DENSITY ESTIMATION FOR CONSERVATIVE Q-LEARNING

Paul Daoudi, Ludovic Dos Santos, Merwan Barlier & Aladin Virmaux
Netmind - Huawei Noah's Ark Lab
Paris, France
{paul.daoudi1,ludovic.dos.santos,merwan.barlier,aladin.virmaux}@huawei.com

ABSTRACT

Batch Reinforcement Learning algorithms aim at learning the best policy from a batch of data without interacting with the environment. Within this setting, one difficulty is to correctly assess the value of state-action pairs far from the data set. Indeed, the lack of information may provoke an overestimation of the value function, leading to non-desirable behaviours. A compromise between enhancing the performance of the behaviour policy and staying close to it must be found. To alleviate this issue, most existing approaches introduce a regularization term to favor state-action pairs from the data set. In this paper, we refine this idea by estimating the density of these state-action pairs to distinguish neighbourhoods. The resulting regularization guides the policy toward meaningful unseen regions, improving the learning process. We hence introduce Density Conservative Q-Learning (D-CQL), a sound batch RL algorithm that carefully penalizes the value function based on the information collected in the state-action space. The performance of our approach is outlined on many classical benchmark in batch RL.

1 INTRODUCTION

Transposing recent successes of Reinforcement Learning (RL) from domains such as recommendation systems (Rojanavasu et al., 2005; Zheng et al., 2018), video games (Mnih et al., 2013), go (Silver et al., 2017), to real-world systems is not possible without facing many challenges (Dulac-Arnold et al., 2021). One of those being that in many real-world applications, direct access to the system is limited or even forbidden. A learning controller may incorrectly assess the implications of its actions and damage the system. Batch (or offline) Reinforcement Learning (Lange et al., 2012; Levine et al., 2020) provides a framework to address those RL problems when no interaction with the system is allowed. In place, the learner is given a data set collected under a (possibly unknown) *behavioural* policy and has to derive the most efficient policy out of this data set.

One of the main issue in this setting is known as the value-function over-estimation problem (Fujimoto et al., 2019; Levine et al., 2020; Kumar et al., 2019a; Wu et al., 2019). When the data set covers only a small subset of the state-action space, the agent typically wrongly extrapolates the value function related to *out-of-distribution* (OOD) pairs far from the data set. This incorrect value estimate is then used as a target in the learning process, leading to highly increasing estimates which may lead to a disastrous learned policy. This problem has been extensively studied in the traditional online Reinforcement Learning setting (Sutton & Barto, 1998). However in this case, this issue is quite naturally alleviated: when a value function becomes high whether because the region is promising or because of an erroneous over-optimism, it will drive the agent to visit the related state-action pair. Therefore, the agent will have the chance to directly check the consequences of such pairs and its estimation can be corrected if needed. Incorporating uncertainty with *Ensemble Learning* (van Hasselt, 2010; van Hasselt et al., 2016; Anschel et al., 2017) can also enhance learning.

In the offline case, since no inspection can be done to investigate the accuracy of the estimates, those methods cannot be used, and hence classical deep *off-policy* methods may dramatically fail (Fujimoto et al., 2019; Levine et al., 2020). Additional approaches must thus be investigated to build robust and efficient agents. An important family of state-of-the-art algorithms addressing this problem focus on constraining the learned policy to stay close to the data set either by minimizing

its distance to the behavioural policy (Siegel et al., 2020a; Wu et al., 2019; Kumar et al., 2019b) or by penalizing unseen state-action pairs (Luo et al., 2019; Kumar et al., 2020; Yu et al., 2021). In particular, Kumar et al. (2020) propose to use a lower bound on the value function as a proxy during the policy optimization process. This bound should be tight when state-action pairs are contained in the data set and loosened otherwise. Following this direction, they proposed Conservative Q-Learning (CQL) that introduces a penalization on the value functions associated to OOD actions. Nevertheless, given the lack of information regarding the *behavioural policy*, their regularization remains abrupt and might be problematic in practice. First, the resulting lower bound is loosened with the recommended distributions. Second, their approach does not consider neighbourhoods of *actions* and could therefore over-penalize interesting regions of the action space. This problem is particularly relevant when the data set has been gathered with a behavioural policy with medium performances: actions close to the data set distribution leading to better results are very likely to appear in the optimization process and they must not be too heavily penalized.

In this paper, we tackle these problems by introducing a novel penalization based on an estimation of the probability density function of the data set instead of the behavioural policy. Besides being easier to learn than the behavioural policy, this density is able to provide information on which regions are near the data set and thus safe to learn from, and which are the ones far from it and prone to over-estimation errors. We show how to integrate this penalization to refine CQL, leading to our proposed algorithm namely *Density Conservative Q-Learning* (D-CQL). On top of instigating those relevant information, we show that this new regularization leads to a more appropriate lower bound on the value function. We finally investigate empirically the relevance of our approach.

2 Preliminaries and Motivations

2.1 Notations and Policy Iteration

The agent-environment framework is modeled as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, r, P, \gamma)$ defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition kernel $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, a reward function $r : \mathcal{S} \times \mathcal{A} \to [R_{min}, R_{max}]$ and a discount factor $\gamma \in]0, 1[$. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a decision rule mapping a state over a distribution of actions. The value of a policy is measured through the value function $V^\pi(s) = \mathbb{E}_P \left[\sum_{t=0} \gamma^t r(s_t, a_t) \mid s_0 = s,\ a_t \sim \pi(\cdot|s_t)\right]$ and its associated Q-value function $Q^\pi(s, a) = \mathbb{E}_P \left[\sum_{t=0} \gamma^t r(s_t, a_t) \mid s_0 = s,\ a_0 = a,\ a_t \sim \pi(\cdot|s_t)\ \forall t \geq 1\right]$. The goal is to find the policy maximizing these value functions.

Let r, Q be matrices associated to all state-action pairs and P^π be the transition matrix induced by the policy π: $P^\pi(s, s') = \mathbb{E}_{a\sim\pi(\cdot|s)} \left[P(s'|s, a)\right]$. Value functions can be learned by iterating the Bellman operator defined for the Q-function as $\mathcal{B}^\pi Q = r + \gamma P^\pi Q$ where $P^\pi Q(s, a) = \mathbb{E}_{s'\sim P(\cdot|s,a), a'\sim\pi(\cdot|s)} \left[Q(s', a')\right]$. This operator is a γ-contraction (Puterman, 1994), hence having a unique fixed point Q^π. Many algorithms rely on the *Policy Iteration* scheme (Lillicrap et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2018), where the agent alternates between *Policy Evaluation* (PE) with the computation of Q^π and *Policy Improvement* (PI) by maximizing the learned Q-values:

$$Q^{k+1} \leftarrow \arg\min_Q \mathcal{B}^{\pi^k} Q^k - Q, \qquad \text{(policy evaluation)}$$

$$\pi^{k+1} \leftarrow \arg\max_\pi \mathbb{E}_{a\sim\pi} \left[Q^{k+1}(\cdot, a)\right]. \qquad \text{(policy improvement)}$$

This iterative process converges towards the optimal policy (Sutton & Barto, 1998; Santos & Rust, 2004). Since no knowledge is assumed on the environment, these steps are commonly solved using samples from a data set $\mathcal{D} = \{(s_i, a_i, r_i)_{i=1}^N\}$. The expectation under $P(\cdot|s, a)$ is now estimated using samples and leads to the *empirical* Bellman operator $\hat{\mathcal{B}}$. Actions that belong to the data set distribution given a state s are denoted as in-distribution (ID) and the ones far from it are out-of-distribution (OOD). We slightly abuse notations and consider $\mathcal{D}$ is also the distribution on $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ induced by the data set and propose:

$$Q^{k+1} \leftarrow \arg\min_Q \mathbb{E}_{s,a,s'\sim\mathcal{D}} \left[\left(\hat{\mathcal{B}}^{\pi^k} Q^k(s, a) - Q(s, a)\right)^2\right], \qquad \text{(approximate policy evaluation)}$$

$$\pi^{k+1} \leftarrow \arg\max_\pi \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi(\cdot|s)} \left[Q^{k+1}(s, a)\right]. \qquad \text{(approximate policy improvement)}$$

The Q-values and the policy π are commonly estimated using Neural Networks that are trained with gradient optimization methods. In the approximate policy evaluation step, an important aspect is the expectation of actions from $\pi(\cdot|s')$ in the bootstrapped target. This is where OOD actions may appear: extrapolation errors emerge then are back-propagated to eventually lead to highly over-optimistic estimates. Note that there are no OOD states in the Bellman update as the empirical operator only depends on a seen state s'. They might appear in model-based algorithms (Yu et al., 2021) but are, for now, out of the scope of model-free algorithms.

In this work, we focus on the batch setting where the agent cannot interact with the environment. A fixed data set has been gathered with an unknown behavioural policy π_β. No knowledge is assumed on π_β as the data set can come from different sources: optimal control, human or a mixture of them (Fu et al., 2020; Gülçehre et al., 2020). A major challenge is to find a good trade-off between staying close to the data set to avoid extrapolation errors and taking some liberty to overcome the suboptimality of the behaviour policy.

2.2 Conservative Q-Learning

CQL addresses the problem of Q-value overestimation exacerbated in batch RL (Fujimoto et al., 2019; Levine et al., 2020), caused by the back-propagation of a possibly incorrect bootstrapped target in the policy evaluation step. The error associated to the the target may be important when the consequences of a given state-action pair are unknown, and could lead to too optimistic estimates. In batch RL, the data set often describes a small subset of the state-action space thus OOD actions are likely to appear. Besides, the agent may never get the chance to visit the state-action pairs related to high (learned) Q-values and cannot correct its wrong estimations.

Kumar et al. (2020) propose to penalize the Q-values associated to actions not described by the data set while keeping intact actions from the data set. This greatly stabilizes learning: the Q-estimates no longer take drastically high values, and it implicitly drives the agent to favor state-action pairs described in the data set. This penalization translates into minimizing the Q-values on an arbitrary distribution μ (*e.g.* uniform) and maximizing them on π_β formally expressed as:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha \, \mathbb{E}_{s\sim\mathcal{D}} \left[\left(\mathbb{E}_{a\sim\mu(\cdot|s)} \left[Q(s,a) \right] - \mathbb{E}_{a\sim\pi_\beta(\cdot|s)} \left[Q(s,a) \right] \right) \right] + \mathbb{E}_{s,a,s'\sim\mathcal{D}} \left[\left(\hat{\mathcal{B}}^{\pi_k} \hat{Q}^k(s,a) - Q(s,a) \right)^2 \right] . \tag{1}$$

Assuming no sampling errors and that $\operatorname{supp}\mu \subset \operatorname{supp}\pi_\beta$, this update provides a lower-bound on the expected Q-values on the distribution μ: $\mathbb{E}_{a\sim\mu}\left[Q^{CQL}(s,a)\right] \leq \mathbb{E}_{a\sim\mu}\left[Q^{\pi}(s,a)\right]$ for all $s \in \mathcal{S}$ with $\alpha > 0$. When μ is the current policy π, the lower bound is in the value function, that is $V^{CQL}(s) \leq V^{\pi}(s)$. This property may be desirable on applications where we must check if the current value function is above a certain threshold such as Constrained Policy Optimization (Achiam et al., 2017) and Conservative Exploration (Garcelon et al., 2020). However, for the common setting where μ is a uniform distribution over the action space, the lower bound is loosened to a uniform expectation, and the inequality in the value function no longer holds.

Another point of attention is the maximization term relying on the knowledge of the behavioural policy in equation 1. In the general setting, it is out of reach as the data set may be gathered in various ways. Kumar et al. (2020) propose to use the empirical data set distribution $\hat{\pi}_\beta(a|s) = \frac{\sum_{s',a'\in\mathcal{D}} \mathbb{1}_{[s'=s,a'=a]}}{\sum_{s'\in\mathcal{D}} \mathbb{1}_{[s'=s]}}$, a sum of Dirac over the data set actions. Not only does this further breaks the assumption $\operatorname{supp}\mu \subset \operatorname{supp}\hat{\pi}_\beta$ (as in practice $\operatorname{supp}\pi_\beta \not\subset \operatorname{supp}\hat{\pi}_\beta$), it also might be problematic on the penalization itself: the distance between the selected actions and the data set distribution is no longer considered explicitly. Actions not appearing in the data set will be equally penalized no matter their distance with $\mathcal{D}$. Thus, the regularization drives the agent to stay close to the behavioural policy. We believe having a smoother regularization that considers which neighbourhoods of the data set are relevant is important to get the best agent as possible.

3 Density Conservative Q-Learning

In this section, we introduce Density-CQL (D-CQL), our algorithm circumventing these issues.

3.1 A NEW WEIGHTING SCHEME ON THE OUT-OF-DISTRIBUTION ACTIONS

The penalization of CQL pushes the learned policy of the agent to stay very close to the behavioural policy as it penalizes any action that does not belong to the data set and thereby prevents the agent from investigating on potential relevant areas. This problem is of high interest, for example, when the data set is gathered with a distinctly sub-optimal policy but more efficient actions exist in the neighborhood of the sub-optimal ones. The chances of finding a better policy would be reduced as any action that does not belong to the data set will be equally penalized. Nevertheless, in view of the fixed nature of the batch setting and the high sensibility of the Bellman update, we should be cautious with respect to which regions the agent should focus on.

Bearing these complications in mind, we propose to refine the original regularization by appropriately weighting the actions according to how *uncertain* their consequences are given $\mathcal{D}$. To do so, we quantify the distance between the data set and the (sampled) penalized actions to soften the regularization. We propose a regularization based on the joined density of state-action pairs in the data set, denoted $\rho_\beta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$. This density describes how *likely* a state-action pair belongs to the data set and, by extension, how *uncertain* its associated Q-function is. Various techniques can be used to learn such a distribution. In fact, the literature regarding density estimation is quite abundant (Bishop, 2007; Uria et al., 2013; Kobyzev et al., 2020), even when dealing with high dimensional inputs such as images (Winkler et al., 2019; Papamakarios et al., 2019). Our method can also benefit from future developments in this area.

Using this density, we build a weighting scheme $\zeta_\nu(a|s)$[1] based on how uncertain the consequences of an action a, given s, are w.r.t. $\mathcal{D}$. It is integrated as follows:

$$\hat{Q}^{k+1} \leftarrow \arg\min_Q \alpha\, \mathbb{E}_{s\sim\mathcal{D}, a\sim\mu(\cdot|s)} \left[\zeta_\nu(a|s) Q(s,a)\right] + \mathbb{E}_{s,a,s'\sim\mathcal{D}} \left[\left(\hat{\mathcal{B}}^{\pi_k} Q^k(s,a) - Q(s,a)\right)^2\right] . \tag{2}$$

The weighting scheme $\zeta_\nu(a|s)$ is able to control how much the Q-value of an action a chosen under $\mu(\cdot|s)$ should be penalized and therefore satisfies our goal. Note that CQL corresponds to a particular version of D-CQL where $\zeta_\nu(a|s) = 1 - \delta_{a_\mathcal{D}}(a)$, where $\delta_{a_\mathcal{D}}$ denotes the Dirac over $a_\mathcal{D}$ (the actions associated to state s in the data set). The new regularization leads to a point-wise lower bound on the Q-values that is tight on *in-distribution* (ID) actions with a well-chosen ζ_ν. It thereby follows the classic Q-Learning procedure for ID actions, while only penalizing the Q-values associated to OOD actions depending on their distance with respect to the data set distribution, controlled by ζ_ν.

Theorem 1 *For any μ satisfying $\operatorname{supp}\mu \subset \operatorname{supp}\pi_\beta$ and assuming the absence of sampling errors, the Q-values obtained by iterating equation 2 are*

$$\forall s \in \mathcal{D}, a \in \operatorname{supp}\pi_\beta(\cdot|s),\ Q^{\text{D-CQL}}(s,a) = Q^\pi(s,a) - \alpha \left[(I - \gamma P^\pi)^{-1} \frac{\mu\, \zeta_\nu}{\pi_\beta}\right](a|s) , \tag{3}$$

and $\alpha > 0$ leads to a point-wise lower bound on the true Q-values that becomes tight whenever $\zeta_\nu(a|s) = 0$.

The proof of this theorem follows closely the one in (Kumar et al., 2020, Theorem 3.1), but we still include it in Appendix A for completeness.

3.2 CHOICE OF THE WEIGHTING SCHEME

The weighting scheme should capture the uncertainty of an action a given a state s. With $a_\mathcal{D}$ the action present in the data set related to the state s, we propose the following indicator:

$$\zeta_\nu(a|s) = \max\left(\nu\ ,\ 1 - \frac{\rho_\beta(s,a)}{\rho_\beta(s,a_\mathcal{D})}\right) . \tag{4}$$

[1] ζ_ν is a weighting scheme, not a density. However, we abuse notations by taking a conditional notation of ζ on the state s to highlight that it depends only on the action given a state.

Notice that the weighting scheme contains a few tricks on top of the density information. First, because the density may take high values in some areas, especially when dealing with narrow distributions, we normalize the weighting scheme by $\rho_\beta(s, a_\mathcal{D})$. The state-action pair $(s, a_\mathcal{D})$ belongs to the data set, so its associated density is able to scale the weighting scheme well. It also frees its dependency on the state s. Second, considering the propensity of the Q-values to take extremely high values rapidly, the weighting scheme must be below 1. Last but not least, making a too harsh distinction between ID and OOD actions may result in an unstable learning, hence the introduction of the hyper-parameter ν.

Hyper-parameter ν We need $\nu = 0$ for the lower bound of Theorem 1 to be tight. Indeed, when $\nu = 0$, the Q-values of ID actions are not penalized as $\zeta_\nu = 0$. It follows the intuition of penalizing only Q-values associated to OOD actions. However, since OOD actions do not appear in the Bellman loss term of Equation (2), the minimum of their Q-values is unbounded which eventually affects the performances of the algorithm. By setting $\nu > 0$, the regularization term will also include ID actions and this minimization will be counterbalanced by the Bellman loss that tends to maximize their Q-values. In other words, the lower ν, the greater the difference between OOD and ID Q-values to balance the objective. In practice, it is common to have knowledge on the quality of the behavioural policy and on the system, which can help setting the right ν. For instance, if we know that an expert has gathered the data set, we will want to stay close to it and set ν close to 0. Otherwise, if we know that the exploration has not been good and that there is no risk of harming the system, we can choose a bigger ν to allow the controller to consider uncertain actions. In Appendix C.2, we conduct a detailed analysis of the effects of the OOD penalization with varying ν.

4 Practical Algorithm

Following the ideas expressed in (Kumar et al., 2020, Appendix A), we add an entropic regularization to Equation (2) that leads to the following update for Density Conservative Q-Learning (D-CQL):

$$\begin{aligned}\hat{Q}^{k+1} \leftarrow \arg\min_Q \alpha\, \mathbb{E}_{s\sim\mathcal{D}}\left[\log \sum_{a\in\mathcal{A}} \exp\left(\zeta_\nu(a|s) Q(s,a)\right)\right] \\ + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(\mathcal{B}^{\pi_k}\hat{Q}^k(s,a) - Q(s,a)\right)^2\right].\end{aligned} \tag{5}$$

The `log-sum-exp` operator can be seen as a soft differential maximum operator. Equation (5) only penalizes the highest Q-values weighted by ζ_ν. This expression exacerbate D-CQL's behaviour described in Section 3.2 when ν is close to zero. Same conclusion holds, the lower ν, the greater the difference between OOD and ID Q-values to balance the objective. In practice, this operator is computed using *Importance Sampling* (IS) using M action samples from both the current policy and the uniform distribution $\mathcal{U}(\mathcal{A})$. We describe D-CQL in Algorithm 1.

Algorithm 1 D-CQL

Learn ρ_β with behavioural cloning
Initialize Q_θ and π_Φ
for $k \in (1, \dots)$ **do**
  Sample a batch $\mathcal{B}$ from $\mathcal{D}$
  For each state $s \in \mathcal{B}$ sample M actions from $\mathcal{U}(\mathcal{A})$, and M actions from $\pi_\Phi(\cdot|s)$
  Update Q_θ with gradient descent on Equation 5 using Importance Sampling
  Update π_Φ with gradient ascent on Equation policy improvement
end for

5 Related Works

Learning a policy from a fixed data set has been of high interest in Reinforcement Learning. A first way to address this problem is using Behavioural Cloning. When the data set is complete and

comes from an expert policy, it may achieve great successes as shown in special cases of *autonomous driving* (Pomerleau, 1988) and *flying* (Sammut et al., 1992). However, when the agent finds itself in a situation not described by the data set, it may choose catastrophic actions and its performance can quickly degrade (Codevilla et al., 2019). Imitation Learning (Hussein et al., 2017; Ho & Ermon, 2016) aims to counter this problematic but still requires an expert policy. In real-world applications, the data set may come from sub-optimal policies and instead of reproducing the behavioural policy, the goal of Batch RL is to extract a *better* one. Recently, many algorithms have been introduced, all relying on making sure the learned policy stays close to the behavioural one, avoiding the presence of Out-Of-Distribution (OOD) actions (Levine et al., 2020). It can take several forms.

Many recent algorithms cope with the distributional shift by directly avoiding the selection of OOD actions by the policy, preventing the agent to learn from highly extrapolated values. BCQ (Fujimoto et al., 2019) explicitly parametrize the policy to stay within a parametric ball around the behavioural policy. Similarly, SPIBB (Laroche et al., 2019) restricts the support of the learned policy to remain in the support of the behavioural policy. Other methods - such as BRAC (Kumar et al., 2019a) and BEAR (Wu et al., 2019) - modify the RL objective with a measure of closeness between learned and behavioural policy. AWR (Peng et al., 2019) or ABM (Siegel et al., 2020b) build a Trust Region around the behavioural policy. Rather than a constraint between the policies, AlgaeDICE (Nachum et al., 2019) and OptiDICE (Lee et al., 2021) focused on the *state-action stationary distributions* which provided competitive results. While successful on a variety of tasks, they do not explicitly counter the over-estimation problem that may prevent learning (Kostrikov et al., 2021). Another line of work is to enhance the robustness of the agent while leaving the objective unchanged. REM (Agarwal et al., 2020) chooses to not modify the objectives, but uses Ensemble Learning to build a mixture of targets stabilizing the updates. In practice, it rarely competes with the above methods.

More related to our work is (Dadashi et al., 2021; Rezaeifar et al., 2021) where a clever pessimistic bonus is introduced in the rewards acting as anti-exploration. It represents the distance of the selected action with the one who would be selected by the behavioural policy. The authors showed it acted similarly than a KL-penalty between the learned and behavioural policy, linking their work with Wu et al. (2019) and Kumar et al. (2019a). However, playing with the rewards might lead to a completely modified goal (Ng et al., 1999) and finding an appropriate reward modification might be complicated (Harutyunyan et al., 2015). We argue that modifying the Q-values lead to a better policy control as the whole objective is considered. In this line of search, Kostrikov et al. (2021) augmented the Q-values with the entropy of the behavioural policy and constrained them with as Fisher divergence term. In contrast, rather than modifying the RL objective with an explicit penalty on the data set distance, we use the density as an uncertainty surrogate in the value function estimation.

Note our data set density estimation can be combined with most presented methods. Especially, it could replace the "anti-exploration" bonus or be introduced to improve the Behavioural Cloning step in (Wu et al., 2019) required to build the measures of closeness between policies.

6 EXPERIMENTS

In this section, we evaluate D-CQL on various settings. Our algorithm was designed to take the most of data sets collected with a distinctly sub-optimal policy, where the trade-off between staying close to the behavioural policy and discovering better actions is culminating. An ablation study on ν and on the density estimation technique used can be found in Appendix C.2.

6.1 DATA SETS

Our algorithm is first tested on four different data sets gathered on two classic control environments with continuous actions: Pendulum and MountainCarContinuous (Moore, 1990). For each environment, two data sets were gathered with a `bad` and `medium` policy, where the policies have been created by running a Soft-Actor-Critic agent (Haarnoja et al., 2018) and early-stopping the training.

Pendulum is an inverted pendulum swing up problem, where the reward is a smooth function contained in $[-16, 0]$. The goal is to is to swing it up so it stays upright for 200 time-steps. The expert policy induces an average return of -150, and the `bad` and `medium` policies respectively induce an average return of -1000 and -700. The data set possesses 20 thousand transitions. MountainCarContinuous is also a classic environment in Reinforcement Learning. A car must drive up a hill,

but must first drive back and forth to be able to climb it. Unlike Pendulum, the reward is sporadic: it takes the value of $+100$ when the goal is reached, and $-$*energy* else. The `bad` data set is composed of 1 success out of 100 trajectories, and the `medium` data set 4 successes out of 50. We have found both CQL and D-CQL easily solve this environment when the number of successes is higher. The expert policy induces an average return of 95.

Finally, we test our agents on the `medium` data sets from D4RL which is closer to real-world performances. To further match a realistic setting we only consider a reduce proportion, $X\%$, of the data sets. Indeed, huge data collection might not be that realistic in practice (*e.g.* low acquisition frequency) and learning on such huge data sets can be problematic w.r.t. computation time.

6.2 Experimental protocol

Learning the density D-CQL relies on an accurate density estimation step. Though any technique can be used, we propose to use a highly expressive method: *Normalizing Flows* (Rezende & Mohamed, 2015; Kobyzev et al., 2020). This choice was fueled with their successes on high dimensional distributions (Papamakarios et al., 2019) and their elegance. Indeed, they are known to generalize and scale well from complex distributions (Winkler et al., 2019) - including images (Kobyzev et al., 2020) - while keeping a tractable learning. We optimize the model hyper-parameters with a grid search where each data set was split into a *train* and a *test* set. The density estimator was then trained on the whole data set using the best selected hyper-parameters. A comparison of different density estimation algorithms used for D-CQL can be found in Appendix C.2.

Learning the agents We compare our agents with CQL[2], a state-of-the-art method that also penalizes OOD actions in the approximate policy evaluation step. Experiments on the classic control data sets have been done with a grid search on all the hyper-parameters, including experiments with CQL. Considering the significant length of the experiments on D4RL, the hyper-parameter search was done on 1 million (M) gradient steps. Additional details are reported in Appendix C.1.

Metric We compare the agents' performances using the *normalized average return* (NAR):

$$\text{normalized score} = 100 * \frac{\text{score} - \text{random score}}{\text{expert score} - \text{random score}}$$

However, it is common knowledge that these methods are unstable and a decrease in performance during learning (Aviral Kumar, 2021; Kumar et al., 2021). We thus propose to evaluate the best agents after 10M gradient steps. Thus, once the best hyper-parameters selected on 1M gradient steps, we further continue the learning for 9M steps for the best agent. Regarding classic control data sets, we observed that 100k updates were sufficient to draw a relevant comparison. We report the *overall best normalized average return*, *i.e.* the best normalized average return considering all epochs, and the *best final normalized average return* in Table 1 and Table 2.

6.3 Results

Results on the classic control data sets are shown in Table 1. We observe our refined regularization improve the agent's performance on both the *overall* and *final* metrics on 3 out of 4 tasks. The agent was able to take the most out of the neighbourhood of the data set and extract more efficient policies. Notice the greater variance on the MountainCarContinuous environment, explained by the sparsity of its reward function. In any case, D-CQL was able to better take advantage of the information provided by the data set.

Regarding the more complex D4RL's data sets, the results are reported in Table 2. Our agent was able to beat CQL respectively 5 and 6 times out of 9 w.r.t. the *overall* and *final* metric. On average, it D-CQL outperforms CQL by 3% and 7.5% on the *overall* and *final* metric.

Overall, both Table 1 and Table 2 illustrate the fact that D-CQL outperforms or is comparable with CQL. Detailed plots can be found in Appendix C.3. Furthermore note the decrease of performance for both methods w.r.t. gradient steps emphasized by the differences between the *best* and *final*

[2]The results for CQL have been generated with the code `https://github.com/young-geng/CQL`.

Table 1: Normalized scores between CQL and D-CQL on the classic control data sets.

DATA SET	METRIC (100K)	CQL	D-CQL	IMPROVEMENT (%)
PENDULUM-BAD	BEST OVERALL	$\mathbf{64.2 \pm 13.4}$	$\mathbf{67.7 \pm 11.9}$	**5.4**
	BEST FINAL	$\mathbf{50.1 \pm 6.6}$	$\mathbf{58.2 \pm 15.6}$	**16.1**
PENDULUM-MEDIUM	BEST OVERALL	$\mathbf{81.3 \pm 9.4}$	$\mathbf{88 \pm 6.7}$	**8.3**
	BEST FINAL	$\mathbf{74.7 \pm 23.7}$	$\mathbf{83.4 \pm 9.5}$	**11.7**
MOUNTAINCARC-BAD	BEST OVERALL	$\mathbf{40 \pm 17.7}$	$\mathbf{59.5 \pm 36.6}$	**48.7**
	BEST FINAL	$\mathbf{34.9 \pm 38.3}$	$\mathbf{47 \pm 29.1}$	**34.9**
MOUNTAINCARC-MEDIUM	BEST OVERALL	92 ± 0.9	92 ± 0.3	0
	BEST FINAL	60.3 ± 23.4	59.4 ± 27.4	-1.5
TOTAL MEAN	BEST OVERALL			**15.6**
	BEST FINAL			**15.3**

Table 2: Normalized scores between CQL and D-CQL on the D4RL MuJoCo medium data sets.

DATA SET	METRIC (10M)	CQL	D-CQL	IMPROVEMENT (%)
HALFCHEETAH-MEDIUM 10%	BEST OVERALL	14.7 ± 3.2	14.4 ± 3.8	-2
	BEST FINAL	$\mathbf{10.3 \pm 4.7}$	$\mathbf{12.6 \pm 4.5}$	**22.3**
WALKER2D-MEDIUM 10%	BEST OVERALL	$\mathbf{48.4 \pm 8.4}$	$\mathbf{50.4 \pm 3.5}$	**4.1**
	BEST FINAL	$\mathbf{44 \pm 5.4}$	$\mathbf{48.2 \pm 6.7}$	**9.5**
HOPPER-MEDIUM 10%	BEST OVERALL	81.5 ± 9.3	80.9 ± 7.7	-0.7
	BEST FINAL	43.7 ± 9.3	37.2 ± 13.9	-14.9
HALFCHEETAH-MEDIUM 5%	BEST OVERALL	$\mathbf{15 \pm 3.9}$	$\mathbf{15.8 \pm 3.5}$	**5.3**
	BEST FINAL	$\mathbf{9.4 \pm 5.5}$	$\mathbf{11.1 \pm 4.1}$	**18.1**
WALKER2D-MEDIUM 5%	BEST OVERALL	40.5 ± 6.6	37.4 ± 6.3	-7.7
	BEST FINAL	33.5 ± 8.4	31.6 ± 6.4	-5.4
HOPPER-MEDIUM 5%	BEST OVERALL	$\mathbf{58 \pm 13}$	$\mathbf{67.1 \pm 8}$	**15.7**
	BEST FINAL	$\mathbf{43.6 \pm 14.9}$	$\mathbf{48.7 \pm 13.6}$	**11.7**
HALFCHEETAH-MEDIUM 1%	BEST OVERALL	18.6 ± 3.2	17.6 ± 2.8	-5.4
	BEST FINAL	$\mathbf{11.6 \pm 5.1}$	$\mathbf{15 \pm 4.6}$	**29.3**
WALKER2D-MEDIUM 1%	BEST OVERALL	$\mathbf{24.1 \pm 14.1}$	$\mathbf{26.7 \pm 15.4}$	**10.8**
	BEST FINAL	22.9 ± 16.5	20.2 ± 8.7	-11.8
HOPPER-MEDIUM 1%	BEST OVERALL	$\mathbf{64.1 \pm 9.7}$	$\mathbf{68.4 \pm 9.1}$	**6.7**
	BEST FINAL	$\mathbf{63 \pm 4.7}$	$\mathbf{64.4 \pm 7.9}$	**3.6**
TOTAL MEAN	BEST OVERALL			**3**
	BEST FINAL			**7.5**

performances. While out of the scope of this paper, it remains an open question in off-policy algorithms (Aviral Kumar, 2021; Kumar et al., 2021). Last but not least, our experiments shown that even though the methods might prevent the over-estimation of the Q-values, agents' performances have still non-negligible variances.

7 CONCLUSION

In this paper we propose to introduce the use of the data set density ρ_β which provides insights to guide the agent towards meaningful areas, inducing better performances in practice. This work focused on model-free agents that do not consider OOD states. However, the density ρ_β also yields information on OOD states which can be integrated on model-based algorithms.

In addition, our experiments highlighted two other points of interests: the decreasing effect and the relatively high variance of the batch agents. Despite some previous works aiming to cope with these problems, a complete understanding of these phenomenons is lacking which is crucial before batch RL can be applied on real-world systems.

ACKNOWLEDGMENTS

We thank Christophe Prieur and Bogdan Robu from GIPSA Lab of Grenoble-Alpes University for the useful discussions we had regarding this work. We also thank the anonymous reviewers for their feedback which improved the paper.

REFERENCES

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pp. 22–31. PMLR, 2017.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pp. 104–114. PMLR, 2020.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pp. 176–185. PMLR, 2017.

Aaron Courville Tengyu Ma George Tucker Sergey Levine Aviral Kumar, Rishabh Agarwal. Value-based deep reinforcement learning requires explicit regularization. 2021.

Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *ICCV*, pp. 9328–9337. IEEE, 2019.

Robert Dadashi, Shideh Rezaeifar, Nino Vieillard, Léonard Hussenot, Olivier Pietquin, and Matthieu Geist. Offline reinforcement learning with pseudometric learning. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2307–2318. PMLR, 2021.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *ICLR (Poster)*. OpenReview.net, 2017.

Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Mach. Learn.*, 110(9):2419–2468, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1582–1591. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2052–2062. PMLR, 2019.

Evrard Garcelon, Mohammad Ghavamzadeh, Alessandro Lazaric, and Matteo Pirotta. Conservative exploration in reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 1431–1441. PMLR, 2020.

Çaglar Gülçehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel J. Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, Jerry Li, Mohammad Norouzi, Matthew Hoffman, Nicolas Heess, and Nando de Freitas. RL unplugged: A collection of benchmarks for offline reinforcement learning. In *NeurIPS*, 2020.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowé. Expressing arbitrary reward functions as potential-based advice. In *AAAI*, pp. 2652–2658. AAAI Press, 2015.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NeurIPS*, pp. 4565–4573, 2016.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, 2017.

Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5774–5783. PMLR, 2021.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *NeurIPS*, pp. 11761–11771, 2019a.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *NeurIPS*, 32:11784–11794, 2019b.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *ICLR*. OpenReview.net, 2021.

Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pp. 45–73. Springer, 2012.

Romain Laroche, Paul Trichelair, and Remi Tachet des Combes. Safe policy improvement with baseline bootstrapping. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3652–3661. PMLR, 2019.

Jongmin Lee, Wonseok Jeon, Byung-Jun Lee, Joelle Pineau, and Kee-Eung Kim. Optidice: Offline policy optimization via stationary distribution correction estimation. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6120–6130. PMLR, 2021.

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.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

Yuping Luo, Huazhe Xu, and Tengyu Ma. Learning self-correctable policies and value functions from demonstrations with negative sampling. In *ICLR*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Andrew William Moore. Efficient memory-based learning for robot control. 1990.

Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*, 2019.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pp. 278–287. Morgan Kaufmann, 1999.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Dean Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *NeurIPS*, pp. 305–313. Morgan Kaufmann, 1988.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

Shideh Rezaeifar, Robert Dadashi, Nino Vieillard, Léonard Hussenot, Olivier Bachem, Olivier Pietquin, and Matthieu Geist. Offline reinforcement learning as anti-exploration. *arXiv preprint arXiv:2106.06431*, 2021.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 1530–1538. JMLR.org, 2015.

Pornthep Rojanavasu, Phaitoon Srinil, and Ouen Pinngern. New recommendation system using reinforcement learning. *Special Issue of the Intl. J. Computer, the Internet and Management*, 13 (SP 3), 2005.

Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *ML*, pp. 385–393. Morgan Kaufmann, 1992.

Manuel S. Santos and John Rust. Convergence properties of policy iteration. *SIAM J. Control. Optim.*, 42(6):2094–2115, 2004.

Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *ICLR*, 2020a.

Noah Y. Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin A. Riedmiller. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *ICLR*. OpenReview.net, 2020b.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

B. Uria, I. Murray, and H. Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *NeurIPS*, 2013.

Hado van Hasselt. Double q-learning. In *NeurIPS*, pp. 2613–2621. Curran Associates, Inc., 2010.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100. AAAI Press, 2016.

Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. In *Self-Supervision for Reinforcement Learning Workshop-ICLR 2021*, 2021.

Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. DRN: A deep reinforcement learning framework for news recommendation. In *WWW*, pp. 167–176. ACM, 2018.

A LOWER BOUND DETAILS

In this section, we re-state our main theorem and prove it. An additional insight is provided to handle the sampling errors.

Theorem 2 *For any μ satisfying $\operatorname{supp}\mu \subset \operatorname{supp}\pi_\beta$ and assuming there are no sampling errors, then the Q-values obtained by iterating equation 2 are*

$$\forall s \in \mathcal{D}, a \in \operatorname{supp}\pi_\beta(\cdot|s),\ Q^{\text{D-CQL}}(s,a) = Q^\pi(s,a) - \alpha\left[(I-\gamma P^\pi)^{-1}\,\frac{\mu\,\zeta_\nu}{\pi_\beta}\right](a|s)\,. \tag{6}$$

$\alpha > 0$ leads to a point-wise lower bound on the true Q-values that becomes tight whenever $\zeta(a|s) = 0$.

Proof 1 *This proof is done assuming the true Bellman operator $\mathcal{B}$ has been used instead of $\hat{\mathcal{B}}$. Setting the integrande of the derivative of 2 leads to*

$$Q^{k+1}(s,a) = \mathcal{B}^\pi Q^k(s,a) - \alpha\frac{\mu(a|s)\,\zeta(a|s)}{\pi_\beta(a|s)}\,. \tag{7}$$

The interest of ζ is clearly exposed in this equation: the learned Q-values will be highly conservative when associated to OOD actions. On the other hand, when $\zeta(a|s) \approx 0$, the update leads to an application of the classic Bellman operator and drive ID Q-values to stay close to their real associated Q-value.

Let μ, π_β and ζ the matrices associated to $\{\mu(a|s)\}$, $\{\pi_\beta(a|s)\}$ and $\{\zeta(a|s)\}$ for all $s \in \mathcal{D}$ and $a \in \operatorname{supp}\pi_\beta(\cdot|s)$. The operator associated to this update: $\mathcal{T}_\zeta : Q \to \mathcal{B}^\pi Q^k - \alpha\frac{\mu\,\zeta}{\pi_\beta}$ defined for any function Q on defined on states belonging to $\mathcal{D}$ and actions belonging to $\operatorname{supp}\pi_\beta(\cdot|s)$ is a γ-contraction. Its fixed point is

$$Q^{\text{D-CQL}} = Q^\pi - \alpha\left[(I-\gamma P^\pi)^{-1}\,\frac{\mu\,\zeta}{\pi_\beta}\right]\,. \tag{8}$$

Remark 1 *The sampling error can be handled using concentration inequalities and would lead to the following lower bound, that holds with probability $1-\delta$:*

$$Q^{\text{D-CQL}} \leq Q^\pi - \alpha\left[(I-\gamma P^\pi)^{-1}\,\frac{\mu\,\zeta}{\pi_\beta}\right] + (I-\gamma P^\pi)^{-1}\,\frac{C_{r,P,\delta}}{(1-\gamma)\,\sqrt{|\mathcal{D}|}}\,. \tag{9}$$

We refer to Appendix C of Kumar et al. (2020) for additional details.

B NORMALIZING FLOWS

We now describe the used architecture to estimate ρ_β based on Normalizing Flows. They are based on the change of variable for a diffeomorphism $f : X \to Y$ and two densities defined on X and Y

$$\rho_X(x) = \rho_Y\left(f(x)\right)\det\left(\operatorname{Jac} f(x)\right)\,. \tag{10}$$

We then learn a diffeomorphism f that maps the unknown distribution on X to a known distribution on Y, for example a multivariate normal distribution so that the factor $\rho_Y f(X)$ is efficiently calculable.

In our case, let $x = (s,a)$ and y its associated sample belonging to the Y distribution and let D their dimension. Let's assume there is only one differential mapping $y = f(x)$, and let $d < D$. We follow the architecture from RealNVP (Dinh et al., 2017) where f maps x to y as follows:

$$y_{1:d} = x_{1:d}\,,$$
$$y_{d+1:D} = x_{d+1:D} \odot \exp\left(s(x_{1:d})\right) + t(x_{1:d})\,, \tag{11}$$

where s and t are functions defined in $\mathbb{R}^d \rightarrow \mathbb{R}^{m-d}$ and $\odot$ is the Hadamard product. This transformation results in a simple formula for the Jacobian determinant: $\left|\det\left(\mathrm{Jac}\, f\right)(x)\right| = \exp\left(\sum_j s(x_{1:d})_j\right)$.

This mapping is then repeated N times where each input is randomly permuted before applying equation 11 to ensure each dimension is rightfully processed.

In all datasets, s and t are parametrized by a Neural Network with 1 layer. The number of mappings N as well as the number of units in each layer were selected thanks to grid search on respectively (5, 10, 20) and (64, 128, 256). Y was chosen as a Multivariate Normal distribution $\mathcal{N}(0, \mathcal{I})$.

C D-CQL EXPERIMENTS

For reproducible purposes, we now present additional details regarding the experiments. CQL's code was extracted from `https://github.com/young-geng/CQL`. On all plots, the abscissa represents the number of epochs and the ordinate the cumulative average return over 5 seeds. For the experiments on the classic control data sets, one epoch represents 100 gradient updates and 1000 updates for the ones on the D4RL data sets.

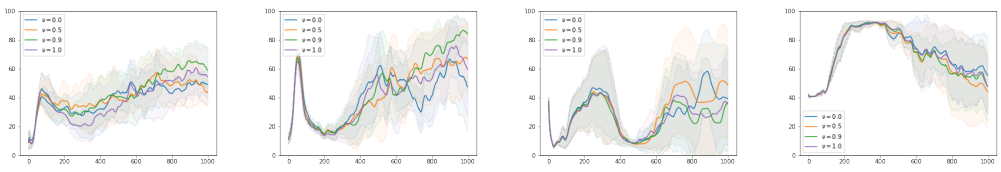
C.1 HYPER-PARAMETERS

We first present all the hyper-parameters used in the experiments. For the experiments on the data sets of D4RL, most of the hyper-parameters - learning rate, size of the networks, optimizer, ... - were unchanged to have a proper comparison with CQL. For the experiments on the classic control environments, the size of the hidden layers of the neural networks were minimized to 64 with the same number of layers.

The main hyper-parameters of the algorithms are α and ν, on which we performed a grid search for each experiment. This grid search was done on (1, 5, 10) for α and on (0.0, 0.1, 0.9, 1.0) for ν. As mentioned in Section 6, the grid search for the classic control environments (Pendulum and MountainCarContinuous) was done on the complete experiments, that is on the $100K$ gradient updates. Regarding the experiments on the D4RL MuJoCo data sets, the grid search was performed on only $1M$ gradient updates. The best hyper-parameters were then chosen and we continued this experiment on $10M$ updates.

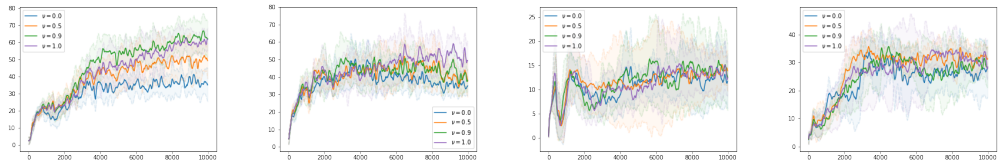
C.2 ABLATION STUDY

Study on ν First, we present an ablation study on the hyper-parameter ν on the classic control environments. For all environments, we set α to 5 and launched experiments with a varying ν. Considering the length of the experiments on the D4RL data sets, we performed the ablation study on `hopper-medium` (1% and 5%) and on `walker2d-medium` (1% and 5%).



(a) pendulum-bad (b) pendulum-medium (c) mountaincarc-bad (d) mountaincarc-medium

Figure 1: Performances of the algorithm w.r.t. to the hyper-parameter ν of the weighting scheme on the classic control data sets.



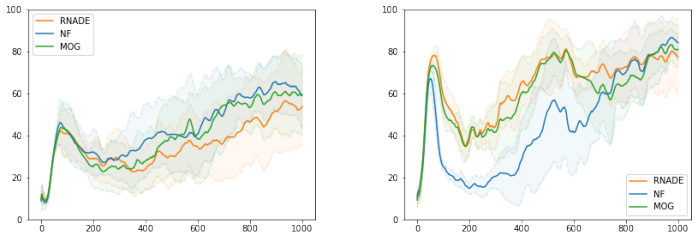
(a) hopper-medium-1% (b) hopper-medium-5% (c) walker2d-medium-1% (d) walker2d-medium-5%

Figure 2: Performances of the algorithm w.r.t. to the hyper-parameter ν of the weighting scheme for D4RL data sets.

The impact on the hyper-parameter ν varies according to the environment. Nevertheless, we observe the intuitive choice $\nu = 0$ often never produce the best results which were generated with a higher ν. We attribute this effect to the unbounded minimization of the Q-values towards minus infinity discussed in Section 3.2. This can be notably seen on `pendulum-medium` and `hopper-medium-1%`, where the best ν is 0.9. This high number has to be put in perspective as ζ_ν has been introduced as a multiplicative factor and is inside the soft differential maximum operator, both of which increase the importance of our weighting factor.

We also observe that having a distinction between ID and OOD actions in the penalization may not be useful in some environments. For instance, data sets from the walker2d environment do not need to make such distinction as illustrated by the results of $\nu = 1$. Forcing the Q-values to have a reasonable scale during learning seems to be enough for the Bellman update to work on this environment.

Study on the density estimation technique used We continue the ablation study by testing the robustness on the density estimation technique used. We compare the Normalizing Flows RealNVP with two other methods: the auto-regressive algorithm RNADE (Uria et al., 2013) and a Mixture of Gaussians (MOG). To make fair comparisons, RNADE and MOG densities have also been created with a grid search on their hyper-parameters, that is on the number of their components, the number and size of the layers of the Neural Network(s) used. We performed the study on data sets from the Pendulum environment, where we can see our algorithm is robust with respect to the density estimation technique used. It remains to be confirmed on higher dimensional environments.

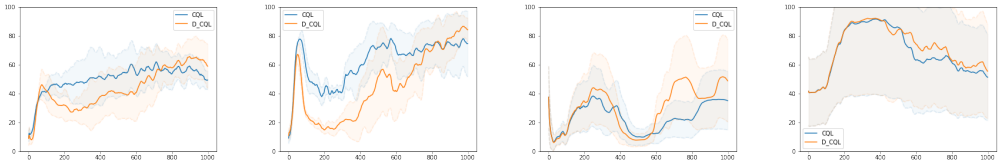


(a) pendulum-bad (b) pendulum-medium

Figure 3: Performances of the algorithm w.r.t. to the density estimation technique used.

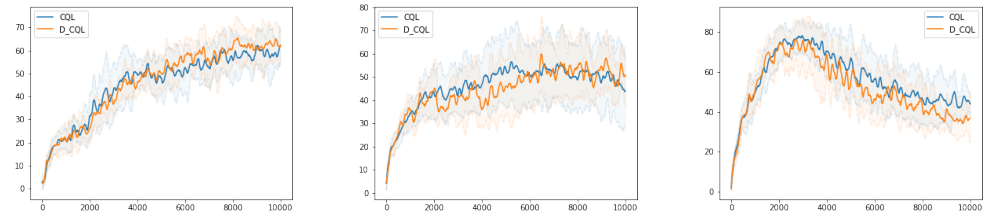
C.3 Comparison with CQL

We finally provide the plots for Table 1 and Table 2. The hyper-parameters of the experiments are the one chosen from the *final normalized average return* metric.

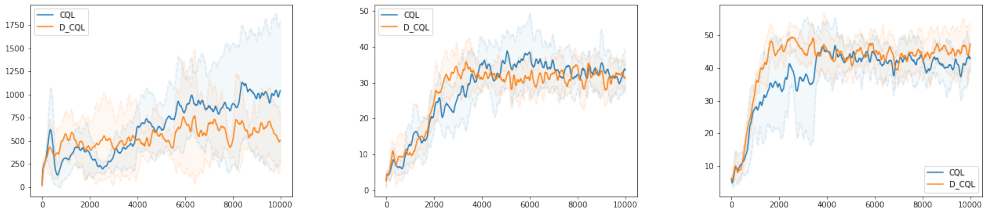


(a) pendulum-bad (b) pendulum-medium (c) mountaincarc-bad (d) mountaincarc-medium

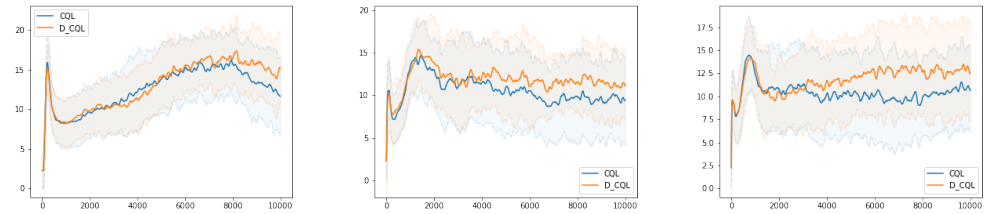
Figure 4: Performances of the algorithm compared to CQL on classic control data sets.



(a) hopper-medium-1% (b) hopper-medium-5% (c) hopper-medium-10%



(d) walker2d-medium-1% (e) walker2d-medium-5% (f) walker2d-medium-10%



(g) halfcheetah-medium-1% (h) halfcheetah-medium-5% (i) halfcheetah-medium-10%

Figure 5: Performances of the algorithm compared to CQL on D4RL data sets.