JOHANNES KEPLER
UNIVERSITY LINZ

Project Report

The Role of Dataset Generation in Offline RL

Author
Kajetan Schweighofer, BSc
k1556273

Supervisors
DI Marius-Constantin Dinu, BSc
Vihang Patil, MSc

Practical Work in AI (365.207) WS20

March 22, 2021

Abstract

There has been much interesting in Offline Reinforcement Learning lately, as the paradigm promises to give practitioners strong decision making tools that are learned from observed data only. The field can get rid of costly interaction with simulators are real world environments and can instead use huge pools of previously collected data. Off-policy algorithms are already suited to work on those previously collected datasets, but the question how these datasets must be collected is still an open question. Current work uses agents trained in the usual online setting to create the datasets, but recent studies showed very different performances depending on what data exactly was used. To be able to use Offline Reinforcement Learning on real world data, the effects of the dataset have to be understood and incorporated accordingly, which this work tries to make a first step towards.

1 Introduction

Offline Reinforcement Learning tries to ameliorate one of the most detrimental limitations of reinforcement learning, the interaction with the environment. Not only is this often the bottleneck when training deep reinforcement learning agents, but training with direct interactions might be impossible in real world environments for various reasons. Examples for this would be autonomous driving or decisions in the medical domain, where no untrained policy could be applied for safety reasons. Both domains could be simulated, but unfortunately this leaves a gap between simulation and real world samples one has to bridge later on and might not be trivial [Bousmalis et al., 2017]. It is more appealing to just use logged real world experiences to train a policy, which is feasible with for example all off-policy model-free deep reinforcement learning algorithms this work will focus on.

Further, there is no need to temper the imminent problem of exploration vs. exploitation as no additional samples can be collected. The agent can therefore solemnly focus on exploiting the data at hand, which deep learning excels at in the supervised learning domain.

Various recent results reported poor performance of ususal off-policy deep reinforcement learning algorithms in an offline setting, mainly due to an extrapolation error that stems from evaluating state-action pairs not contained in the provided offline dataset. [Fujimoto et al., 2019b]. The erroneous extrapolation is then propagated through temporal difference updates which are at the heart of most off-policy algorithms, causing action-value overestimation or even divergence and result in poor performance. Contrary to that, [Agarwal et al., 2020b] reported very optimistic results of standard off-policy methods on the Atari environment.

This report aims to look at the reason for the performance gap between [Agarwal et al., 2020b] and [Fujimoto et al., 2019a] that both operate on the same Atari environment reporting very different results even for the same algorithms. The hypothesis is, that the different dataset generation strategies they use lead to the performance gap. [Agarwal et al., 2020b] does simply store the full experience replay buffer of the behavioral policy to train the offline policies later on while [Fujimoto et al., 2019a] generates trajectories with a single behavioral policy with varying ϵ for the ϵ-greedy policy obtained after online training. While the first experiments take a look into the differences between the empirical performance of the two, the latter ones will take a look into the dataset diversity and state coverage to verify that the datasets differ in this respect.

2 Off-Policy Reinforcement Learning

Reinforcement Learning relies on formalizing the environment as Markov Decision Process (MDP) [Puterman, 1994], with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a stochastic reward function $R(s, a)$ for $s \in \mathcal{S}$ and $a \in \mathcal{A}$, transition dynamics $P(s'|s, a)$ where $s', s \in \mathcal{S}$ and $a \in \mathcal{A}$ and a discount factor $\gamma \in [0, 1)$ which is needed to extend the formalism to infinite episodes. The policy π is in general defined as stochastic mapping that maps each state s to a distribution over actions a. In the case of value based learning which is the focus here, the policy maps the state s to an action a by selecting the action (ϵ-)greedy with respect to the action-value.
The action-value denoted as $Q^\pi(s, a)$ and therefore often referred to as Q-value, is defined as the expected cumulative discounted future reward. For basically all practically relevant tasks, the state space is too large to be tabulated, therefore function approximators like neural networks are used to approximate the action-value that is given by

$$Q^\pi(s, a) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{2.1}$$

$$s_0 = s \in \mathcal{S}, a_0 = a \in \mathcal{A}, s_t \sim P(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t)$$

which is then usually denoted $Q_\theta^\pi(s, t)$ to make the parametrization explicit. Reinforcement Learning algorithms strive to find an optimal policy π^* that reaches maximum expected return, or stated differently, $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$ for all π, s, a.
The optimal policy π^* can be described by acting greedily on the optimal Q-value Q^*. The optimal Q-value can be decomposed for every step, known as Bellman optimality equation [Bellman et al., 1957]:

$$Q^*(s, a) = \mathbb{E}\ R(s, a) + \gamma \mathbb{E}_{s' \sim P} \max_{a' \in \mathcal{A}} Q^*(s', a') \tag{2.2}$$

This equation is utilized by the off-policy algorithm Q-learning [Watkins and Dayan, 1992] that iteratively improves its approximation of Q^* by repeatedly regressing it by means of the samples generated from the RHS of equation 2.2. As the state space is usually high-dimensional (e.g. raw image input), one resorts to use neural networks as function approximators to estimate the optimal Q-value Q^*_θ, where θ denote the parameters of the underlying neural network. To stabilize training, it is usual that the target network $Q^*_{\theta'}$ on the RHS of equation 2.2 is fixed for a number of timesteps until it is set to the network Q^*_θ. Another possibility is to regress the target network $Q^*_{\theta'}$ towards Q^*_θ in a soft way.

Equation 2.2 is the reason for the **extrapolation error**. Consider the case that s' is not within our dataset, then $Q^*_\theta(s', a')$ might be arbitrarily poor as we can not guarantee what action-value the function approximator assigns to this unseen state-action pair. The problem in this regard is the $\max_{a' \in \mathcal{A}}$, which leads to the highest positive approximation of the action-value being used for the update. In an online setting, the policy would simply visit a state-action pair that has such a high estimate and would receive a corrective feedback for its action-value, but this is not possible in the offline setting. The algorithms must therefore generalize as well as possible with the data at hand without such overestimations. To this end, five different off-policy algorithms are introduced, starting from simple DQN [Mnih et al., 2013] towards more robust offline RL algorithms.

2.1 Deep Q-Network (DQN)

Deep Q-Network (DQN) [Mnih et al., 2013] utilizes a Convolutional Neural Network to approximate the Q-values in order to implement a Q-learning algorithm. An ϵ-greedy policy where the ϵ is decayed throughout the training process is used. DQN minimizes the temporal difference error Δ_θ through minimizing the loss $\mathcal{L}(\theta)$ of the network Q_θ by adaptive optimizers. The batches given to the optimizers are past interactions that are stored in an experience replay buffer $\mathcal{D}$. These are not all past samples, but a running window of fixed size that is a design choice. The buffer is needed as neural networks expect the data to be i.i.d. and are usually trained on more than just one sample to have more stable gradients. Note that following the findings of [van Hasselt et al., 2015], a fixed target network $Q_{\theta'}$ that is updated to match Q_θ only after a certain number of updates is used to make learning more stable.

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \big[l_\lambda(\Delta_\theta(s, a, r, s')) \big] \tag{2.3}$$
$$\Delta_\theta(s, a, r, s') = Q_\theta(s, a) - r - \gamma \max_{a'} Q_{\theta'}(s', a')$$

where r is the reward of the current transition. l_λ is the Huber loss [Huber, 1964], given by:

$$l_\lambda(x) = \begin{cases} \frac{1}{2}x^2, & \text{if } |x| \leq \lambda \\ \lambda(|x| - \frac{1}{2}\lambda), & \text{else} \end{cases} \tag{2.4}$$

A graphical explanation of the architecture is depicted in Figure 1.

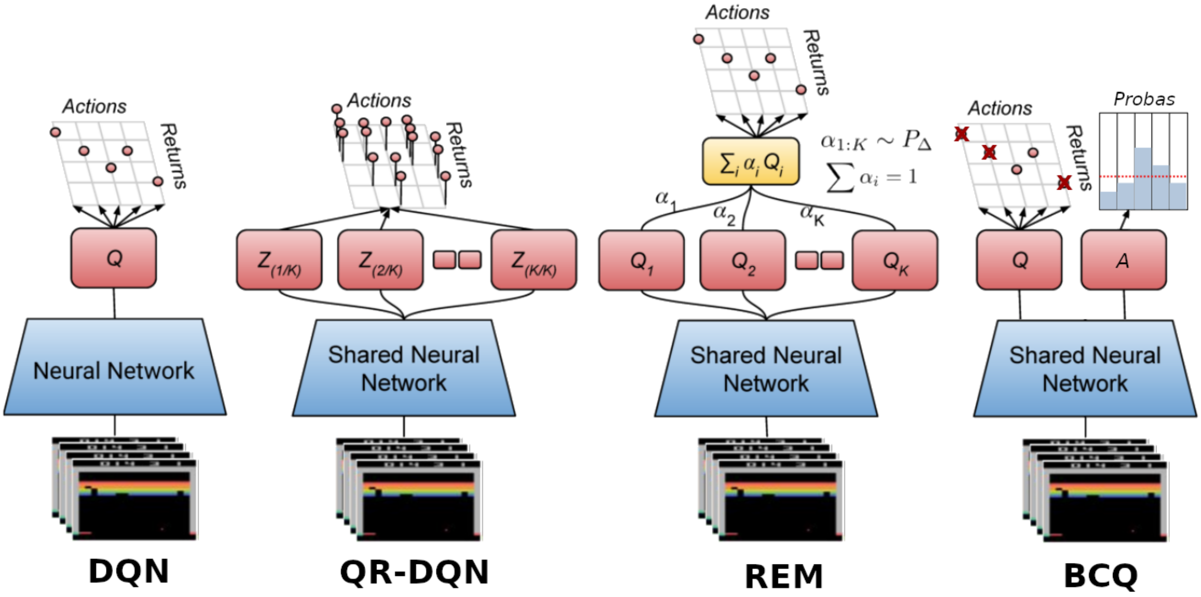

Figure 1: Overview of off-policy algorithms utilized in the offline setting, modified from [Agarwal et al., 2020b]

2.2 Quantile Regression Deep Q-Network (QR-DQN)

Another take on the problem is to not just learn a single estimate of the Q-value, but a distribution. Such algorithms estimate the density over returns for each state-action pair, denoted as $Z^\pi(s,a)$. Therefore the Bellman optimality equation must be expressed in a distributional manner:

$$Z^*(s,a) \overset{D}{=} r + \gamma Z^*(s', \text{argmax}_{a' \in \mathcal{A}} Q^*(s',a')) \tag{2.5}$$

$$r \sim R(s,a), s' \sim P(\cdot|s,a)$$

where $\overset{D}{=}$ denotes distributional equivalence. The Q-value is then given as the expected value of the distribution. Quantile Regression DQN (QR-DQN) [Dabney et al., 2017] uses a uniform mixture of K different delta functions to approximate the distribution:

$$Z_\theta(s,a) := \frac{1}{K}\sum_{i=1}^{K} \delta_{\theta_i(s,a)} \tag{2.6}$$

$$Q_\theta(s,a) = \frac{1}{K}\sum_{i=1}^{K} \theta_i(s,a) \tag{2.7}$$

QR-DQN is the state of the art algorithm on the Atari environment among algorithms that do not utilize prioritized experience replay or n-step updates.

2.3 Random Ensemble Mixture (REM)

Ensemble methods are usually deployed in supervised settings to make prediction more robust. The easiest idea in the reinforcement learning domain is, to predict multiple Q-values and take the mean as prediction. This can be seen as multiple instances of e.g. DQN, where we have separate networks that we all train sequentially. As ensemble methods generally tend to get better, the more approximators are included, an efficient method to train the ensemble is needed. Inspired by dropout [Srivastava et al., 2014], Random Ensemble Mixture (REM) [Agarwal et al., 2020b] strives to achieve exactly this. Multiple Q-value approximators are used, similar to the naive DQN ensemble proposed above, but they are trained simultaneously against their target networks. This is possible as a random convex combination of those Q-values again approximates a Q-value itself. That is trivial to see at the fixed point, where the Q-value given by each approximator coincides. For training, a categorical distribution α that defines a convex combination of the K estimates is drawn for each batch. Other than that, the loss $\mathcal{L}(\theta)$ is basically that of DQN:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\big[\mathbb{E}_{\alpha\sim P_\Delta}\left[l_\lambda(\Delta_\theta^\alpha(s,a,r,s'))\right]\big] \tag{2.8}$$

$$\Delta_\theta^\alpha(s,a,r,s') = \sum_k \alpha_k Q_\theta^k(s,a) - r - \gamma \max_{a'} \sum_k \alpha_k Q_{\theta'}^k(s',a')$$

To select an action, the average of the K different Q-value estimates is used:

$$Q(s,a) = \frac{1}{K}\sum_{k=1}^{K} Q_\theta^k \tag{2.9}$$

A very easy distribution P_Δ that works empirically is given by sampling K different α_k' from the uniform distribution and normalizing them, to get a valid categorical distribution:

$$\alpha_k' \sim \mathcal{U}(0,1) \qquad \alpha_k = \frac{\alpha_k'}{\sum_i^K \alpha_i'} \tag{2.10}$$

2.4 Batch Constrained Q-Learning (BCQ)

Batch Constrained deep Q-learning (BCQ) [Fujimoto et al., 2019b] utilizes a state conditioned model of the dataset $\mathcal{D}$. This model is used to compute the probability p_b of each action being sampled by the behavioral policy $\pi_b(s)$ that generated the dataset, thus eliminating actions in the current state that are unlikely and thus potentially poorly estimated by the offline policy, simply because they were not part of the dataset. This probability is estimated by means of a generative model that given the state, is trained in a supervised fashion with cross-entropy against the action the behavioral policy took. The offline policy is therefore not only expressed by the argmax of the Q-value, but is given by:

$$\pi(s) = \underset{a \in \mathcal{A} \text{ where } p_b(a) > \tau}{\operatorname{argmax}} Q_\theta(s, a) \tag{2.11}$$

where τ is the minimal probability an action must be assigned by the generative model. The resulting algorithm is quite similar to the basic DQN algorithm:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}\left[l_\lambda(\Delta_\theta(s, a, r, s'))\right] \tag{2.12}$$

$$\Delta_\theta(s, a, r, s') = Q_\theta(s, a) - r - \gamma \max_{a'} Q_{\theta'}(s', a')$$

$$\text{with} \qquad a' = \underset{a' \in \mathcal{A} \text{ where } p_b(a') > \tau}{\operatorname{argmax}} Q_\theta(s', a')$$

The max values action gets selected with the current network Q_θ and is evaluated by the target network $Q_{\theta'}$. Note that $\tau = 0$ results again in DQN and $\tau > \frac{1}{2}$ results in imitation learning, thus the most likely action given the dataset is returned no matter what Q-value the action has. Of course the generative model has to be trained as well, but this could either happen during training or The graphical overview is again given in Figure 1.

3 Dataset creation

There exist two principal methods to collect a dataset for Offline Reinforcement Learning, when using an agent as behavioral policy training in an Online Reinforcement Learning fashion. The easiest possibility with the least computational overhead is to use the transitions the online agent encounters during learning. Due to that, effectively as many policies as there where training steps populate this replay buffer. The second possibility is to use either the final or one or more intermediate policies that act on a different version of the same environment to create the transitions for the offline dataset. There are suggestions [Gauci et al., 2019] that the second method seems to be the more reasonable if one wants to incorporate real world transitions where the behavioral policy is usually fixed and changes only slightly over time (e.g. a human car driver). A visual depiction of the two dataset creation methods is given in Figure 2.

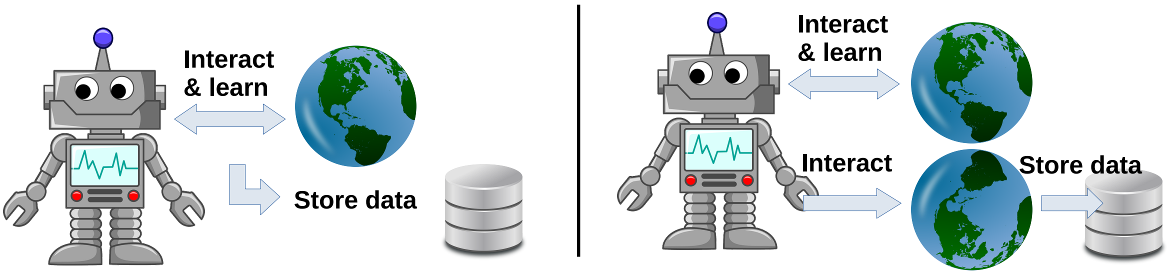

Figure 2: Different dataset creation methods to create Offline Reinforcement Learning datasets from an online Agent as behavioral policy. Using full experience replay buffer (left) and generate the dataset from new interactions from one of more of the behavioral policies obtained from online training (right).

4 Offline Reinforcement Learning on Atari Games

As initially stated, this work tries to verify the hypothesis that the different performance in [Agarwal et al., 2020b] and [Fujimoto et al., 2019a] stems from the different dataset generation strategies. While [Agarwal et al., 2020b] did an extensive study on 60 different Atari games, [Fujimoto et al., 2019a] focused on only nine of them. As the computational power is quite prohibitive for this large state space domain where several million transitions / policy updates are necessary to obtain a good policy, a further reduction to just one Atari game, "Breakout" that was very illustrative in both studies is done. Of course, a full verification of the assumption must be done on at least all of the nine games tackled by both studies, but this serves as a starting point.

It is worth to mention that both studies followed the suggestions of [Machado et al., 2017] for comparable results on the Atari environment, which I will also closely follow. For details see Appendix A.1. Training is measured in environment steps, where every fourth step a gradient update is performed. For details on hyperparameters, see Appendix A.2. Evaluation of the offline agent is done via several online interaction episodes, again holding close the the proposals of the two studies.

4.1 Standard off-policy Reinforcement Learning Algorithms

The easiest experiment in the offline Reinforcement Learning setting is to use standard off-policy algorithms that work well in the online setting. [Agarwal et al., 2020b] reported very promising results for this case, but [Fujimoto et al., 2019a] got very poor results where only BCQ that is at its core more robust imitation learning succeeds. Their results on the Atari game 'Breakout' are depicted in Figure 3.
I conducted the same experiments sticking as close as possible to the original implementations. Noteworthy changes are the limitation of training episodes and training samples from 50 million ([Agarwal et al., 2020b]) to 10 million for both that where also used in [Fujimoto et al., 2019a] and using Adam [Kingma and Ba, 2017] as optimizer for the behavioral policy instead of RMSProp [Hinton et al., 2014] as done in [Agarwal et al., 2020b]. Ablation studies, executed in [Agarwal et al., 2020b] justify the limitation of training samples for this specific game without substantial performance decrease.

When comparing to my reimplementations given in Figure 4, one can see that while my behavioral policies performs worse than those in both literature experiments, the principal that using the full experience replay yields better offline agents seems to hold. As there are many slight differences between [Agarwal et al., 2020b] and [Fujimoto et al., 2019a] and my implementations kept close to the latter, the results seem reasonable. Both DQN as REM perform comparable to the literature implementations, but QR-DQN performs significantly worse in both

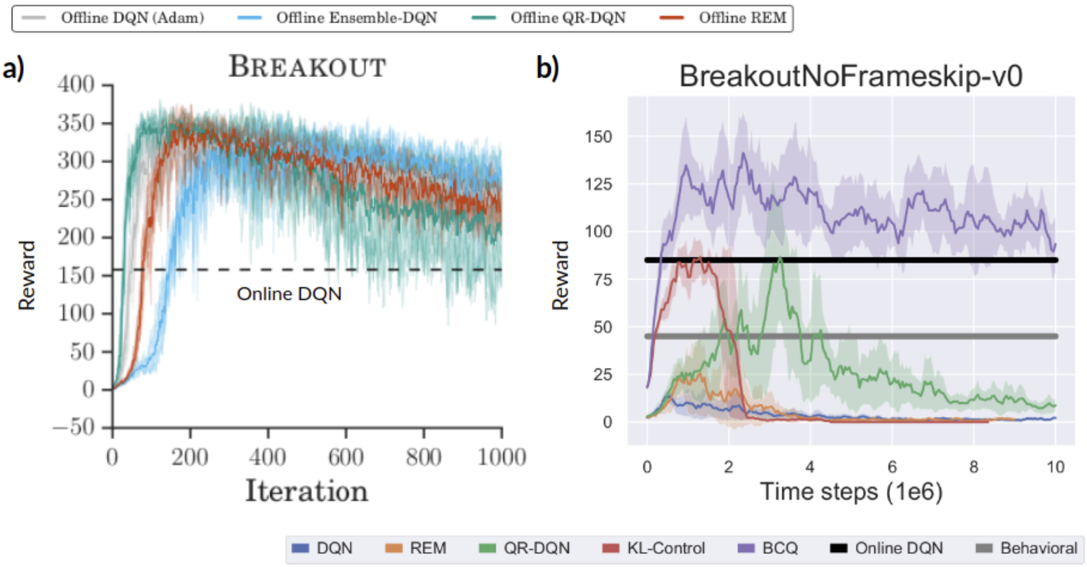

Figure 3: Results on 'Breakout', taken from **a)** [Agarwal et al., 2020b] and **b)** [Fujimoto et al., 2019a].

settings which might be due to slight differences in the implementation. What is really remarkable is the failure of BCQ in the setting of [Fujimoto et al., 2019a] which it is actually proposed for. As the algorithm is performing as expected (very comparable to the behavioral policy) if given the full experience replay buffer it is unlikely an implementation issue. On the other hand, the remaining algorithms seem to be able to learn when trained on the dataset collected from a single policy, therefore it is unlikely that an implementation error in the dataset creation routine is causing the weak performance of BCQ.

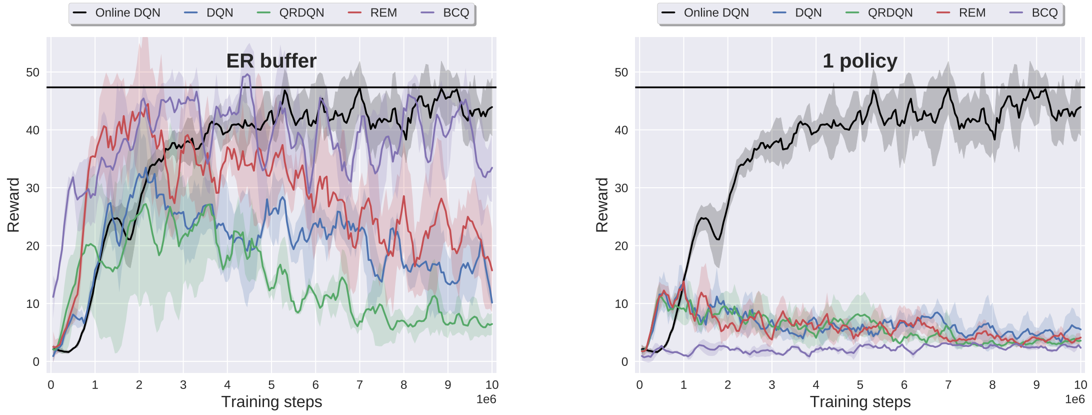

Figure 4: Performance on the two dataset creation techniques, shown for all implemented algorithms. To evaluate the reward, the policy was tested every 50 000 steps for ten episodes in the online environment, resulting in one datapoint that is the mean over the ten episodes. Results above are further smoothed with a running average over the last 5 datapoints, following suggestions by [Agarwal et al., 2020b] and [Fujimoto et al., 2019a]. Every algorithm was executed 3 times for different seeds, the light bars indicating the distance of one standard deviation from the mean across those runs.

4.2 Ablation study

Section 3 introduced the two principle two options to create the offline datasets that are used in [Agarwal et al., 2020b] and [Fujimoto et al., 2019a]. A key question here is, if there is something special about the transitions in the experience replay buffer of whether it is just the huge number of generating policies that are crucial to succeed in the offline setting. Figuratively speaking, the question is whether seeing transitions that helped the behavioral policy to learn something helps more than seeing the behavioral policies actions given its current knowledge.

To that end, an ablation study where the number of generating policies is varied was executed. The policies used for dataset creation are evenly spaced, if two policies would have been used it would be the policies after 5 million training steps and after all 10 million training steps, both generating 5 million transitions each. For details on how the policies were used to create transitions, see Appendix A.2 or [Fujimoto et al., 2019a]. The most interesting case is, what happens when 2.5 million different policies were used, because with training steps every four environment interactions that is identical to how many policies created the Experience Replay Buffer dataset.

As one can see from Figure 5, there is hardly any difference between using 250, 25 thousand or 2.5 million policies to create the dataset for all of the algorithms. Using only the final policy to create the dataset lead to the worst performance basically across all algorithms. The Experience Replay dataset was especially beneficial for REM and BCQ, but QR-DQN also had runs with very good performance on that dataset. Only DQN is quite comparable among all dataset creation strategies except if using only one policy. One notable fact is, that variances are way higher when using the Experience Replay dataset, as runs themself had huge performance variances compared to when using the other datasets. Finally, as already discussed in the previous section, BCQ did only work for the Experience Replay dataset and did not work at all for all other datasets.

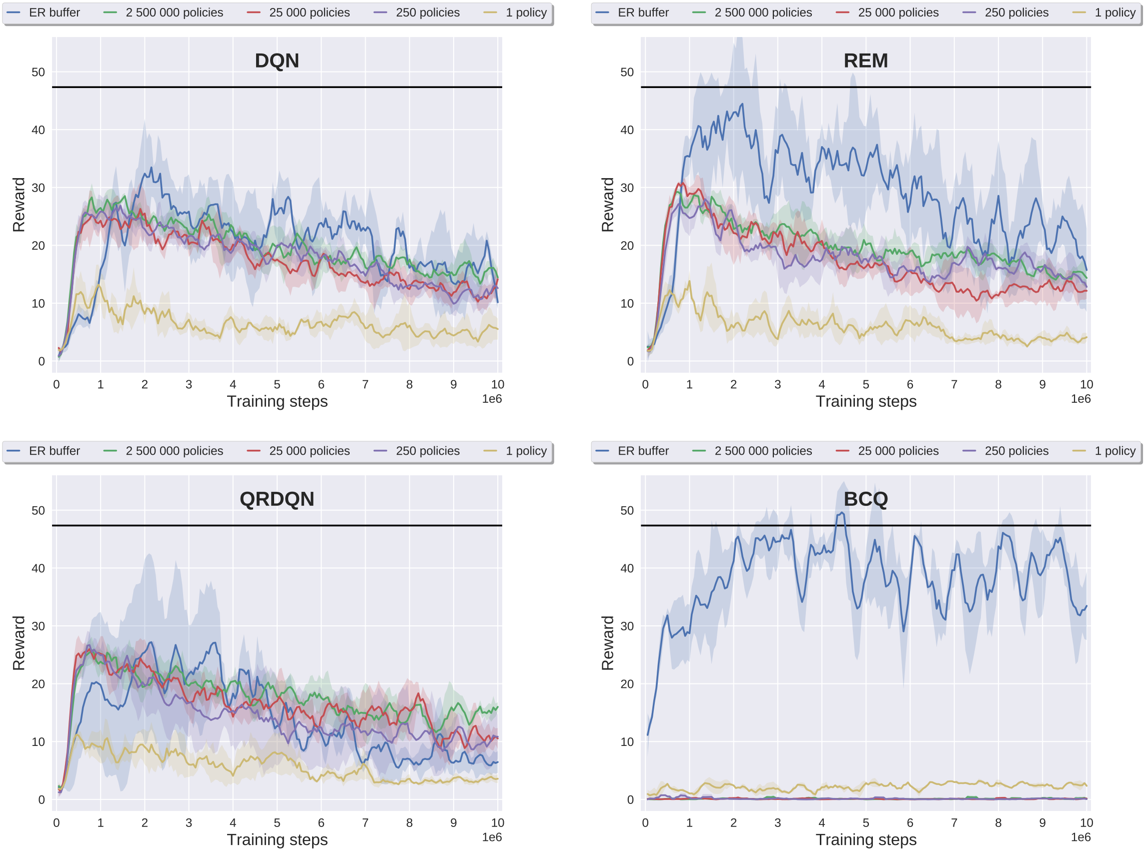

Figure 5: Performance of the algorithms on the different dataset creation conditions. To evaluate the reward, the policy was tested every 50 000 steps for ten episodes in the online environment, resulting in one datapoint that is the mean over the ten episodes. Results above are further smoothed with a running average over the last 5 datapoints, following suggestions by [Agarwal et al., 2020b] and [Fujimoto et al., 2019a]. Every algorithm was executed 3 times for different seeds, the light bars indicating the distance of one standard deviation from the mean across those runs.

4.3 Value divergence

[Fujimoto et al., 2019a] reported divergent value estimates throughout training, especially pronounced for DQN and REM, QR-DQN starting to diverge toward the end of training and BCQ having a stable value estimate which is its purpose. I did not encounter value divergences in any of my settings, which might be due to some implementation details. That is actually remarkable as the divergent values have been found to be one of the key problems in Offline Reinforcement Learning as they are due to the extrapolation error mentioned earlier. Figure 6 shows the results from [Fujimoto et al., 2019a] and my experiment meeting his dataset creation setting of using only one behavioral policy. For value estimates in all other settings see Appendix A.3.

It is interesting that DQN, REM and QR-DQN always converge towards the final value estimate of the behavioral policy, but BCQ is a way more conservative estimation. This is most likely due to the restriction of eligible actions in the value update that neglects possibly overestimated values that were not encountered by the behavioral policy.

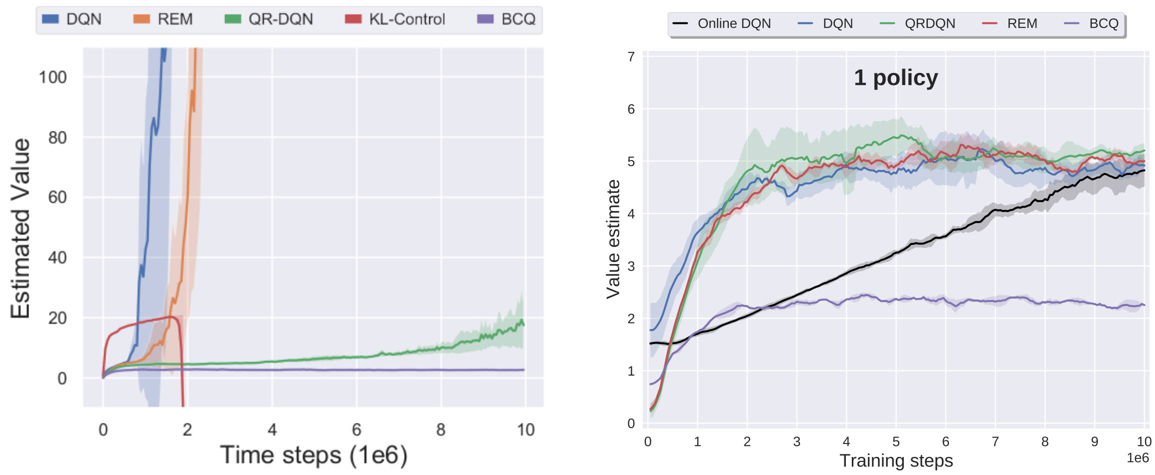

Figure 6: [Fujimoto et al., 2019a] reported diverging value estimates for DQN and REM (left image), which I can not confirm as I got convergent value estimates for all algorithms even for BCQ that exhibited very poor performance on the dataset condition matching the one from [Fujimoto et al., 2019a].

5 Dataset assessment

One can see from section 4.2, that the different dataset generation strategies have a huge impact on performance. A difference can of course be a different coverage of the state space, which is something hard to measure. While there is no canonical way that fits every environment to estimate how much of the state space is already covered by the observed transitions, I follow the line of [Seo et al., 2021] and [Kusari, 2020] that both try to do coverage estimation to propose interesting regions to visit during usually random exploration steps of online reinforcement learning agents. Further, the datasets can be characterized by their reward and episode length structure.

5.1 k-Nearest Neighbour Entropy Estimate through Random Encoders

[Seo et al., 2021] use a Random Encoder Network to project the state to a lower dimensional space, which is basically a random initialization of the function approximator used for estimating the Q-value, just with 2 instead of the number of actions as output. They further introduce a k-nearest neighbor entropy estimator $\mathcal{H}_N^k(\boldsymbol{X})$ over the dataset $\boldsymbol{X}$, given by

$$\mathcal{H}_N^k(\boldsymbol{X}) \approx \frac{1}{N}\sum_{i=1}^{N} \log ||\boldsymbol{x}_i - \boldsymbol{x}_i^{\text{k-NN}}||_2 \tag{5.1}$$

where we sample N different states $\boldsymbol{x}_i$ from the full dataset and compare them to their k-Nearest Neighbours. For this report, N = 10 000 and k = 10 was used. This entropy estimate reflects how different the states are, so higher means more coverage of the state space.

5.2 Approximate Pseudo-Coverage

[Kusari, 2020] takes a different route, with an idea they call Approximate Pseudo-Coverage (APC). They use a non-linear embedding that aims to preserve the local neighbourhood of an high dimensional state-space, namely t-SNE [van der Maaten and Hinton, 2008] to project the dataset or a sample of it into 2D space. Then the covered space is assessed by an over-approximation using a evenly spaced grid, where the percentage of non-empty cells is a proxy for the coverage of the state space.

Due to performance reasons when working on the high dimensional state space of an Atari game, I resorted to using UMAP [McInnes et al., 2020], which is computationally cheaper and even provides a meaningful hyperparameter that tempers how much of local and global structure of the data is preserved. Note that [Kusari, 2020] verfied their hypothesis only on very

	1 Policy	250 Policies	25 000 Policies	2 500 000 Policies	Experience Replay Buffer
$N \cdot \mathcal{H}_N^k$	0.0143 ± 0.0001	0.0127 ± 0.0001	0.0123 ± 0.0001	0.0123 ± 0.0001	0.0137 ± 0.0001
APC (Random Encoder)	0.3538 ± 0.0007	0.2995 ± 0.0034	0.2905 ± 0.0011	0.2902 ± 0.0024	0.3363 ± 0.0024
APC (UMAP)	0.0542 ± 0.0148	0.0594 ± 0.0011	0.0588 ± 0.0079	0.0635 ± 0.0197	0.1016 ± 0.0223
Mean Episode Length	1097.5	688.1	672.9	679.9	859.1
Mean Reward	32.9	17.5	16.9	17.1	24.1

Table 1

low-dimensional state spaces, e.g. the classic control problem CartPole-v1 implemented in gym [Brockman et al., 2016]. The the resulting structures are very sparse, I also tried Approximate Pseudo-Coverage on the downprojection of the Random Encoder, which qualitatively gave the same results as when using UMAP, but with less pronounced differences between the datasets and a high base level. Further I tried using the simple linear downprojection method Primary Component Analysis (PCA), which I observed to have very high variance in the estimates across different samples. Additionally, the projection looks rather poor from a human point of view. The visual representations of the UMAP mapping and Random Encoder mapping are to be found in the Appendix A.4

Both estimates are by far not optimal as they both give no real information about how much of the actually possible state space is covered, but they can give information on how diverse the datasets are.

5.3 Reward and Episode Length

Further, one can ask how different the datasets are w.r.t. the received rewards and the episode lengths. Note that no negative reward is given upon life-loss and the reward is clipped to [-1,1] as discussed in Appendix A.1. Therefore the total reward is the number of bricks destroyed within the dataset, which is highly correlated with the episode length for 'Breakout' . All results are to be found in Table 1. Further, the dependence of episode length and reward on the dataset creation technique as well as histograms over episode lengths are depicted in Figure 7. Striking is the lack of suboptimal episodes when using only the final policy, which indicates that the ϵ of 0.2 when generating a high noise episode is too low and is not sufficient to obtain very short sequences.

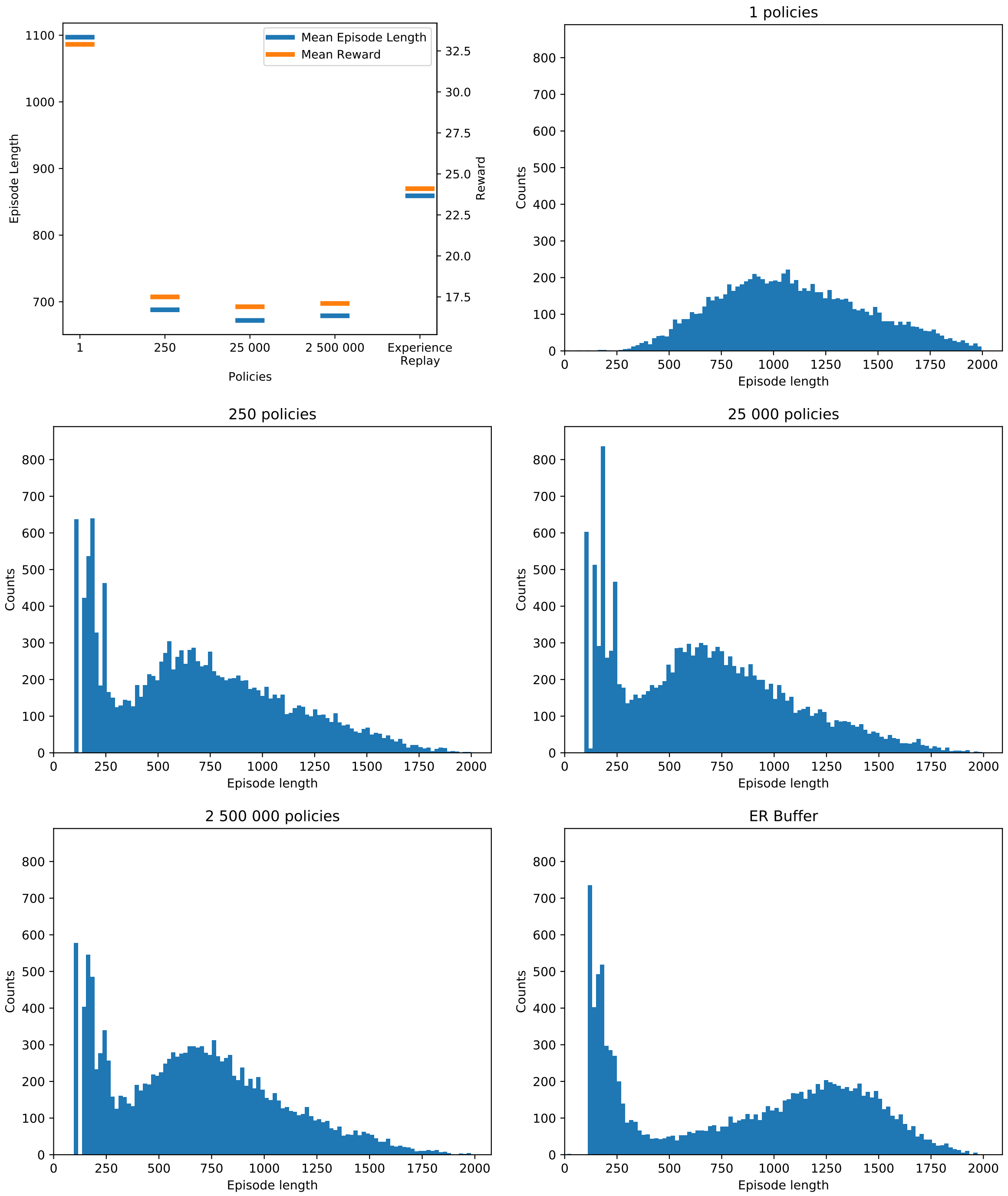

Figure 7: First image illustrates the Mean Episode Length and Mean Reward that are tabulated in Table 1, the other Figures show Histograms for the Episode Lengths for the different dataset creation strategies.

6 Future Work

Working on the Atari domain is quite interesting as the tasks are challenging to solve, but a lot of engineering effort has to be taken to work on it. Further, training times are several hours to days, which is quite prohibitive for rapid prototyping and new ideas. Therefore, follow-up aims for smaller toy problems, where the problems of Offline Reinforcement Learning may be studies with less engineering overhead. REM and BCQ are two promising algorithms that are good baselines, together with distributional RL where QR-DQN is implemented with relative ease although it did not outperform the other algorithms on this specific environment. One additional algorithm that aims for more conservative value estimates without explicitly restricting the policy, Conservative Q-Learning (CQL) [Kumar et al., 2020] seems promising in the offline domain, as well as many other that were introduced during execution of this practical work, Conservative Offline Model-Based Policy Optimization (COMBO) [Yu et al., 2021] and You Only Evaluate Once (YOEO) [Wonjoon and Niekum, 2020] to name a few.

It was shown in this work that there seems to be something beneficial other than sheer state coverage when using the experience replay buffer. This must be further quantified, e.g. by using only a high / low reward subset or only the first / second half of the full dataset. Another interesting direction is to use a dataset curriculum for offline training, where the agent gets random data first and after a number of training steps the dataset is switched to a high reward dataset. This seems to be more like online training, where the sample distribution changes over time to be more focused on high reward tasks, whereas offline training currently always keeps the same distribution over states throughout all training steps. Finally, it is noteworthy to state the the latest NeurIPS Offline Reinforcement Learning Workshop [Agarwal et al., 2020a] brought a lot of new ideas into this sub-field that are noteworthy to check and include in further experiments on the topic. Experiments are looking a lot into the extrapolation error introduced from Equation 2.2, which is beneficial to the fundamental understanding of action-value estimation with non-linear function approximators we are using nowadays, but whose dynamics are poorly understood. Work in the offline domain therefore has high potential to further improve algorithms also in the online paradigm.

7 Conclusion

This work has shown that the different dataset creation techniques used in [Agarwal et al., 2020b] and [Fujimoto et al., 2019a] are an important difference if one wants to succeed in Offline Reinforcement Learning. The ablation study suggests, that the sheer number of policies alone is not the reason why the Experience Replay dataset is working way better in the offline setting. Further, having multiple policies seems not to be very beneficial w.r.t state coverage within the dataset, where actually the settings used in the two papers cover the state space more thoroughly than my experiments of using an intermediate number of policies. A final interesting finding is, that there seems to be implementation details that counteract value overestimation, although the problem is not explicitly tackled in my work other than for the algorithm BCQ where possible value updates are restricted.

References

[Agarwal et al., 2020a] Agarwal, R., Kumar, A., Tucker, G., Precup, D., and Li, L. (2020a). Offline reinforcement learning workshop @ neurips. https://offline-rl-neurips.github.io/index.html.

[Agarwal et al., 2020b] Agarwal, R., Schuurmans, D., and Norouzi, M. (2020b). An optimistic perspective on offline reinforcement learning. arXiv, 1907.04543.

[Bellman et al., 1957] Bellman, R., Bellman, R., and Corporation, R. (1957). *Dynamic Programming*. Rand Corporation research study. Princeton University Press.

[Bousmalis et al., 2017] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V. (2017). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. 1709.07857.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. 1606.01540.

[Dabney et al., 2017] Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017). Distributional reinforcement learning with quantile regression. arXiv, 1710.10044.

[Fujimoto et al., 2019a] Fujimoto, S., Conti, E., Ghavamzadeh, M., and Pineau, J. (2019a). Benchmarking batch deep reinforcement learning algorithms. arXiv, 1910.01708.

[Fujimoto et al., 2019b] Fujimoto, S., Meger, D., and Precup, D. (2019b). Off-policy deep reinforcement learning without exploration. arXiv, 1812.02900.

[Gauci et al., 2019] Gauci, J., Conti, E., Liang, Y., Virochsiri, K., He, Y., Kaden, Z., Narayanan, V., Ye, X., Chen, Z., and Fujimoto, S. (2019). Horizon: Facebook's open source applied reinforcement learning platform. 1811.00260.

[Hessel et al., 2017] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. 1710.02298.

[Hinton et al., 2014] Hinton, G., Srivastava, N., and Swersky, K. (2014). Coursera course; neural networks for machine learning. http://www.cs.toronto.edu/ hinton/coursera/lecture6/lec6.pdf.

[Huber, 1964] Huber, P. J. (1964). Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization. arXiv, 1412.6980.

[Kumar et al., 2020] Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. 2006.04779.

[Kusari, 2020] Kusari, A. (2020). Assessing and accelerating coverage in deep reinforcement learning. 2012.00724.

[Machado et al., 2017] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2017). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. arXiv, 1709.06009.

[McInnes et al., 2020] McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction. 1802.03426.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv, 1312.5602.

[Puterman, 1994] Puterman, M. (1994). *Examples*, chapter 3, pages 33–73. John Wiley Sons, Ltd, https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470316887.ch3.

[Seo et al., 2021] Seo, Y., Chen, L., Shin, J., Lee, H., Abbeel, P., and Lee, K. (2021). State entropy maximization with random encoders for efficient exploration. 2102.09430.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[van der Maaten and Hinton, 2008] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

[van Hasselt et al., 2015] van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. 1509.06461.

[Watkins and Dayan, 1992] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

[Wonjoon and Niekum, 2020] Wonjoon, G. and Niekum, S. (2020). You only evaluate once - a simple baseline algorithm for offline rl. https://offline-rl-neurips.github.io/program/offrl$_60.html.$

[Yu et al., 2021] Yu, T., Kumar, A., Rafailov, R., Rajeswaran, A., Levine, S., and Finn, C. (2021). Combo: Conservative offline model-based policy optimization. 2102.08363.

A Appendix

A.1 Atari preprocessing

Preprocessing of the Atari environment follow previous work, most notably following the suggestions of [Machado et al., 2017] for reproducible results. Following usual terminology, raw image outputs of the atari environment are denoted as *frames*, which represent colored images showing the current game state as humans perceive it. These frames are grayscaled and resized to 84×84 pixels. The agent receives a state only every 4th frame given by the environement and repeats this frame four times, as frames per second are very high (60 fps) in this games. The state is defined as the maximum between two consecutive frames, as the Atari engineers did some tricks of only partially outputting content to increase the framerate, but results in visual artifacts if one looks at a single frame. Further, the input to the network consists of the four previous frames, just using zeros if previous frames do not exist. Therefore the input tensor to the network is of dimenions (4, 84, 84).

As the Atari environment is deterministic, sticky actions with a probability of $p = 0.25$ are used. This sets the current action a_t to the last action a_{t-1} with probability p. Both studies refrained from using random frame skips and random no-op's at the beginning of an episode.

Reward clipping to [-1,1] was utilized and the game is over when it terminates rather then when a life is lost. Also, episodes are terminated on more than 27k timesteps, which is 30 minutes of realtime gameplay. This is not an issue for "Breakout" though, as doing nothing will not stale the state.

A.2 Hyperparameters and Experimental Details

The basic architecture is equivalent for all algorithms, only the linear heads on top of the Convolutional Neural Network as well as the training procedure and losses differ according to the formulas given in the respective chapters. The network architecture consists of three convolutional layers that take in input tensors of shape (4,84,84) as discussed in the Atari preprocessing Appendix A.1. The first convolution has a kernel size of 8×8 with a stride of 4, outputting 32 channels. The second convolution has a kernel size of 4×4 with a stride of 2, outputting 64 channels. The final convolution has a kernel of size 3×3 unstrided, also outputting 64 channels. The linear layer that follows therefore transforms $7 \times 7 \times 64 = 3136$ dimensions to 512 dimensions. A final linear layer that transforms to the number of actions (REM: number of actions $\times$ heads) (QR-DQN: number of actions $\times$ quantiles) is used. BCQ has another linear layer that takes inputs from the penultimate layer and transforms it to the number of actions with a softmax activation, used for assessing which action is the most probable according to the behavioral policy in the given state. ReLU activation functions are used after every layer except at the last one of course.

Hyperparameters are held consistent between algorithms if possible and where chosen to match the suggestions of [Hessel et al., 2017]. A tabular listing is given in Table 2.

Hyperparameter	Value
Batch size	32
Learning rate	0.0000625
Discount γ	0.99
Huber loss λ	1
Optimizer	Adam [Kingma and Ba, 2017]
Adam ϵ	0.00015
Evaluation ϵ	0.001
Target network update frequency	8000 training iterations

Table 2: Hyperparameters used by the agents

Further hyperparameters are used for the respective algorithms. QR-DQN used 50 quantiles, REM used 200 heads and BCQ a treshold of 0.3 to select actions. Additionally, the generative action model in BCQ is regularized by a penalty on the final pre-activation output x by $0.01x^2$. Training the offline algorithms is really straightforward, every 50 000 updates the policy is evaluated with the online environment and the score is averaged over 10 runs every time. The plots then report the moving average over the last 5 evaluations.

Additional hyperparameters for training the online agent (DQN) that serves as behavioral policy are required. They are listed in Table 3. Training frequency means how often the policy is updated w.r.t. the steps in the environment. Warmup timesteps are the environment steps a random policy is deployed until the first update on the policy is performed. An ϵ-greedy policy is used that gets decayed during training.

Hyperparameter	Value
Replay buffer size	1 000 000
Training frequency	Every 4th timestep
Warmup timesteps	20 000
Initial ϵ	1.0
Final ϵ	0.01
ϵ decay period	250 000 training steps

Table 3: Hyperparameters to train DQN online

The dataset strategy of [Fujimoto et al., 2019a] where a single (or several in this study) policies are used to create the trajectories, with a probability $p = 0.2$ the offline evaluation ϵ is used (see Table 2). In all other cases, $\epsilon = 0.2$ is used to ensure sufficient exploration of the state space.

A.3 Offline Reward and Value Approximation

Figure 8 shows the rewards for training all algorithms on the different offline datasets. Figure 9 shows the value estimate throughout training for the different offline datasets.

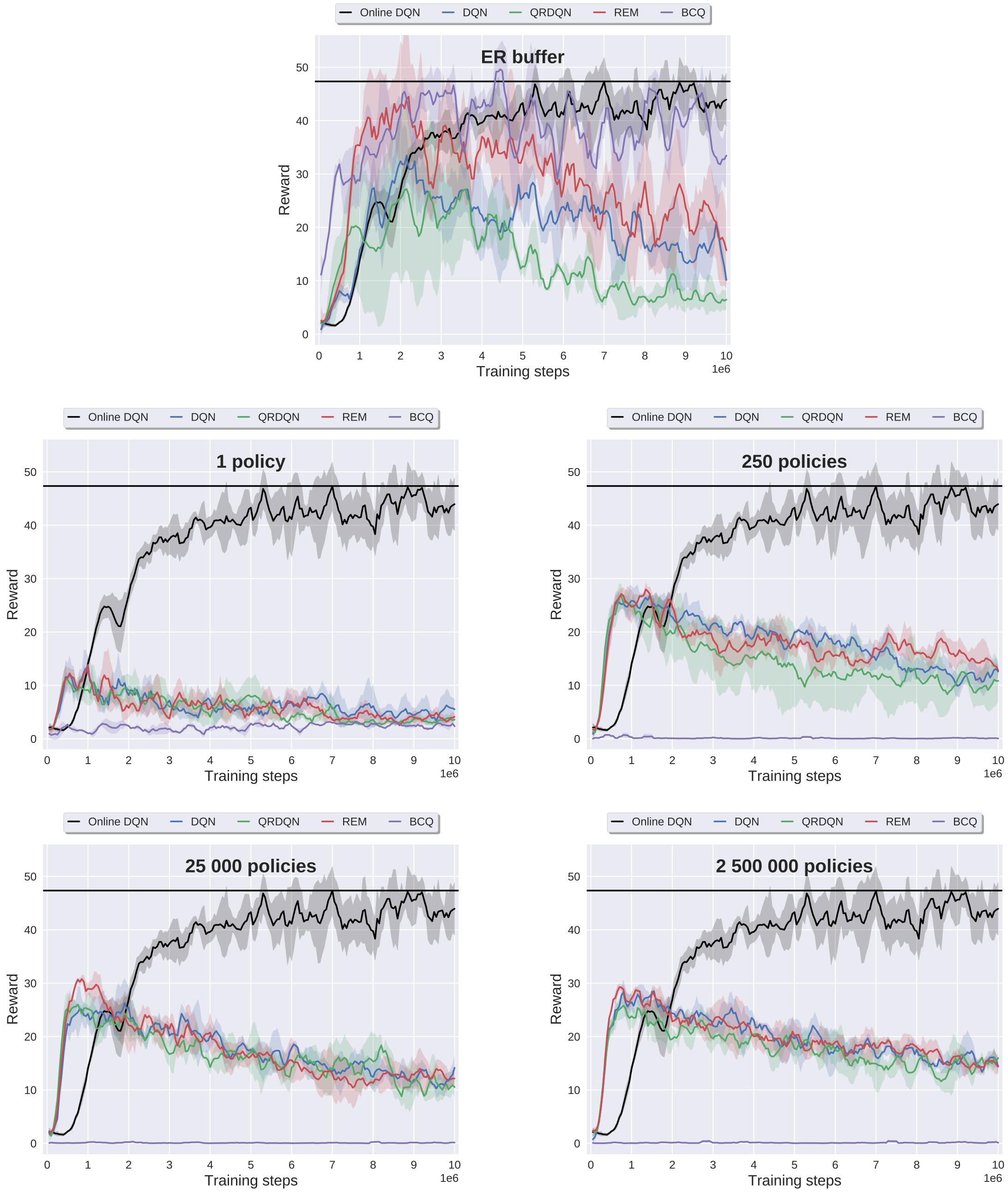

Figure 8: Reward in the offline setting. After 50 000 steps, the policy was tested for ten episodes in the online environment, resulting in one datapoint that is the mean over the ten episodes. Results above are further smoothed with a running average over the last 5 datapoints, following suggestions by [Agarwal et al., 2020b] and [Fujimoto et al., 2019a]. Every algorithm was executed 3 times for different seeds, the light bars indicating the distance of one standard deviation from the mean across those runs.

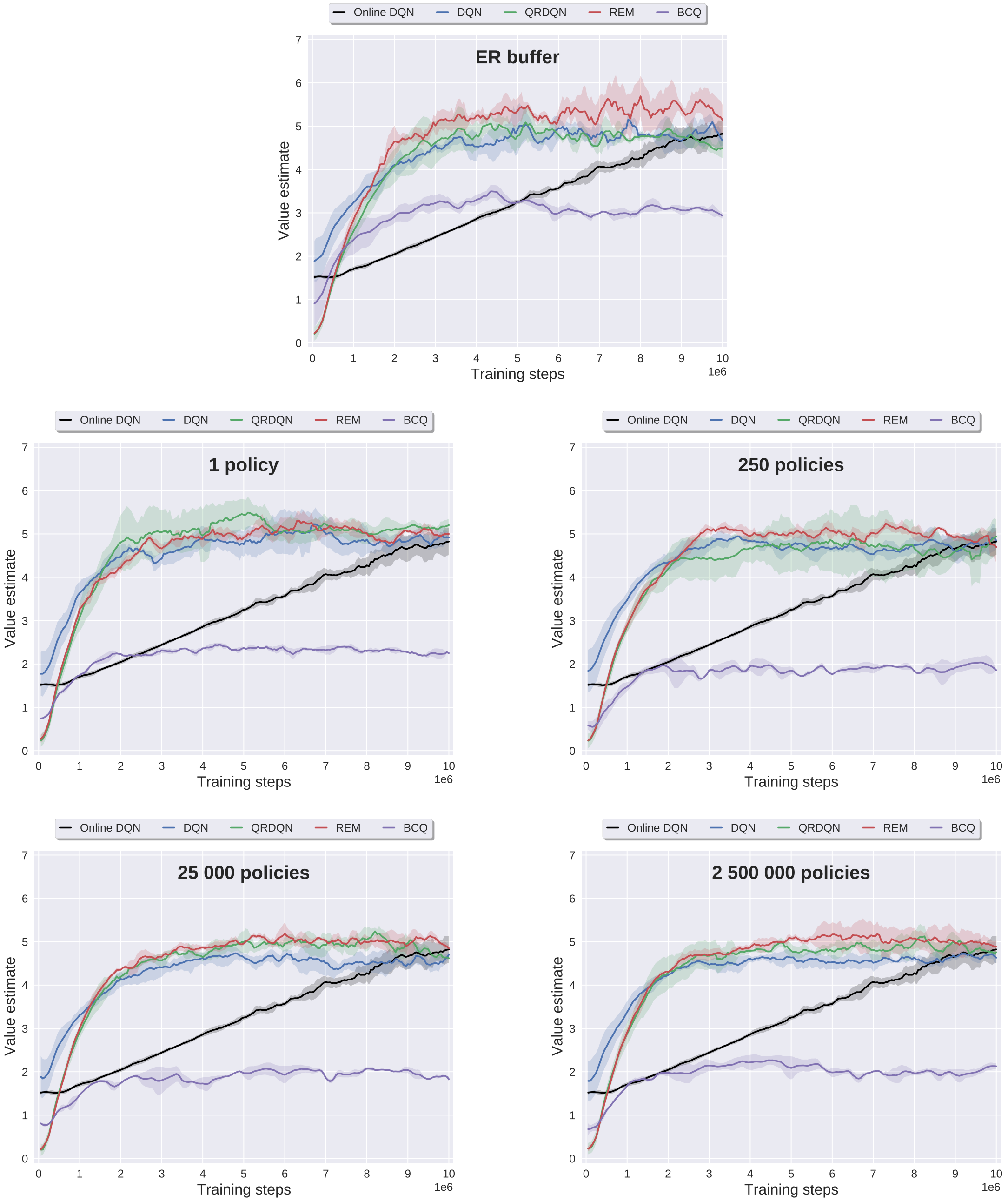

Figure 9: Approximation of the maximum action-value taken in the offline setting. After 50 000 steps, the policy was tested for ten episodes in the online environment, resulting in one datapoint that is the mean over the ten episodes. Results above are further smoothed with a running average over the last 5 datapoints, following suggestions by [Agarwal et al., 2020b] and [Fujimoto et al., 2019a]. Every algorithm was executed 3 times for different seeds, the light bars indicating the distance of one standard deviation from the mean across those runs.

A.4 Projection visualisations

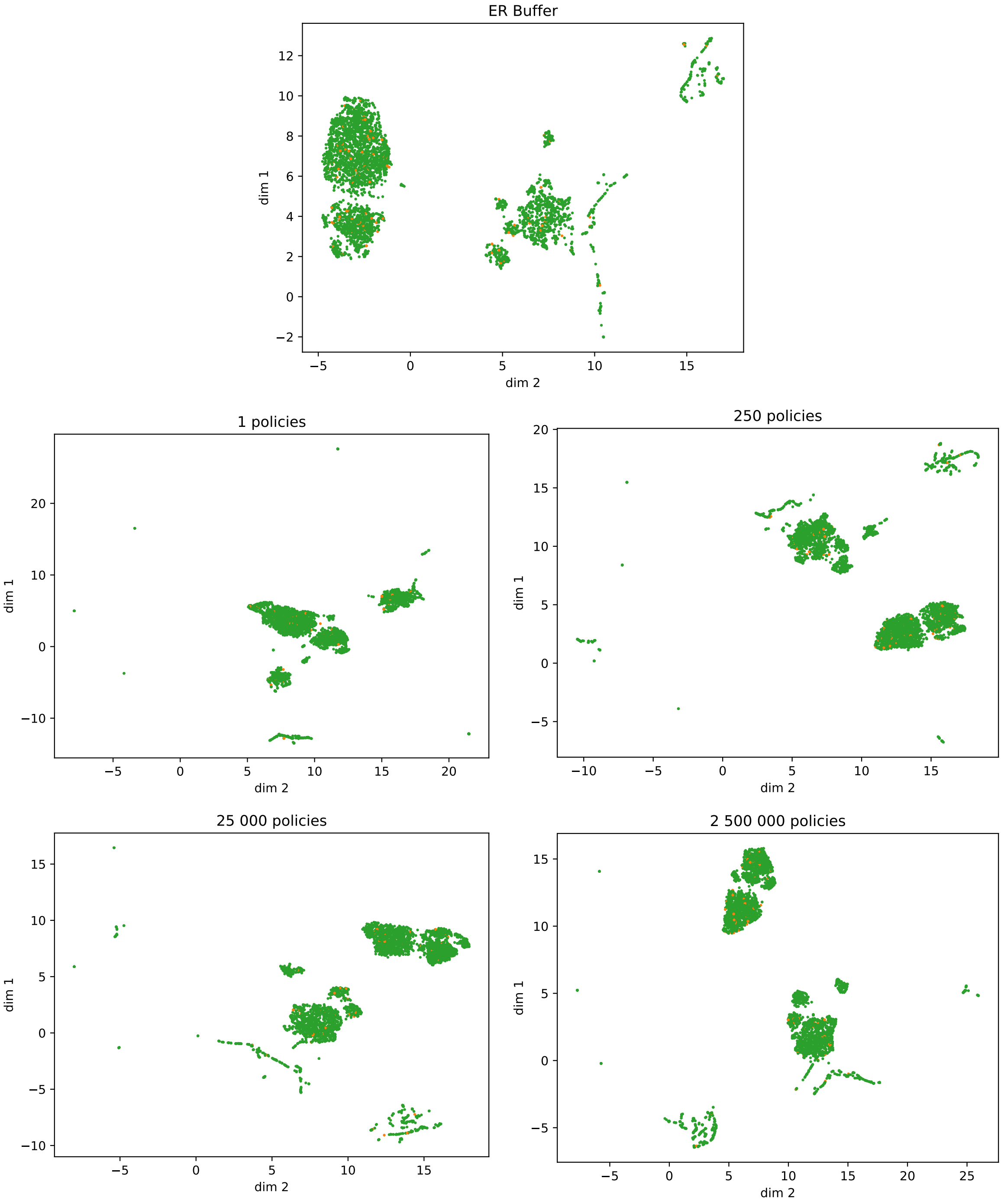

Figure 10: UMAP projections of the dataset samples. Green dots are samples that gave zero reward, orange dots are samples that gave a reward of one.

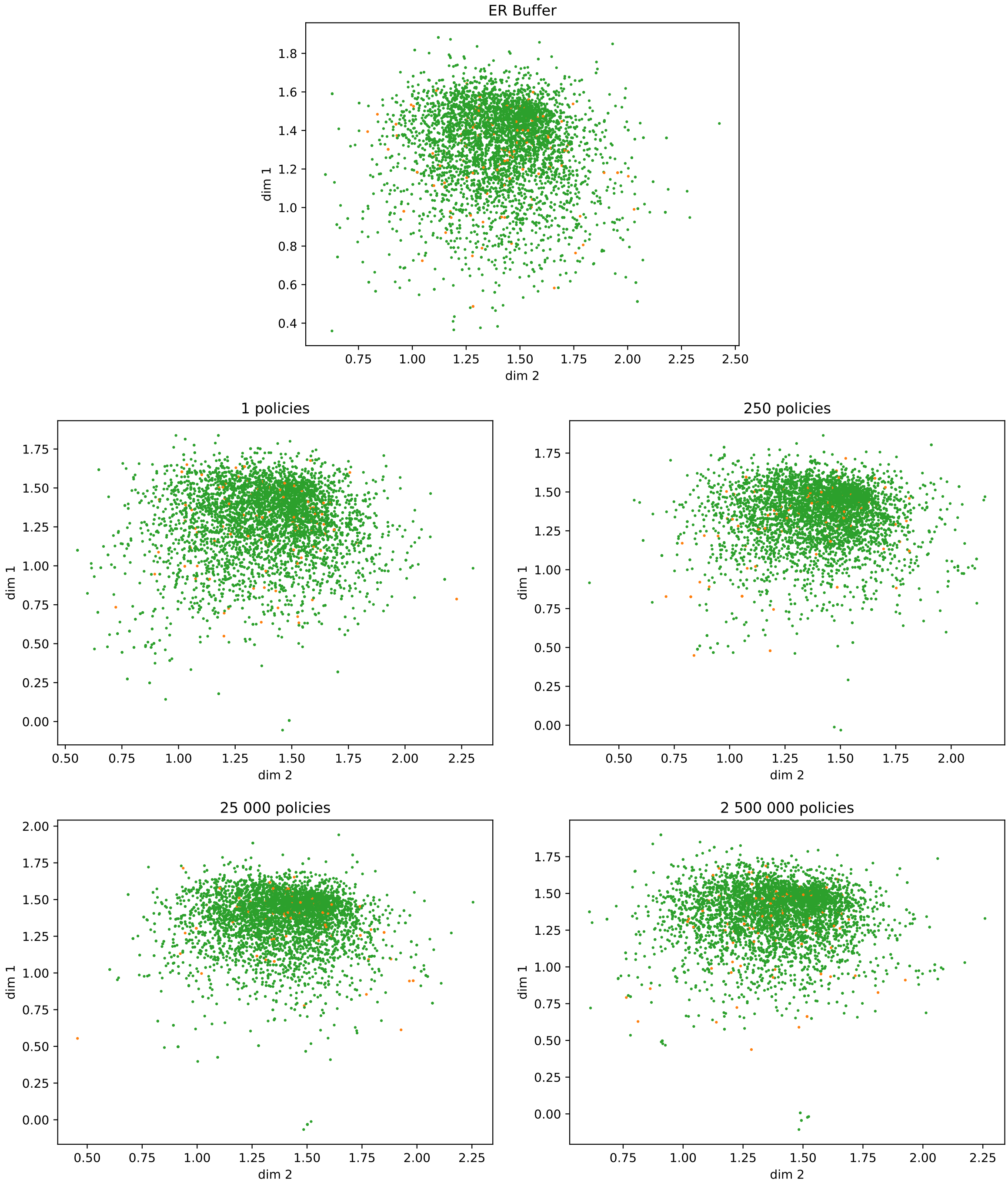

Figure 11: Random Encoder projections of the dataset samples. Green dots are samples that gave zero reward, orange dots are samples that gave a reward of one.