Benchmarking Batch Deep Reinforcement Learning Algorithms

Scott Fujimoto[1,2], **Edoardo Conti**[2], **Mohammad Ghavamzadeh**[3], **Joelle Pineau**[1,3]
[1]Mila, McGill University
[2]Facebook
[3]Facebook AI Research
scott.fujimoto@mail.mcgill.ca

Abstract

Widely-used deep reinforcement learning algorithms have been shown to fail in the batch setting–learning from a fixed data set without interaction with the environment. Following this result, there have been several papers showing reasonable performances under a variety of environments and batch settings. In this paper, we benchmark the performance of recent off-policy and batch reinforcement learning algorithms under unified settings on the Atari domain, with data generated by a single partially-trained behavioral policy. We find that under these conditions, many of these algorithms underperform DQN trained online with the same amount of data, as well as the partially-trained behavioral policy. To introduce a strong baseline, we adapt the Batch-Constrained Q-learning algorithm to a discrete-action setting, and show it outperforms all existing algorithms at this task.

1 Introduction

Batch reinforcement learning is the study of algorithms that can learn from a single batch of data, without directly interacting with the environment [Lange et al., 2012]. Learning with finite data sets is invaluable for a variety of real-world applications, where data collection may be difficult, time-consuming or costly [Guez et al., 2008, Pietquin et al., 2011, Gauci et al., 2018]. In principle, standard off-policy deep reinforcement learning algorithms such as DQN and DDPG [Mnih et al., 2015, Lillicrap et al., 2015] are applicable in the batch reinforcement learning setting, due to basis on more fundamental batch reinforcement learning algorithms such as Fitted Q-iteration [Ernst et al., 2005, Riedmiller, 2005]. However, these traditional algorithms only come with convergence guarantees for non-parametric function approximation [Gordon, 1995, Ormoneit and Sen, 2002], have no guarantees on the quality of the learned policy, and scale poorly to high dimensional tasks.

Recent results demonstrated widely-used off-policy deep reinforcement learning algorithms fail in the batch setting due to a phenomenon known as extrapolation error, which is induced from evaluating state-action pairs which are not contained in the provided batch of data [Fujimoto et al., 2019]. This erroneous extrapolation is propagated through temporal difference update of most off-policy algorithms [Sutton, 1988], causing extreme overestimation and poor performance [Thrun and Schwartz, 1993]. Fujimoto et al. [2019] proposed the batch-constrained reinforcement learning framework, where the agent should favor a state-action visitation similar to some subset of the provided batch, and provided a practical continuous control deep reinforcement learning algorithm, BCQ, which eliminates unseen actions through a sampling procedure over a generative model of the data set. Contrary to these results, Agarwal et al. [2019] showed that using the entire history of a deep reinforcement learning agent as a batch (50 million time steps), standard deep reinforcement learning algorithms could reach a comparable performance to an online algorithm. In particular, they

Preprint. Under review.

highlighted that distributional reinforcement learning algorithms [Bellemare et al., 2017, Dabney et al., 2017] performed particularly well with this large and diverse data set.

Given batch reinforcement learning encompasses a large number of settings, existing algorithms have been tested over a wide range of environments, and under a variety of data distributions, making comparisons difficult and contradictory results possible. In this paper, we benchmark the performance of several algorithms on the Arcade Learning Environment [Bellemare et al., 2013], with a large data set of 10 million data points generated by a partially-trained policy. We find that with a single behavioral policy, not only do widely-used off-policy deep reinforcement learning algorithms perform poorly, even existing batch algorithms are inadequate solutions, failing to outperform the behavioral policy.

We introduce a variant of the BCQ algorithm [Fujimoto et al., 2019] which operates on a discrete action space. Our version of BCQ is simple to implement, while maintaining the core ideas of the original continuous control algorithm. Furthermore, our results demonstrate BCQ greatly outperforms all prior deep batch reinforcement learning algorithms, including KL-Control [Jaques et al., 2019], which was shown to outperform another naïve variant of BCQ for discrete actions. BCQ demonstrates learning akin to a strong robust imitation learning algorithm, matching, or exceeding, the performance of the noiseless behavioral policy, a DQN agent trained online with the same amount of data. While simply matching the noiseless behavioral policy is often unsatisfactory, we hope that BCQ will serve as a strong baseline in this setting.

Our contributions are as follows:

- We benchmark the performance of several batch deep reinforcement learning algorithms under a single unified setting. This continues the line of work from Agarwal et al. [2019] by examining the performance of widely-used off-policy algorithms in the Atari domain. However under ordinary data conditions, we find that standard off-policy reinforcement learning algorithms perform poorly.
- We validate the batch reinforcement learning experiments from Fujimoto et al. [2019] on the more challenging, discrete-action Atari environments, and demonstrate the phenomenon of extrapolation error still occurs in this domain.
- We introduce a discrete-action version of BCQ which achieves a state of the art performance in our batch reinforcement learning setting, and will serve as a strong baseline for future methods.

2 Preliminaries

Reinforcement Learning. Reinforcement learning studies sequential decision making processes, generally formulated by a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the corresponding state and action spaces respectively. At a given discrete time step, a reinforcement learning agent takes action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, and receives a new state $s' \in \mathcal{S}$ and reward $r(s, a, s')$, in accordance to the transition dynamics $p(s', r|s, a)$. The aim of the agent is to maximize the sum of discounted rewards, also known as the return $R_t = \sum_{i=t+1}^{\infty} \gamma^i r(s_i, a_i, s_{i+1})$, where the discount factor $\gamma \in [0, 1)$, determines the effective horizon by weighting future rewards. The decisions of an agent are made by its policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps a given state s to a distribution over actions.

For a given policy π, we define the value function as the expected return of an agent following the policy $Q^\pi(s, a) = \mathbb{E}_\pi[R_t|s, a]$. Given Q^π, a new policy π' of equal or better performance can be derived by greedy maximization $\pi' = \text{argmax}_a Q^\pi(s, a)$ [Sutton and Barto, 1998]. The optimal policy $\pi^* = \text{argmax} Q^*(s, a)$, can be obtained by greedy selection over the optimal value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$. For $\gamma \in [0, 1)$ the value function and optimal value function are the unique fixed points of the Bellman operator $\mathcal{T}^\pi$ and optimality operator $\mathcal{T}^*$, respectively [Bellman, 1957, Bertsekas and Tsitsiklis, 1996]:

$$\mathcal{T}^\pi Q(s, a) = \mathbb{E}_{s', r, a' \sim \pi}[r + \gamma Q(s', a')] \tag{1}$$

$$\mathcal{T}^* Q(s, a) = \mathbb{E}_{s', r}[r + \gamma \max_{a'} Q(s', a')]. \tag{2}$$

Deep Reinforcement Learning. In deep reinforcement learning, the value function is approximated by a neural network Q_θ. In the Deep Q-Network algorithm (DQN) [Mnih et al., 2015], this value function Q_θ is updated in a manner that approximates the optimality operator, through Q-learning [Watkins, 1989]:

$$\mathcal{L}(\theta) = l_\kappa \left(r + \gamma \max_{a'} Q_{\theta'}(s', a') - Q_\theta(s, a) \right), \tag{3}$$

where l_κ defines the Huber loss [Huber et al., 1964]:

$$l_\kappa(\delta) = \begin{cases} 0.5\delta^2 & \text{if } \delta \leq \kappa \\ \kappa(|\delta| - 0.5\kappa) & \text{otherwise.} \end{cases}, \tag{4}$$

but is generally interchangeable with other losses such as mean-squared error. A target network $Q_{\theta'}$ with frozen parameters is used to maintain a fixed target over multiple updates, where θ' is updated to θ after a set number of learning steps. The loss (Eqn. 3) is minimized over mini-batches of transitions (s, a, r, s') sampled from some data set, or replay buffer $\mathcal{B}$ [Lin, 1992]. For an *on-policy* algorithm, $\mathcal{B}$ is generated by the current policy, however, for an *off-policy* algorithm $\mathcal{B}$ may be generated by any collection of policies.

Batch Deep Reinforcement Learning. In *batch reinforcement learning*, we additionally assume the data set is fixed, and no further interactions with the environment will occur. This is in contrast to many off-policy deep reinforcement learning algorithms which assume further interactions with the current policy, but train with a history of experiences generated by previous iterations of the policy. In some instances, access to the behavioral policy is assumed [Precup et al., 2001, Thomas and Brunskill, 2016, Petrik et al., 2016, Laroche et al., 2019], but in our experiments, the behavioral policy is treated as unknown. For notational simplicity, we sometimes refer to a collection of behavioral policies as a single behavioral policy π_b.

Batch deep reinforcement learning algorithms have been shown to be susceptible to *extrapolation error* [Fujimoto et al., 2019], induced by generalization from the neural network function approximator. When selecting actions a' (Eqn. 3), such that (s', a') is distant from data contained in the batch, the estimate $Q_{\theta'}(s', a')$ may be arbitrarily poor, introducing extrapolation error. In systems where further environment interactions are possible this error can be mitigated by simply attempting the action a', which occurs naturally as long as the behavioral policy is similar to the target policy.

3 Batch Deep Reinforcement Learning Algorithms

In this section, we survey recent batch deep reinforcement learning algorithms, including off-policy algorithms which have been to shown to work in a batch setting [Agarwal et al., 2019].

QR-DQN. Quantile Regression DQN (QR-DQN) [Dabney et al., 2017] is a distributional reinforcement learning method [Morimura et al., 2010, Bellemare et al., 2017] which aims to estimate the set of K τ-quantiles of the return distribution, $\{\tau\}^K = \{\frac{i+0.5}{K}\}_{i=0}^{K-1}$. Instead of outputting a single value for each action, QR-DQN outputs a K-dimensional vector representing these quantiles. A pairwise loss between each quantile is computed, similarly to DQN:

$$\mathcal{L}(\theta) = \frac{1}{K^2} \sum_{\tau} \sum_{\tau'} l_\tau \left(r + \gamma \max_{a'} Q^{\tau'}_{\theta'}(s', a') - Q^{\tau}_{\theta}(s, a) \right), \tag{5}$$

where l_τ is a weighted variant of the Huber loss, denoted the quantile Huber loss:

$$l_\tau(\delta) = |\tau - \mathbb{1}_{\delta<0}| l_\kappa(\delta). \tag{6}$$

An estimate of the value can be recovered through the mean over the quantiles, and the policy π is defined by greedy selection over this value:

$$\pi(s) = \operatorname*{argmax}_a \frac{1}{K} \sum_{\tau} Q^{\tau}_{\theta}(s, a). \tag{7}$$

REM. Random Ensemble Mixture (REM) [Agarwal et al., 2019] is an off-policy Q-learning method which aims to capture the success of distributional reinforcement learning algorithms with a simpler algorithm. Similar to QR-DQN, the output of the Q-network is a K-dimensional vector. During

each update, this vector is combined with a convex combination of K weights α_k sampled from a $(K-1)$-simplex:

$$\mathcal{L}(\theta) = l_\kappa \left(r + \gamma \max_{a'} \sum_k \alpha_k Q^k_{\theta'}(s', a') - \sum_k \alpha_k Q^k_\theta(s, a) \right). \tag{8}$$

The policy is defined by the argmax over the mean of the output vector $\pi = \text{argmax}_a \frac{1}{K} \sum Q^k_\theta(s, a)$.

BCQ. Batch-Constrained deep Q-learning (BCQ) [Fujimoto et al., 2019] is a batch reinforcement learning method for continuous control. BCQ aims to perform Q-learning while constraining the action space to eliminate actions which are unlikely to be selected by the behavioral policy π_b, and are therefore unlikely to be contained in the batch. At its core, BCQ uses a state-conditioned generative model $G_\omega : \mathcal{S} \to \mathcal{A}$ to model the distribution of data in the batch, $G_\omega \approx \pi_b$ akin to a behavioral cloning model. As it is easier to sample from $\pi_b(a|s)$ than model $\pi_b(a|s)$ exactly in a continuous action space, the policy is defined by sampling N actions a_i from $G_\omega(s)$ and selecting the highest valued action according to a Q-network. Since BCQ was designed for continuous actions, the method also includes a perturbation model $\xi_\phi(s, a)$, which is a residual added to the sampled actions in the range $[-\Phi, \Phi]$, and trained with the deterministic policy gradient [Silver et al., 2014]. Finally the authors include a weighted version of Clipped Double Q-learning [Fujimoto et al., 2018] to penalize high variance estimates and reduce overestimation bias, using Q^k_θ with $k = \{1, 2\}$:

$$\mathcal{L}(\theta) = \sum_k \left(r + \gamma \max_{\hat{a}} \left(\lambda \min_{k'} Q^{k'}_{\theta'}(s', \hat{a}) + (1-\lambda) \max_{k'} Q^{k'}_{\theta'}(s', \hat{a}) \right) - Q^k_\theta(s, a) \right)^2, \tag{9}$$
$$\text{where } \hat{a} = a_i + \xi_\phi(s', a_i), \qquad a_i \sim G_\omega(s').$$

During evaluation, the policy is defined similarly, by sampling N actions from the generative model, perturbing them and selecting the argmax:

$$\pi(s) = \underset{\hat{a}=a_i+\xi_\phi(s',a_i)}{\text{argmax}} Q^0_\theta(s, \hat{a}), \qquad a_i \sim G_\omega(s). \tag{10}$$

BEAR-QL. Bootstrapping Error Accumulation Reduction Q-Learning (BEAR-QL) [Kumar et al., 2019] is an actor-critic algorithm which builds on the core idea of BCQ, but instead of using a perturbation model, samples actions from a learned actor. As in BCQ, BEAR-QL trains a generative model of the data distribution in the batch. Using the generative model G_ω, the actor π_ϕ is trained using the deterministic policy gradient [Silver et al., 2014], while minimizing the variance over an ensemble of K Q-networks, and constraining the maximum mean discrepancy (MMD) [Gretton et al., 2012] between G_ω and π_ϕ through dual gradient descent:

$$\mathcal{L}(\phi) = -\left(\frac{1}{K} \sum_k Q^k_\theta(s, \hat{a}) - \tau \text{var}_k Q^k_\theta(s, \hat{a}) \right) \text{ s.t. MMD}\left(G_\omega(s), \pi_\phi(s)\right) \leq \epsilon, \tag{11}$$

where $\hat{a} \sim \pi_\phi(s)$ and the MMD is computed over some choice of kernel. The update rule for the ensemble of Q-networks matches BCQ, except the actions $\hat{a}$ can be sampled from the single actor network π_ϕ rather than sampling from a generative model and perturbing:

$$\mathcal{L}(\theta) = \sum_k \left(r + \gamma \max_{\hat{a} \sim \pi_\phi(s')} \left(\lambda \min_{k'} Q^{k'}_{\theta'}(s', \hat{a}) + (1-\lambda) \max_{k'} Q^{k'}_{\theta'}(s', \hat{a}) \right) - Q^k_\theta(s, a) \right)^2. \tag{12}$$

The policy used during evaluation is defined similarly to BCQ, but again samples actions directly from the actor:

$$\pi(s) = \underset{\hat{a}}{\text{argmax}} \frac{1}{K} \sum_k Q^k_\theta(s, \hat{a}), \qquad \hat{a} \sim \pi_\phi(s). \tag{13}$$

KL-Control. KL-Control with Ψ-learning and Monte-Carlo target value estimation is a combination of methods introduced by Jaques et al. [2019] for batch reinforcement learning of a dialog task with discrete actions. KL-control uses a KL-regularized objective to incorporate a prior p into learning, which is weighted by the hyper-parameter c. In this method, the prior is set to a learned estimate of the behavioral policy $p = \pi_b$, again via a generative model G_ω, in similar fashion to BCQ, noting that in a discrete-action setting the probabilities $G_\omega(a|s) \approx \pi_b(a|s)$ can be computed exactly through

behavioral cloning. Rather than a hard maximum in the target, Ψ-learning uses the log over the sum of the exponential of the action-values [Jaques et al., 2017]. Finally, Jaques et al. [2019] estimate a lower-bound over the target value by Monte-Carlo estimation, from sampling K dropout masks [Srivastava et al., 2014, Gal and Ghahramani, 2016] and taking the minimum:

$$\mathcal{L}(\theta) = l_\kappa \left(\log G_\omega(a|s) + \frac{r}{c} + \gamma \min_k \left(\log \sum_{a'} \exp Q^k_{\theta'}(s', a') \right) - Q_\theta(s, a) \right). \tag{14}$$

In Ψ-learning, the policy $\pi(a|s) = \frac{\exp Q_\theta(s,a)}{\sum_{\hat{a} \in \mathcal{A}} \exp Q_\theta(s,\hat{a})}$, as a form of Boltzmann exploration, however in a setting where diversity is not beneficial, we found $\pi = \text{argmax}_a Q_\theta(s, a)$ to achieve higher performance.

SPIBB-DQN. Safe Policy Improvement with Baseline Bootstrapping DQN (SPIBB-DQN) [Laroche et al., 2019] is a safe batch reinforcement learning algorithm for discrete actions which resembles Q-learning, but modifies the policy to match the behavioral policy, or a known baseline, π_b when there is insufficient access to data. Additionally, the authors assume access to some estimate of the state-action distribution of the batch $D(s, a)$. The authors define state-action pairs $(s, a) \in \mathfrak{B}$ as the data points which are unlikely under the data distribution $D(s, a) \leq \epsilon$. For a given state s, the actions a, such that $(s, a) \in \mathfrak{B}$, the policy is set to match the baseline policy $\pi(a|s) = \pi_b(a|s)$. Otherwise, the highest valued action $a \notin \mathfrak{B}$ is set to the remaining probability, defining the policy as follows:

$$\pi(a|s) = \begin{cases} \pi_b(a|s) & \text{if } (s, a) \in \mathfrak{B} \\ \sum_{a \notin \mathfrak{B}} \pi_b(a|s) & \text{if } (s, a) \notin \mathfrak{B} \text{ and } a = \text{argmax}_{a|(s,a) \notin \mathfrak{B}} Q_\theta(s, a) \\ 0 & \text{otherwise.} \end{cases}. \tag{15}$$

The Q-network is updated following the standard DQN update, swapping the max operator with π:

$$\mathcal{L}(\theta) = l_\kappa \left(r + \gamma Q_{\theta'}(s', a') - Q_\theta(s, a) \right), \qquad a' \sim \pi(s'). \tag{16}$$

Although appealing due to its theoretical guarantees in the tabular setting, the authors unfortunately do not include a complete implementation of their deep algorithm in their paper, instead analyzing the algorithm under settings where $D(s, a)$ can be computed almost exactly. However, in principle $D(s, a)$ can be computed with a number of pseudo-count methods [Bellemare et al., 2016, Tang et al., 2017, Burda et al., 2018].

4 Discrete Batch-Constrained Deep Q-learning

In this section, we introduce a discrete variant of the Batch Constrained deep Q-Learning (BCQ) algorithm [Fujimoto et al., 2019]. Much of the complexity of the original algorithm is introduced to deal with the continuous action space, and the core principles of the algorithm can be maintained in a much simpler manner in the discrete setting.

In the original BCQ, a state-conditioned model of the data set G_ω is learned and the highest valued action is selected after sampling actions from G_ω and perturbing the sampled actions within a set range. However, in a discrete-action setting, we can compute the probabilities of every action $G_\omega(a|s) \approx \pi_b(a|s)$, and instead utilize some threshold to eliminate actions:

$$\pi(s) = \underset{a|G_\omega(a|s)/\max \hat{a}\ G_\omega(\hat{a}|s) > \tau}{\text{argmax}} Q_\theta(s, a). \tag{17}$$

To adaptively adjust this threshold, we scale it by the maximum probability from the generative model over all actions, to only allow actions whose relative probability is above some threshold. This results in an algorithm comparable to DQN [Mnih et al., 2015] where the policy is defined by a constrained argmax. The Q-network is trained by swapping the max operation with actions selected by the policy:

$$\mathcal{L}(\theta) = l_\kappa \left(r + \gamma \max_{a'|G_\omega(a'|s')/\max \hat{a}\ G_\omega(\hat{a}|s') > \tau} Q_{\theta'}(s', a') - Q_\theta(s, a) \right). \tag{18}$$

With this threshold τ, we maintain the original property of BCQ where setting $\tau = 0$ returns Q-learning and $\tau = 1$ returns an imitator of the actions contained in the batch.

Given the original BCQ used Clipped Double Q-learning [Fujimoto et al., 2018] to reduce overestimation bias in the continuous-action setting, we instead apply Double DQN [Van Hasselt et al.,

Algorithm 1 BCQ

Input: Batch $\mathcal{B}$, number of iterations T, target_update_rate, mini-batch size N, threshold τ.
Initialize Q-network Q_θ, generative model G_ω and target network $Q_{\theta'}$ with $\theta' \leftarrow \theta$.
for $t = 1$ **to** T **do**
  Sample mini-batch M of N transitions (s, a, r, s') from $\mathcal{B}$.
  $a' = \text{argmax}_{a'|G_\omega(a'|s')/\max_{\hat{a}} G_\omega(\hat{a}|s') > \tau} Q_\theta(s', a')$
  $\theta \leftarrow \text{argmin}_\theta \sum_{(s,a,r,s') \in M} l_\kappa \left(r + \gamma Q_{\theta'}(s', a') - Q_\theta(s, a)\right)$
  $\omega \leftarrow \text{argmin}_\omega - \sum_{(s,a) \in M} \log G_\omega(a|s)$
  If t mod target_update_rate $= 0$: $\theta' \leftarrow \theta$
end for

2016], selecting the max valued action with the current Q-network Q_θ, and evaluating with the target Q-network $Q_{\theta'}$:

$$\mathcal{L}(\theta) = l_\kappa \left(r + \gamma Q_{\theta'}(s', a') - Q_\theta(s, a)\right), \qquad a' = \underset{a'|G_\omega(a'|s')/\max_{\hat{a}} G_\omega(\hat{a}|s') > \tau}{\text{argmax}} Q_\theta(s', a'). \quad (19)$$

The generative model G_ω, effectively a behavioral cloning network, is trained in a standard supervised learning fashion, with a cross-entropy loss. We summarize our discrete BCQ in algorithm 1.

5 Experiments

We validate our methods on the Arcade Learning Environment platform [Bellemare et al., 2013] of Atari 2600 games through OpenAI gym [Brockman et al., 2016]. We use standard preprocessing to image frames and environment rewards [Mnih et al., 2015, Castro et al., 2018], and match the recommendations of Machado et al. [2018] for fair and reproducible results. Exact experimental and algorithmic details are explained in the supplementary. In general, consistent hyper-parameters are kept across all algorithms, and minimal hyper-parameter optimization was performed.

Algorithms. We use each of the algorithms listed in Section 3, other than BEAR-QL and SPIBB-DQN, and use our discrete version of BCQ rather than the original, which was defined for continuous control. We omit BEAR-QL due to the reliance on a continuous action space, similarity to BCQ and number of hyper-parameter choices. Although SPIBB-DQN can be extended to settings where the behavioral policy is estimated [Simão et al., 2019], we omit it due to the lack of implementation for a deep setting without access to pseudo-counts of the dataset, which again would require a large number of design and hyper-parameter choices.

Experimental Setting. We use a partially-trained DQN agent [Mnih et al., 2015] as our behavioral policy. This DQN agent was trained online for 10 million time steps (40 million frames), following a standard training procedure. The behavioral policy is used to gather a new set of 10 million transitions which is used to train each off-policy agent. To ensure exploration, the behavioral policy uses $\epsilon = 0.2$ for the whole episode with $p = 0.8$ and $\epsilon = 0.001$ with $p = 0.2$. This mix of ϵ is used to ensure the data set includes data reaching the max performance of the agent as well as exploratory behavior. Unlike the experiments by Agarwal et al. [2019], the batch data is generated by this single behavioral policy, rather than a series of changing policies. This setup is used to closely match batch settings used by real-world systems, which generally rely on a single behavioral for a fixed period of time [Gauci et al., 2018]. For each environment, the agents are trained on this data set for 10 million time steps, and evaluated on 10 episodes every 50k time steps. We graph both the final performance of the online DQN, as well as the performance of the behavioral policy, the online DQN with exploration noise. Results are displayed in Figure 1. Additionally, in Figure 2 we graph the value estimates of each algorithm to examine if the divergence from extrapolation error [Fujimoto et al., 2019] is present in the Atari domain.

Discussion. Under our experimental conditions, both the online DQN and offline agents have been trained with 10 million data points, where the offline agents are trained with $4\times$ more iterations. Regardless, it is clear from Figure 1 that standard off-policy algorithms (DQN, QR-DQN, REM) perform poorly in this single behavioral policy setting. Out of the three QR-DQN is a clear winner, but generally underperforms the noisy behavioral policy. While Agarwal et al. [2019] showed these algorithms performed well with large replay buffers and high diversity, by training agents using the

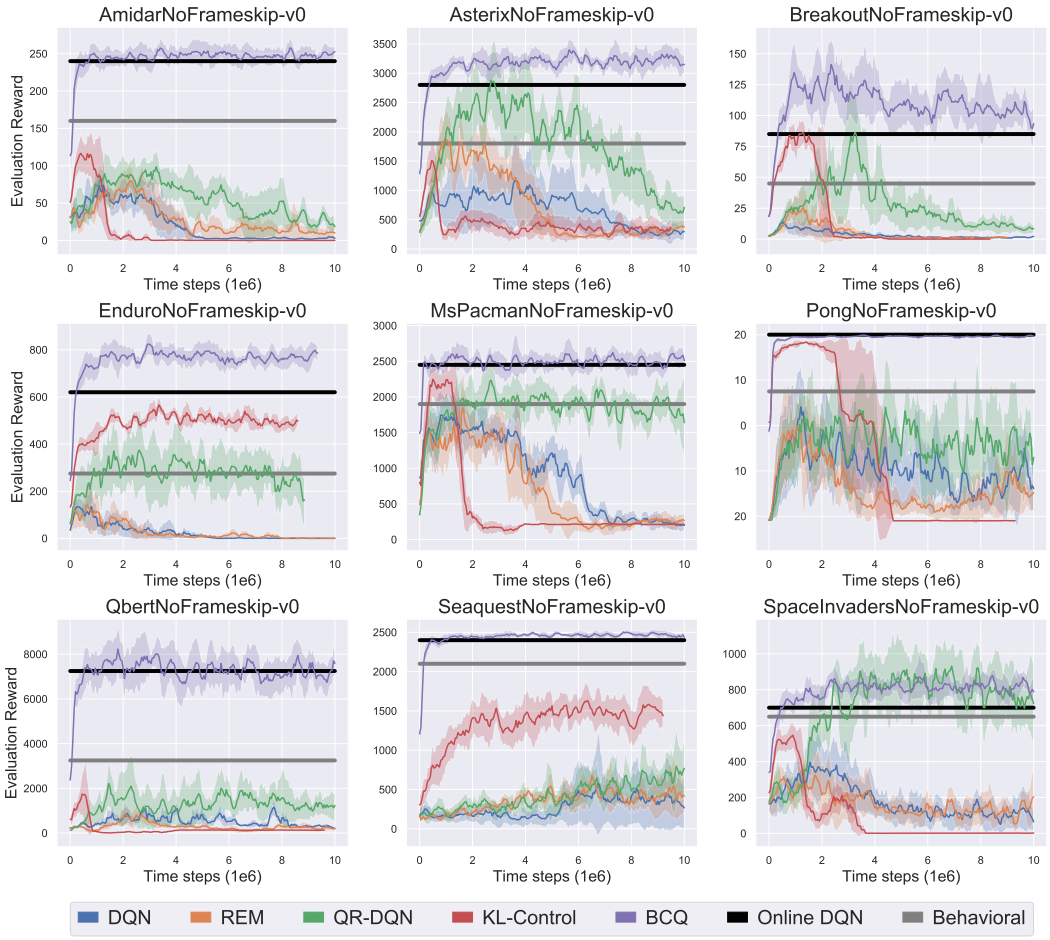

Figure 1: Results on 9 Atari 2600 games. Agents are trained from a buffer of 10 million transitions collected by a single partially-trained DQN. Performance of online DQN, and the behavorial policy (online DQN with added noise) is included. Agents are evaluated every 50k time steps over 10 episodes, and averaged over 3 seeds and a sliding window of 5. The shaded area measures a single standard deviation across trials.

entire replay buffer from training a DQN agent (50 million transitions), it is clear there is a reliance on their specific setting for these algorithms to perform well. We do remark that our results confirm their observation that distributional reinforcement learning algorithms (QR-DQN) outperforms their standard counterpart (DQN), suggesting that learning a distribution aids in exploitation.

In comparison to the off-policy algorithms, the batch reinforcement learning algorithms perform reasonably well. BCQ, in particular, outperforms every other method in all tested games. However, these results also shown the current downsides of these methods. Although BCQ has the strongest performance, on most games it only matches the performance of the online DQN, which is the underlying noise-free behavioral policy. These results suggest BCQ achieves something closer to robust imitation, rather than true batch reinforcement learning when there is limited exploratory data. KL-Control often demonstrates a strong initial performance before failing. Examining Figure 2, the drop in performance corresponds to a negative divergence in the value estimate. In the games where the value estimate does not diverge (Enduro and Seaquest), KL-Control performs well. It is possible that with additional hyper-parameter tuning, stability could be maintained, however, this suggests that KL-Control is not robust to hyper-parameters or varied tasks.

These results additionally confirm the experiments from Fujimoto et al. [2019], which showed that standard off-policy deep reinforcement learning algorithms fail in the batch setting, due to high extrapolation error from selecting out-of-distribution actions during value updates. Furthermore, it is

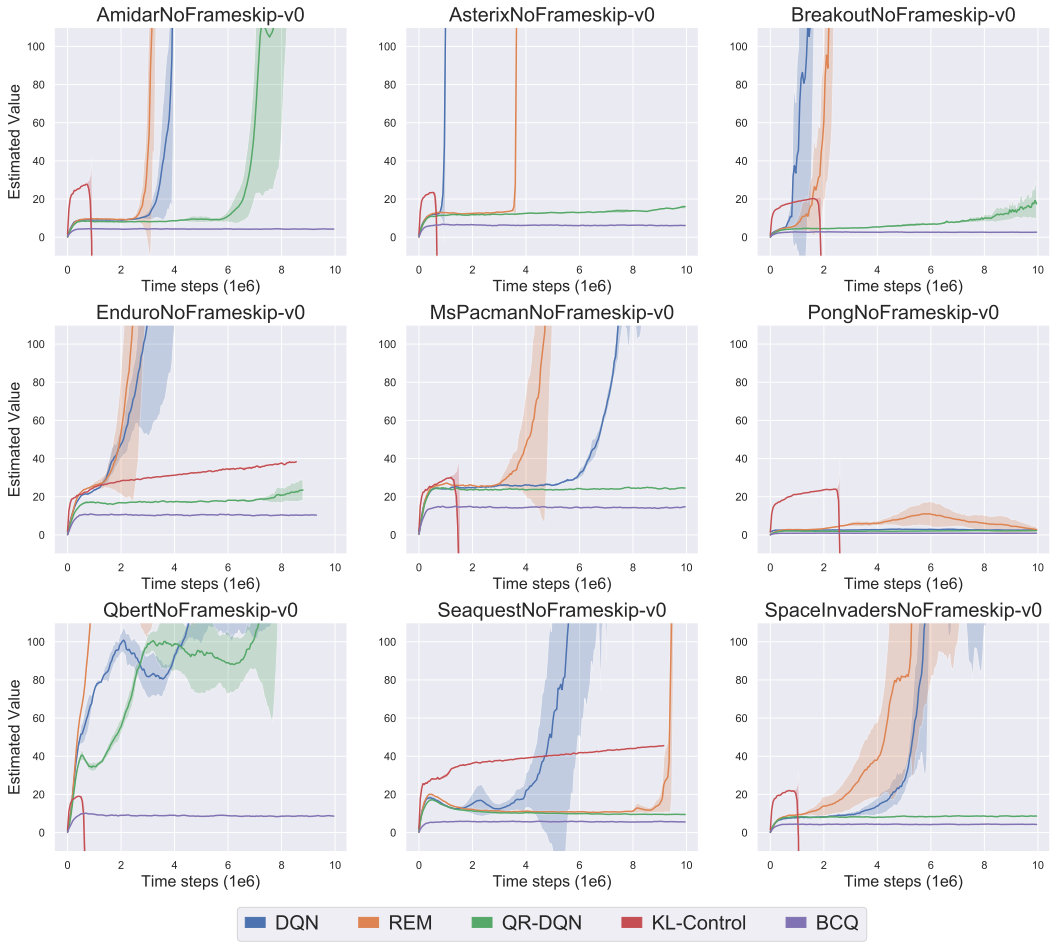

Figure 2: Value estimates from the Q-networks of each agent on 9 Atari 2600 games. This figure shows which agents have stable outputs, demonstrating a resistance to extrapolation error. Additionally, drops in performance (Figure 1) can be seen to correspond to divergence in the value estimates. Value estimates are averaged over 5000 mini-batches of 32 and 3 seeds. The shaded area measured a single standard deviation across trials, clipped to 100 for visual clarity.

clear that the algorithms with the strongest performance also have stable value estimates, suggesting that the mitigation of extrapolation error is important for batch deep reinforcement learning.

6 Conclusion

In this paper, we perform empirical analysis on current off-policy and batch reinforcement learning algorithms in a simple single behavioral policy task on several Atari environments [Bellemare et al., 2013]. Our experiments show that current algorithms fail to achieve a satisfactory performance in this setting, by under-performing online DQN and the behavioral policy. Our results suggest that algorithms which do not consider extrapolation error or the distribution of data will perform poorly in the batch setting with low data diversity, due to unstable value estimates. Lastly, we introduce a discrete version of Batch-Constrained deep Q-learning (BCQ) [Fujimoto et al., 2019], which outperforms all previous algorithms in this setting, while being straightforward to implement. We hope BCQ will serve as a strong baseline for future methods in this area.

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for simplicity in off-policy deep reinforcement learning. *arXiv preprint arXiv:1907.04543*, 2019.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458, 2017.

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Dimitri P Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena scientific Belmont, MA, 1996.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. *arXiv preprint arXiv:1710.10044*, 2017.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, volume 80, pages 1587–1596, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, Xiaohui Ye, Zhengxing Chen, and Scott Fujimoto. Horizon: Facebook's open source applied reinforcement learning platform. *arXiv preprint arXiv:1811.00260*, 2018.

Geoffrey J Gordon. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*, pages 261–268. Elsevier, 1995.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

Arthur Guez, Robert D Vincent, Massimo Avoli, and Joelle Pineau. Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *Proceedings of the 20th national conference on Innovative applications of artificial intelligence-Volume 3*, pages 1671–1678. AAAI Press, 2008.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

Peter J Huber et al. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.

Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, José Miguel Hernández-Lobato, Richard E Turner, and Douglas Eck. Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1645–1654. JMLR. org, 2017.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661, 2019.

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. *arXiv preprint arXiv:1509.02971*, 2015.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Nonparametric return distribution approximation for reinforcement learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 799–806, 2010.

Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine learning*, 49(2-3): 161–178, 2002.

Marek Petrik, Mohammad Ghavamzadeh, , and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems*, pages 2298–2306, 2016.

Olivier Pietquin, Matthieu Geist, Senthilkumar Chandramohan, and Hervé Frezza-Buet. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)*, 7(3):7, 2011.

Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *International Conference on Machine Learning*, pages 417–424, 2001.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

Thiago D Simão, Romain Laroche, and Rémi Tachet des Combes. Safe policy improvement with an estimated baseline policy. *arXiv preprint arXiv:1909.05236*, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pages 2753–2762, 2017.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016.

Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

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

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

A Experimental Details

A.1 Atari Preprocessing.

The Atari 2600 environment is preprocessed in the same manner as previous work [Mnih et al., 2015, Machado et al., 2018, Castro et al., 2018] and we use consistent preprocessing across all tasks and algorithms.

We denote the output of the Atari environment as *frames*. These frames are grayscaled and resized to 84 by 84 pixels. Furthermore, the agent only receives a *state* and selects an action every 4th frame. The selected action is repeated for the next 4 frames. The state is defined by the maximum between the previous two frames. Furthermore, the input to the networks is a concatenation of the previous 4 states. This means each network receives a tensor with dimensions $(4, 84, 84)$, which considers a history of 16 frames (4 frames over 4 states). For the first 3 time steps, the input to the networks includes *states* which are set to all 0s. If the environment terminates before 4 frames have passed, the state is defined by the maximum of the final two frames before termination.

In accordance to Machado et al. [2018], sticky actions are used, such that the action a_t is set to the previous action a_{t-1} with probability $p = 0.25$. No-operations are not applied at the beginning of episodes, and random frame skips are not used.

The reward function is defined by the in-game reward, but clipped to a range of $[-1, 1]$. The environment terminates when the game terminates (rather than on a lost life), or after 27k time steps, corresponding to 108k frames or 30 minutes of real time.

A.2 Architecture and Hyper-parameters.

Unless stated otherwise, all networks use the same architecture and hyper-parameters.

Image inputs are passed through a 3-layered convolutional neural network taking an input size of $(4, 84, 84)$. The first layer has a kernel depth of 32, kernel size of 8×8 and stride 4. The second layer has a kernel depth of 32, kernel size of 4×4 and stride 2. The third layer has a kernel depth of 64, kernel size of 2×2 and stride 1. This output is flattened into a vector of 3136 and passed to a full-connected network with one hidden layer of 512. The output of the Q-network is a Q-value for each action. ReLU activation functions are used after layer besides the final fully-connected layer.

For methods which require a generative model, the convolutional neural network is shared between both the Q-network and the generative model. The generative model is a secondary fully-connected network with the same architecture. The final layer uses a softmax activation after the output of the network, to recover probabilities for each action.

Hyper-parameters are held consistent across each algorithm and listed in Table 1. Hyper-parameters were chosen to match the implementation of Rainbow [Hessel et al., 2017] in the Dopamine framework [Castro et al., 2018].

Table 1: Hyper-parameters used by each network.

Hyper-parameter	Value
Network optimizer	Adam [Kingma and Ba, 2014]
Learning rate	0.0000625
Adam ϵ	0.00015
Discount γ	0.99
Mini-batch size	32
Target network update frequency	8k training iterations
Huber loss κ	1
Evaluation ϵ	0.001

Algorithm-specific hyper-parameters are listed in Table 2. Additionally, we regularize the generative model G_ω used in BCQ and KL-Control by a penalty on the final pre-activation output x by $0.01x^2$. For KL-Control, dropout is applied before both of the fully-connected layers.

Additionally, we list the hyper-parameters of the online DQN, which served as the behavioral policy, in Table 3. Training frequency corresponds to how often a training update was performed. Warmup

Table 2: Algorithm-specific hyper-parameters.

Algorithm	Hyper-parameter	Value
QR-DQN	Quantiles K	50
REM	Heads K	200
BCQ	Threshold τ	0.3
KL-Control	Dropout masks K	5
	KL weighting c	2
	Gradient clipping	1.0
	Dropout probability	0.2

time steps defines the initial period where actions are randomly selected and stored in the replay buffer, before any training occurs. We use ϵ-greedy for exploration, where ϵ is decayed over time.

Table 3: Hyper-parameters used by online DQN.

Hyper-parameter	Value
Replay buffer size	1 million
Training frequency	Every 4th time step
Warmup time steps	20k time steps
Initial ϵ	1.0
Final ϵ	0.01
ϵ decay period	250k training iterations