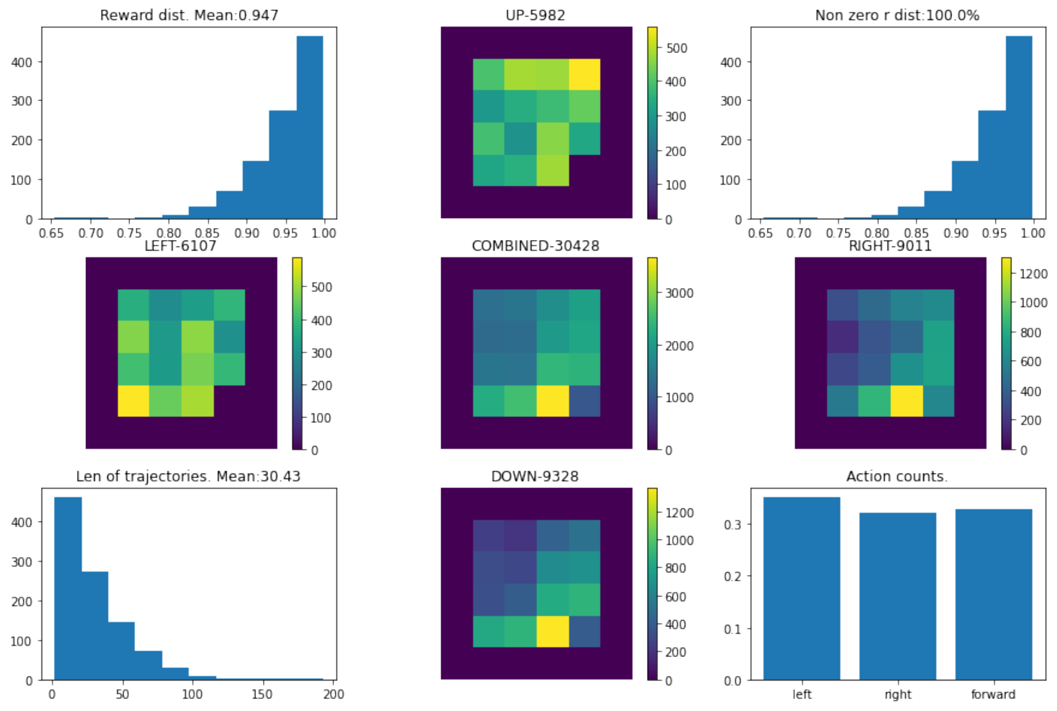


Figure 8: Expert dataset with with 1000 episodes and $\epsilon = 0.8$.

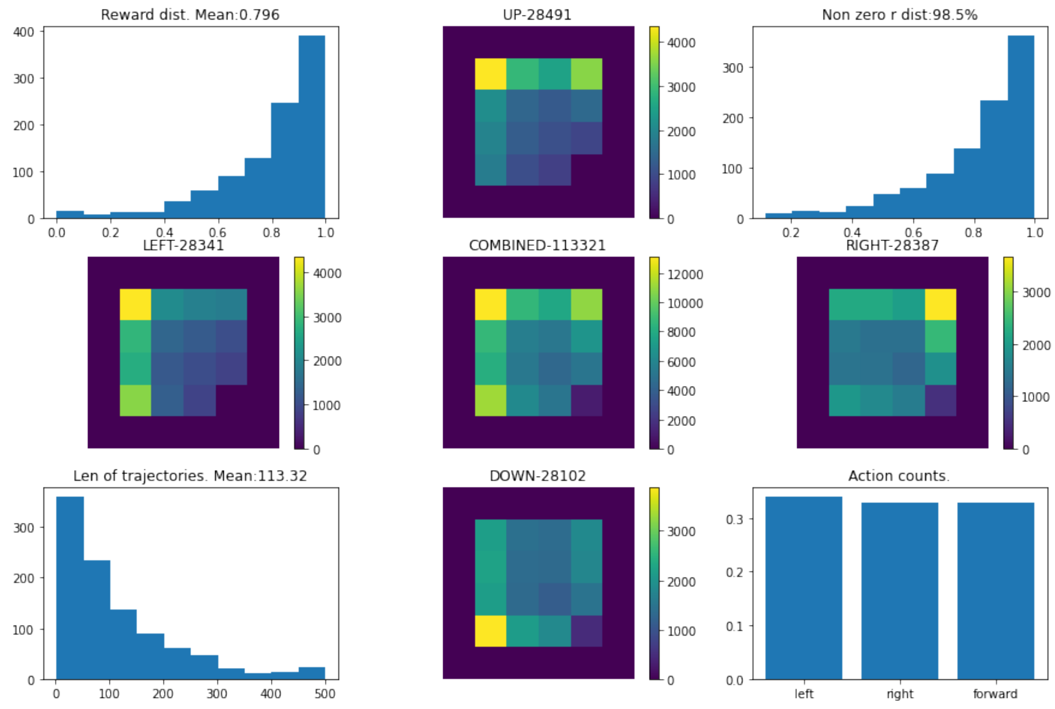


Figure 9: Expert dataset with with 1000 episodes and $\epsilon = 1$.

B.2 EXPERIMENT 2

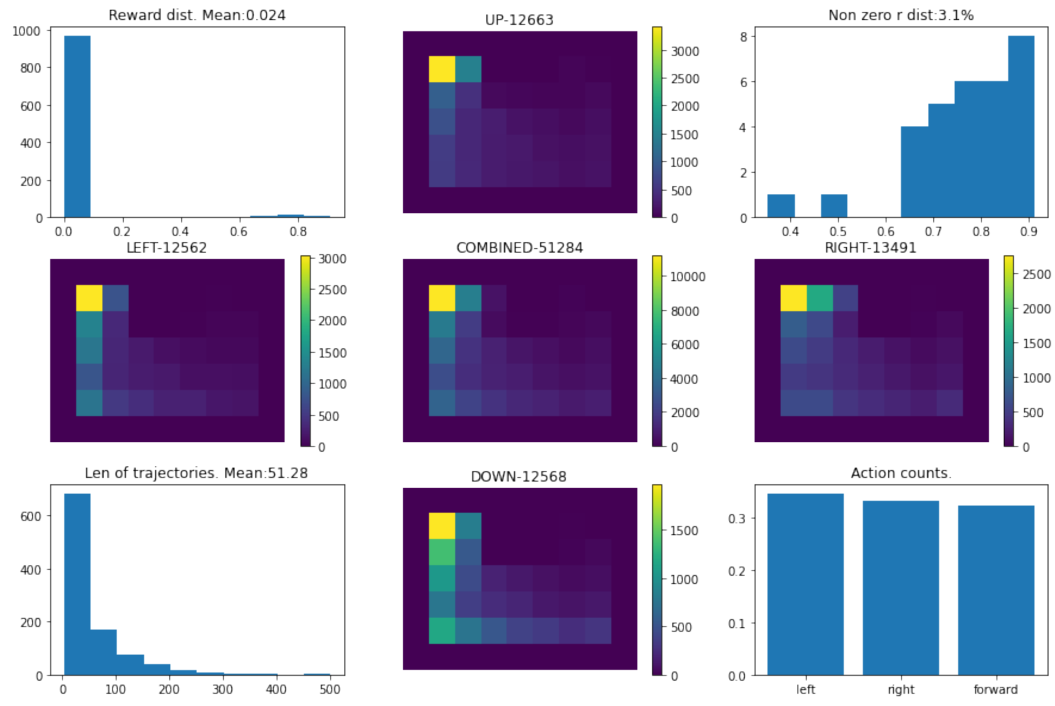


Figure 10: Dataset collected by a random agent acting in the Lava gird world.

B.3 EXPERIMENT 3

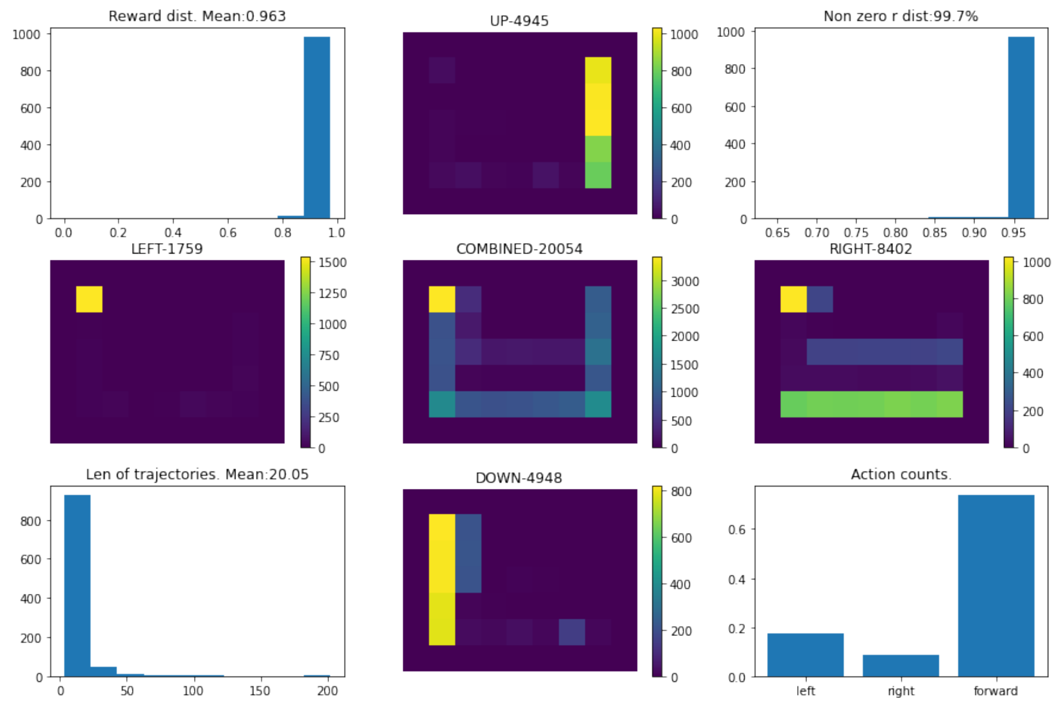


Figure 11: Multi-modal dataset with with 1000 episodes as described in A.1

C Supplemental results

C.1 Hyperparameter search

Experiments below are ran on datasets collected by an ϵ-greedy expert in the Empty-Random Mini-Grid world as described in Appendix A.1.

Epsilon	avg. reward	avg. ep. length
0	0.991	6
0.3	0.986	8.54
0.6	0.974	15.52
0.8	0.947	30.43
0.9	0.908	51.86
1	0.796	113.32

Table 5: Summary of the key characteristics of each dataset.

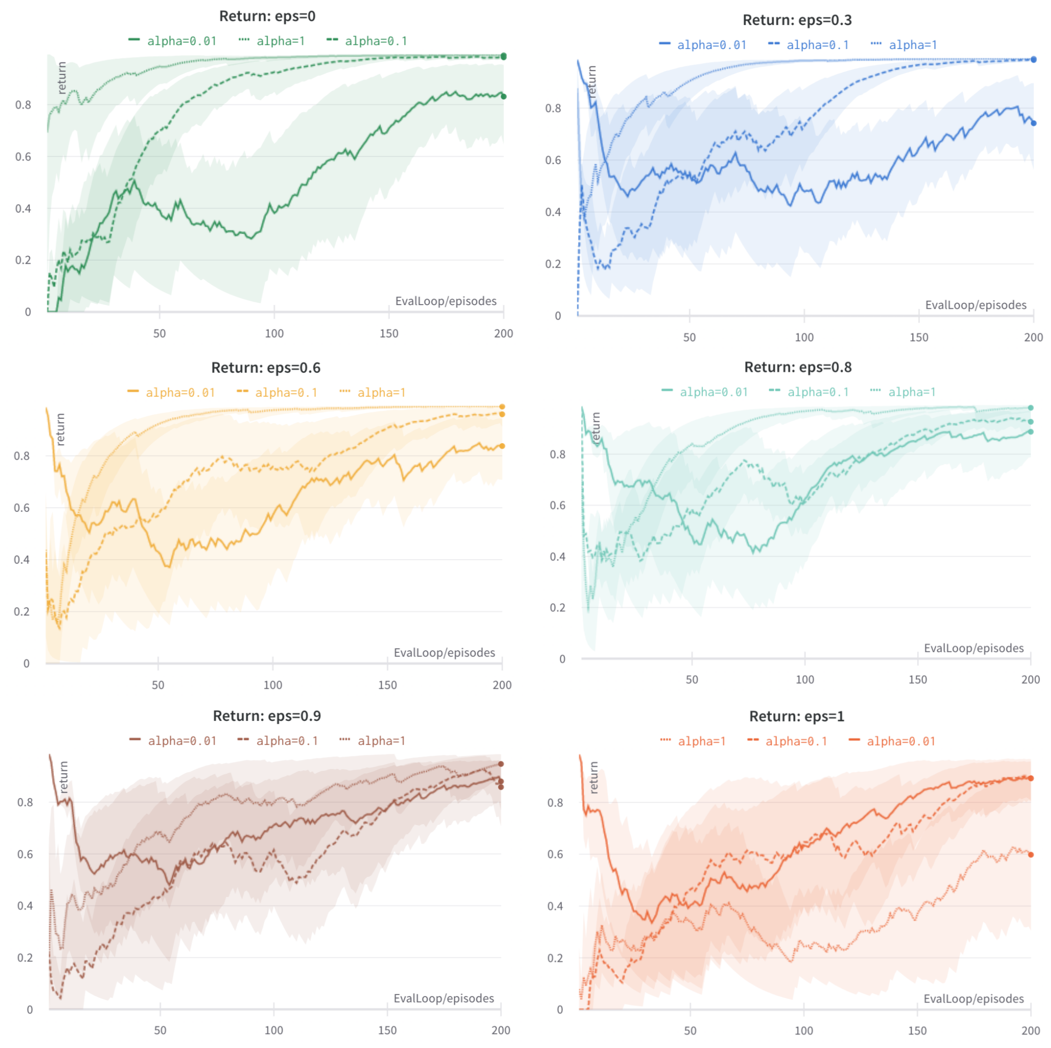


Figure 12: Performance of CQL as a function of the dataset quality.

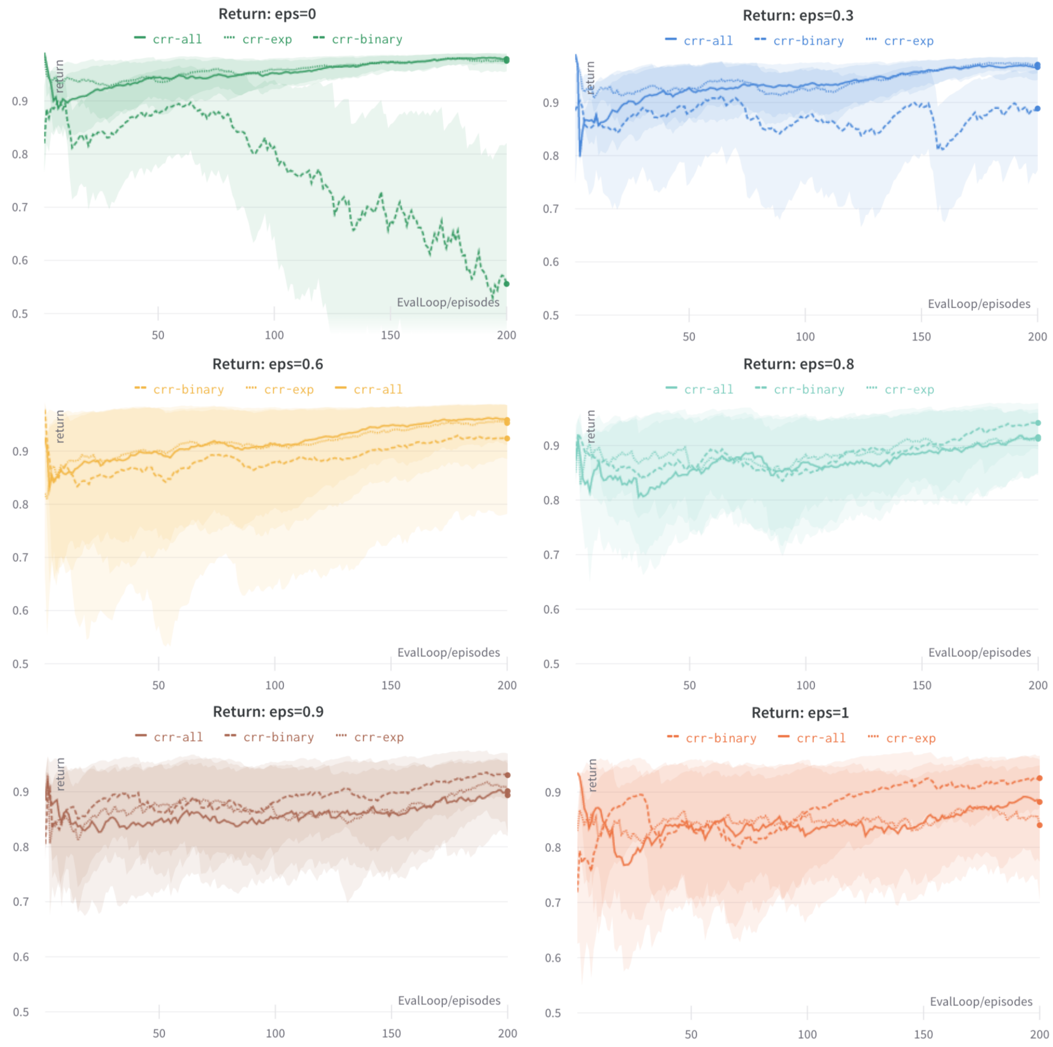


Figure 13: Performance of CRR as a function of the dataset quality.

D Online fine-tuning study

I have access to a simulator, is it possible to get further gains of performance with fine-tuning? Sometimes in real-world scenarios, environment simulators can be accessible but they can be computationally costly and/or time-consuming. Nevertheless, after purely offline training, the policy can be deployed in a simulated environment to perform a limited extra exploration which we refer as *fine-tuning*. This technique is investigated as a way to boost the agent performance with a small number of online trial-and-error interactions (Nair et al., 2020). We first train offline an agent for 10^6 timesteps then fine-tune the policy for 10^5 online interactions in the simulator (see Figure 14).

In the medium quality setting, BC suffers from performance collapse before slowly going back up again. CRR$_{bin}$ performance first reaches the average return contained in the dataset. Then, the performance crumbles entirely. Combining offline RL methods like CRR (CQL is not tested here as the offline training was not successful) with fine-tuning seems to require more care and tuning and doesn't seem to be applicable out-of-the-box.

Further investigation on CRR: batch size analysis. We show in Figure 15 that batch size hyper-parameter can play an important role for CRR, especially in the medium dataset setting. Sampling more examples simply gives a better estimate of the *filtered* function and therefore the policy is

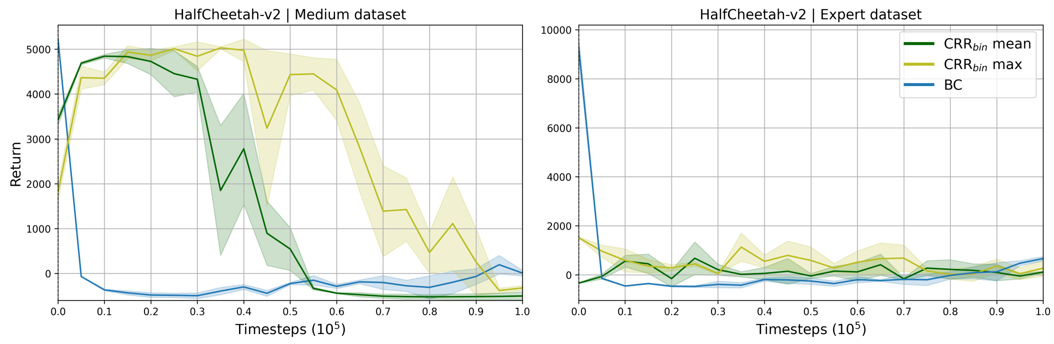


Figure 14: Fine-tuning for 10^5 timesteps after 10^6 fully offline training steps on `HalfCheetah-v2`.

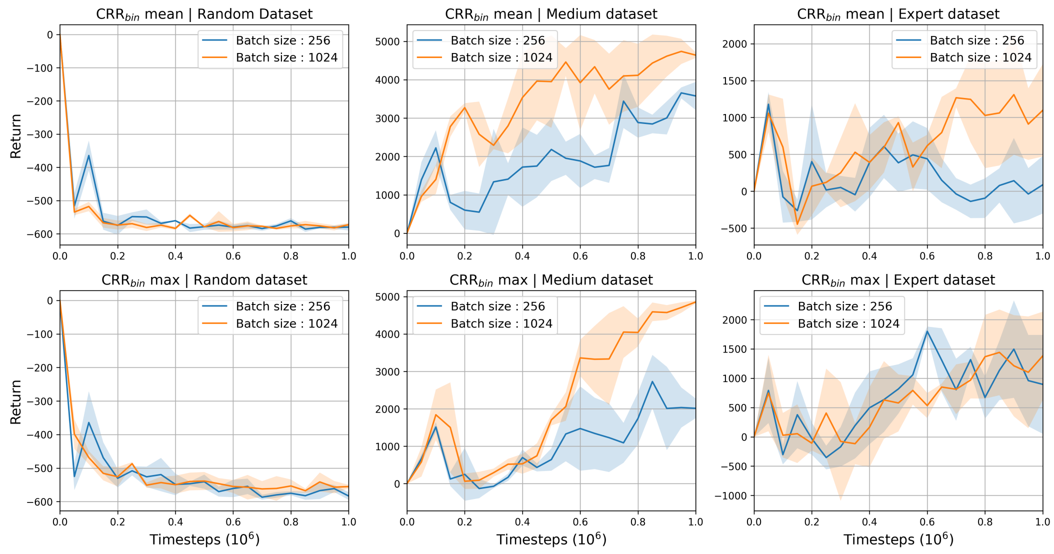


Figure 15: Batch size sensitivity analysis for CRR on `HalfCheetah-v2`.

more efficiently pushed towards the best actions. However, a larger batch size does not affect the conclusions in Section 4.2.