Off-Policy Deep Reinforcement Learning without Exploration

Scott Fujimoto [1 2] **David Meger** [1 2] **Doina Precup** [1 2]

Abstract

Many practical applications of reinforcement learning constrain agents to learn from a fixed batch of data which has already been gathered, without offering further possibility for data collection. In this paper, we demonstrate that due to errors introduced by extrapolation, standard off-policy deep reinforcement learning algorithms, such as DQN and DDPG, are incapable of learning without data correlated to the distribution under the current policy, making them ineffective for this fixed batch setting. We introduce a novel class of off-policy algorithms, batch-constrained reinforcement learning, which restricts the action space in order to force the agent towards behaving close to on-policy with respect to a subset of the given data. We present the first continuous control deep reinforcement learning algorithm which can learn effectively from arbitrary, fixed batch data, and empirically demonstrate the quality of its behavior in several tasks.

1. Introduction

Batch reinforcement learning, the task of learning from a fixed dataset without further interactions with the environment, is a crucial requirement for scaling reinforcement learning to tasks where the data collection procedure is costly, risky, or time-consuming. Off-policy batch reinforcement learning has important implications for many practical applications. It is often preferable for data collection to be performed by some secondary controlled process, such as a human operator or a carefully monitored program. If assumptions on the quality of the behavioral policy can be made, imitation learning can be used to produce strong policies. However, most imitation learning algorithms are known to fail when exposed to suboptimal trajectories, or require further interactions with the environment to compensate (Hester et al., 2017; Sun et al., 2018; Cheng et al., 2018). On the other hand, batch reinforcement learning offers a mechanism for learning from a fixed dataset without restrictions on the quality of the data.

Most modern off-policy deep reinforcement learning algorithms fall into the category of *growing batch learning* (Lange et al., 2012), in which data is collected and stored into an experience replay dataset (Lin, 1992), which is used to train the agent before further data collection occurs. However, we find that these "off-policy" algorithms can fail in the batch setting, becoming unsuccessful if the dataset is uncorrelated to the true distribution under the current policy. Our most surprising result shows that off-policy agents perform dramatically worse than the behavioral agent *when trained with the same algorithm on the same dataset*.

This inability to learn truly off-policy is due to a fundamental problem with off-policy reinforcement learning we denote *extrapolation error*, a phenomenon in which unseen state-action pairs are erroneously estimated to have unrealistic values. Extrapolation error can be attributed to a mismatch in the distribution of data induced by the policy and the distribution of data contained in the batch. As a result, it may be impossible to learn a value function for a policy which selects actions not contained in the batch.

To overcome extrapolation error in off-policy learning, we introduce batch-constrained reinforcement learning, where agents are trained to maximize reward while minimizing the mismatch between the state-action visitation of the policy and the state-action pairs contained in the batch. Our deep reinforcement learning algorithm, Batch-Constrained deep Q-learning (BCQ), uses a state-conditioned generative model to produce only previously seen actions. This generative model is combined with a Q-network, to select the highest valued action which is similar to the data in the batch. Under mild assumptions, we prove this batch-constrained paradigm is necessary for unbiased value estimation from incomplete datasets for finite deterministic MDPs.

Unlike any previous continuous control deep reinforcement learning algorithms, BCQ is able to learn successfully without interacting with the environment by considering extrapolation error. Our algorithm is evaluated on a series of batch reinforcement learning tasks in MuJoCo environ-

[1]Department of Computer Science, McGill University, Montreal, Canada [2]Mila Québec AI Institute. Correspondence to: Scott Fujimoto <scott.fujimoto@mail.mcgill.ca>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

ments (Todorov et al., 2012; Brockman et al., 2016), where extrapolation error is particularly problematic due to the high-dimensional continuous action space, which is impossible to sample exhaustively. Our algorithm offers a unified view on imitation and off-policy learning, and is capable of learning from purely expert demonstrations, as well as from finite batches of suboptimal data, without further exploration. We remark that BCQ is only one way to approach batch-constrained reinforcement learning in a deep setting, and we hope that it will be serve as a foundation for future algorithms. To ensure reproducibility, we provide precise experimental and implementation details, and our code is made available (`https://github.com/sfujim/BCQ`).

2. Background

In reinforcement learning, an agent interacts with its environment, typically assumed to be a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, p_M, r, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, and transition dynamics $p_M(s'|s,a)$. At each discrete time step, the agent receives a reward $r(s,a,s') \in \mathbb{R}$ for performing action a in state s and arriving at the state s'. The goal of the agent is to maximize the expectation of the sum of discounted rewards, known as the return $R_t = \sum_{i=t+1}^{\infty} \gamma^i r(s_i, a_i, s_{i+1})$, which weighs future rewards with respect to the discount factor $\gamma \in [0, 1)$.

The agent selects actions with respect to a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which exhibits a distribution $\mu^\pi(s)$ over the states $s \in \mathcal{S}$ visited by the policy. Each policy π has a corresponding value function $Q^\pi(s,a) = \mathbb{E}_\pi[R_t|s,a]$, the expected return when following the policy after taking action a in state s. For a given policy π, the value function can be computed through the Bellman operator $\mathcal{T}^\pi$:

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

The Bellman operator is a contraction for $\gamma \in [0,1)$ with unique fixed point $Q^\pi(s,a)$ (Bertsekas & Tsitsiklis, 1996). $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ is known as the optimal value function, which has a corresponding optimal policy obtained through greedy action choices. For large or continuous state and action spaces, the value can be approximated with neural networks, e.g. using the Deep Q-Network algorithm (DQN) (Mnih et al., 2015). In DQN, the value function Q_θ is updated using the target:

$$r + \gamma Q_{\theta'}(s', \pi(s')), \quad \pi(s') = \text{argmax}_a\, Q_{\theta'}(s', a), \tag{2}$$

Q-learning is an *off-policy* algorithm (Sutton & Barto, 1998), meaning the target can be computed without consideration of how the experience was generated. In principle, off-policy reinforcement learning algorithms are able to learn from data collected by any behavioral policy. Typically, the loss is minimized over mini-batches of tuples of the agent's past data, $(s,a,r,s') \in \mathcal{B}$, sampled from an experience replay dataset $\mathcal{B}$ (Lin, 1992). For shorthand, we often write $s \in \mathcal{B}$ if there exists a transition tuple containing s in the batch $\mathcal{B}$, and similarly for (s,a) or $(s,a,s') \in \mathcal{B}$. In batch reinforcement learning, we assume $\mathcal{B}$ is fixed and no further interaction with the environment occurs. To further stabilize learning, a target network with frozen parameters $Q_{\theta'}$, is used in the learning target. The parameters of the target network θ' are updated to the current network parameters θ after a fixed number of time steps, or by averaging $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ for some small τ (Lillicrap et al., 2015).

In a continuous action space, the analytic maximum of Equation (2) is intractable. In this case, actor-critic methods are commonly used, where action selection is performed through a separate policy network π_ϕ, known as the actor, and updated with respect to a value estimate, known as the critic (Sutton & Barto, 1998; Konda & Tsitsiklis, 2003). This policy can be updated following the deterministic policy gradient theorem (Silver et al., 2014):

$$\phi \leftarrow \text{argmax}_\phi\, \mathbb{E}_{s \in \mathcal{B}}[Q_\theta(s, \pi_\phi(s))], \tag{3}$$

which corresponds to learning an approximation to the maximum of Q_θ, by propagating the gradient through both π_ϕ and Q_θ. When combined with off-policy deep Q-learning to learn Q_θ, this algorithm is referred to as Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015).

3. Extrapolation Error

Extrapolation error is an error in off-policy value learning which is introduced by *the mismatch between the dataset and true state-action visitation of the current policy*. The value estimate $Q(s,a)$ is affected by extrapolation error during a value update where the target policy selects an unfamiliar action a' at the next state s' in the backed-up value estimate, such that (s',a') is unlikely, or not contained, in the dataset. The cause of extrapolation error can be attributed to several related factors:

Absent Data. If any state-action pair (s,a) is unavailable, then error is introduced as some function of the amount of similar data and approximation error. This means that the estimate of $Q_\theta(s', \pi(s'))$ may be arbitrarily bad without sufficient data near $(s', \pi(s'))$.

Model Bias. When performing off-policy Q-learning with a batch $\mathcal{B}$, the Bellman operator $\mathcal{T}^\pi$ is approximated by sampling transitions tuples (s,a,r,s') from $\mathcal{B}$ to estimate the expectation over s'. However, for a stochastic MDP, without infinite state-action visitation, this produces a biased estimate of the transition dynamics:

$$\mathcal{T}^\pi Q(s,a) \approx \mathbb{E}_{s' \sim \mathcal{B}}[r + \gamma Q(s', \pi(s'))], \tag{4}$$

where the expectation is with respect to transitions in the batch $\mathcal{B}$, rather than the true MDP.

Training Mismatch. Even with sufficient data, in deep Q-learning systems, transitions are sampled uniformly from the dataset, giving a loss weighted with respect to the likelihood of data in the batch:

$$\approx \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} ||r + \gamma Q_{\theta'}(s', \pi(s')) - Q_\theta(s,a)||^2. \quad (5)$$

If the distribution of data in the batch does not correspond with the distribution under the current policy, the value function may be a poor estimate of actions selected by the current policy, due to the mismatch in training.

We remark that re-weighting the loss in Equation (5) with respect to the likelihood under the current policy can still result in poor estimates if state-action pairs with high likelihood under the current policy are not found in the batch. This means only a subset of possible policies can be evaluated accurately. As a result, learning a value estimate with off-policy data can result in large amounts of extrapolation error if the policy selects actions which are not similar to the data found in the batch. In the following section, we discuss how state of the art off-policy deep reinforcement learning algorithms fail to address the concern of extrapolation error, and demonstrate the implications in practical examples.

3.1. Extrapolation Error in Deep Reinforcement Learning

Deep Q-learning algorithms (Mnih et al., 2015) have been labeled as off-policy due to their connection to off-policy Q-learning (Watkins, 1989). However, these algorithms tend to use near-on-policy exploratory policies, such as ϵ-greedy, in conjunction with a replay buffer (Lin, 1992). As a result, the generated dataset tends to be heavily correlated to the current policy. In this section, we examine how these off-policy algorithms perform when learning with uncorrelated datasets. Our results demonstrate that the performance of a state of the art deep actor-critic algorithm, DDPG (Lillicrap et al., 2015), deteriorates rapidly when the data is uncorrelated and the value estimate produced by the deep Q-network diverges. These results suggest that off-policy deep reinforcement learning algorithms are ineffective when learning *truly off-policy*.

Our practical experiments examine three different batch settings in OpenAI gym's Hopper-v1 environment (Todorov et al., 2012; Brockman et al., 2016), which we use to train an off-policy DDPG agent with no interaction with the environment. Experiments with additional environments and specific details can be found in the Supplementary Material.

Batch 1 (Final buffer). We train a DDPG agent for 1 million time steps, adding $\mathcal{N}(0, 0.5)$ Gaussian noise to actions for high exploration, and store all experienced transitions. This collection procedure creates a dataset with a diverse set of states and actions, with the aim of sufficient coverage.

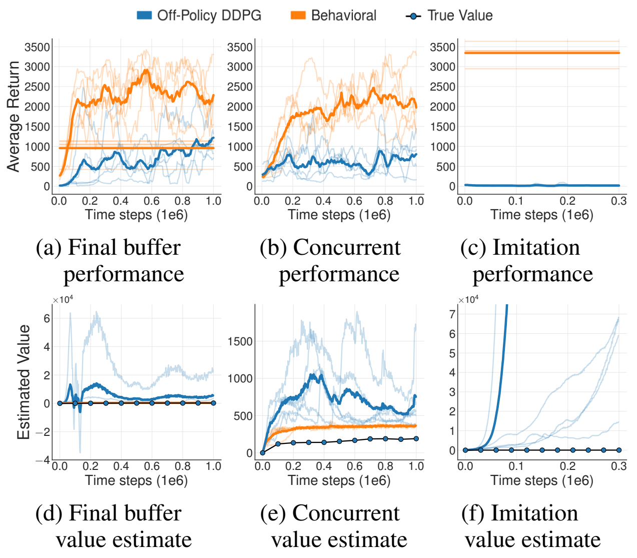


Figure 1. We examine the performance (top row) and corresponding value estimates (bottom row) of DDPG in three batch tasks on Hopper-v1. Each individual trial is plotted with a thin line, with the mean in bold (evaluated without exploration noise). Straight lines represent the average return of episodes contained in the batch (with exploration noise). An estimate of the true value of the off-policy agent, evaluated by Monte Carlo returns, is marked by a dotted line. In all three experiments, we observe a large gap in the performance between the behavioral and off-policy agent, even when learning from the same dataset (*concurrent*). Furthermore, the value estimates are unstable or divergent across all tasks.

Batch 2 (Concurrent). We concurrently train the off-policy and behavioral DDPG agents, for 1 million time steps. To ensure sufficient exploration, a standard $\mathcal{N}(0, 0.1)$ Gaussian noise is added to actions taken by the behavioral policy. Each transition experienced by the behavioral policy is stored in a buffer replay, which both agents learn from. As a result, both agents are trained with the identical dataset.

Batch 3 (Imitation). A trained DDPG agent acts as an expert, and is used to collect a dataset of 1 million transitions.

In Figure 1, we graph the performance of the agents as they train with each batch, as well as their value estimates. Straight lines represent the average return of episodes contained in the batch. Additionally, we graph the learning performance of the behavioral agent for the relevant tasks.

Our experiments demonstrate several surprising facts about off-policy deep reinforcement learning agents. In each task, the off-policy agent performances significantly worse than the behavioral agent. Even in the *concurrent* experiment, where both agents are trained with the same dataset, there is a large gap in performance in every single trial. This result suggests that differences in the state distribution under the initial policies is enough for extrapolation error to drastically offset the performance of the off-policy agent. Additionally, the corresponding value estimate exhibits divergent behavior, while the value estimate of the behavioral agent is highly stable. In the *final buffer* experiment, the off-policy agent is provided with a large and diverse dataset, with the aim of providing sufficient coverage of the initial policy. Even in this instance, the value estimate is highly unstable, and the performance suffers. In the *imitation* setting, the agent is provided with expert data. However, the agent quickly learns to take non-expert actions, under the guise of optimistic extrapolation. As a result, the value estimates rapidly diverge and the agent fails to learn.

Although extrapolation error is not necessarily positively biased, when combined with maximization in reinforcement learning algorithms, extrapolation error provides a source of noise that can induce a persistent overestimation bias (Thrun & Schwartz, 1993; Van Hasselt et al., 2016; Fujimoto et al., 2018). In an on-policy setting, extrapolation error may be a source of beneficial exploration through an implicit "optimism in the face of uncertainty" strategy (Lai & Robbins, 1985; Jaksch et al., 2010). In this case, if the value function overestimates an unknown state-action pair, the policy will collect data in the region of uncertainty, and the value estimate will be corrected. However, when learning off-policy, or in a batch setting, extrapolation error will never be corrected due to the inability to collect new data.

These experiments show extrapolation error can be highly detrimental to learning off-policy in a batch reinforcement learning setting. While the continuous state space and multi-dimensional action space in MuJoCo environments are contributing factors to extrapolation error, the scale of these tasks is small compared to real world settings. As a result, even with a sufficient amount of data collection, extrapolation error may still occur due to the concern of catastrophic forgetting (McCloskey & Cohen, 1989; Goodfellow et al., 2013). Consequently, off-policy reinforcement learning algorithms used in the real-world will require practical guarantees without exhaustive amounts of data.

4. Batch-Constrained Reinforcement Learning

Current off-policy deep reinforcement learning algorithms fail to address extrapolation error by selecting actions with respect to a learned value estimate, without consideration of the accuracy of the estimate. As a result, certain out-of-distribution actions can be erroneously extrapolated to higher values. However, the value of an off-policy agent *can* be accurately evaluated in regions where data is available. We propose a conceptually simple idea: to avoid extrapolation error *a policy should induce a similar state-action visitation to the batch*. We denote policies which satisfy this notion as *batch-constrained*. To optimize off-policy learning for a given batch, batch-constrained policies are trained to select actions with respect to three objectives:

(1) Minimize the distance of selected actions to the data in the batch.
(2) Lead to states where familiar data can be observed.
(3) Maximize the value function.

We note the importance of objective (1) above the others, as the value function and estimates of future states may be arbitrarily poor without access to the corresponding transitions. That is, we cannot correctly estimate (2) and (3) unless (1) is sufficiently satisfied. As a result, we propose optimizing the value function, along with some measure of future certainty, with a constraint limiting the distance of selected actions to the batch. This is achieved in our deep reinforcement learning algorithm through a state-conditioned generative model, to produce likely actions under the batch. This generative model is combined with a network which aims to optimally perturb the generated actions in a small range, along with a Q-network, used to select the highest valued action. Finally, we train a pair of Q-networks, and take the minimum of their estimates during the value update. This update penalizes states which are unfamiliar, and pushes the policy to select actions which lead to certain data.

We begin by analyzing the theoretical properties of batch-constrained policies in a finite MDP setting, where we are able to quantify extrapolation error precisely. We then introduce our deep reinforcement learning algorithm in detail, Batch-Constrained deep Q-learning (BCQ) by drawing inspiration from the tabular analogue.

4.1. Addressing Extrapolation Error in Finite MDPs

In the finite MDP setting, extrapolation error can be described by the bias from the mismatch between the transitions contained in the buffer and the true MDP. We find that by inducing a data distribution that is contained entirely within the batch, batch-constrained policies can eliminate extrapolation error entirely for deterministic MDPs. In addition, we show that the batch-constrained variant of Q-learning converges to the optimal policy under the same conditions as the standard form of Q-learning. Moreover, we prove that for a deterministic MDP, batch-constrained Q-learning is guaranteed to match, or outperform, the behavioral policy when starting from any state contained in the batch. All of the proofs for this section can be found in the Supplementary Material.

A value estimate Q can be learned using an experience replay buffer $\mathcal{B}$. This involves sampling transition tuples (s, a, r, s') with uniform probability, and applying the temporal difference update (Sutton, 1988; Watkins, 1989):

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma Q(s', \pi(s'))\right). \quad (6)$$

If $\pi(s') = \operatorname{argmax}_{a'} Q(s', a')$, this is known as Q-learning. Assuming a non-zero probability of sampling any possible transition tuple from the buffer and infinite updates, Q-learning converges to the optimal value function.

We begin by showing that the value function Q learned with the batch $\mathcal{B}$ corresponds to the value function for an alternate MDP $M_{\mathcal{B}}$. From the true MDP M and initial values $Q(s,a)$, we define the new MDP $M_{\mathcal{B}}$ with the same action and state space as M, along with an additional terminal state s_{init}. $M_{\mathcal{B}}$ has transition probabilities $p_{\mathcal{B}}(s'|s,a) = \frac{N(s,a,s')}{\sum_{\tilde{s}} N(s,a,\tilde{s})}$, where $N(s,a,s')$ is the number of times the tuple (s,a,s') is observed in $\mathcal{B}$. If $\sum_{\tilde{s}} N(s,a,\tilde{s}) = 0$, then $p_{\mathcal{B}}(s_{\text{init}}|s,a) = 1$, where $r(s,a,s_{\text{init}})$ is set to the initialized value of $Q(s,a)$.

Theorem 1. *Performing Q-learning by sampling from a batch $\mathcal{B}$ converges to the optimal value function under the MDP $M_{\mathcal{B}}$.*

We define ϵ_{MDP} as the tabular extrapolation error, which accounts for the discrepancy between the value function $Q^\pi_{\mathcal{B}}$ computed with the batch $\mathcal{B}$ and the value function Q^π computed with the true MDP M:

$$\epsilon_{\text{MDP}}(s,a) = Q^\pi(s,a) - Q^\pi_{\mathcal{B}}(s,a). \tag{7}$$

For any policy π, the exact form of $\epsilon_{\text{MDP}}(s,a)$ can be computed through a Bellman-like equation:

$$\begin{aligned} \epsilon_{\text{MDP}}(s,a) &= \sum_{s'} \left(p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a)\right) \\ &\quad \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a') \right) \\ &\quad + p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s') \epsilon_{\text{MDP}}(s',a'). \end{aligned} \tag{8}$$

This means extrapolation error is a function of divergence in the transition distributions, weighted by value, along with the error at succeeding states. If the policy is chosen carefully, the error between value functions can be minimized by visiting regions where the transition distributions are similar. For simplicity, we denote

$$\epsilon^\pi_{\text{MDP}} = \sum_s \mu_\pi(s) \sum_a \pi(a|s) |\epsilon_{\text{MDP}}(s,a)|. \tag{9}$$

To evaluate a policy π exactly at relevant state-action pairs, only $\epsilon^\pi_{\text{MDP}} = 0$ is required. We can then determine the condition required to evaluate the exact expected return of a policy without extrapolation error.

Lemma 1. *For all reward functions, $\epsilon^\pi_{MDP} = 0$ if and only if $p_{\mathcal{B}}(s'|s,a) = p_M(s'|s,a)$ for all $s' \in \mathcal{S}$ and (s,a) such that $\mu_\pi(s) > 0$ and $\pi(a|s) > 0$.*

Lemma 1 states that if $M_{\mathcal{B}}$ and M exhibit the same transition probabilities in regions of relevance, the policy can be accurately evaluated. For a stochastic MDP this may require an infinite number of samples to converge to the true distribution, however, for a deterministic MDP this requires only a single transition. This means a policy which only traverses transitions contained in the batch, can be evaluated without error. More formally, we denote a policy $\pi \in \Pi_{\mathcal{B}}$ as *batch-constrained* if for all (s,a) where $\mu_\pi(s) > 0$ and $\pi(a|s) > 0$ then $(s,a) \in \mathcal{B}$. Additionally, we define a batch $\mathcal{B}$ as *coherent* if for all $(s,a,s') \in \mathcal{B}$ then $s' \in \mathcal{B}$ unless s' is a terminal state. This condition is trivially satisfied if the data is collected in trajectories, or if all possible states are contained in the batch. With a coherent batch, we can guarantee the existence of a batch-constrained policy.

Theorem 2. *For a deterministic MDP and all reward functions, $\epsilon^\pi_{MDP} = 0$ if and only if the policy π is batch-constrained. Furthermore, if $\mathcal{B}$ is coherent, then such a policy must exist if the start state $s_0 \in \mathcal{B}$.*

Batch-constrained policies can be used in conjunction with Q-learning to form batch-constrained Q-learning (BCQL), which follows the standard tabular Q-learning update while constraining the possible actions with respect to the batch:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \text{s.t.} (s',a') \in \mathcal{B}} Q(s',a')). \tag{10}$$

BCQL converges under the same conditions as the standard form of Q-learning, noting the batch-constraint is nonrestrictive given infinite state-action visitation.

Theorem 3. *Given the Robbins-Monro stochastic convergence conditions on the learning rate α, and standard sampling requirements from the environment, BCQL converges to the optimal value function Q^*.*

The more interesting property of BCQL is that for a deterministic MDP and any coherent batch $\mathcal{B}$, BCQL converges to the optimal batch-constrained policy $\pi^* \in \Pi_{\mathcal{B}}$ such that $Q^{\pi^*}(s,a) \geq Q^\pi(s,a)$ for all $\pi \in \Pi_{\mathcal{B}}$ and $(s,a) \in \mathcal{B}$.

Theorem 4. *Given a deterministic MDP and coherent batch $\mathcal{B}$, along with the Robbins-Monro stochastic convergence conditions on the learning rate α and standard sampling requirements on the batch $\mathcal{B}$, BCQL converges to $Q^\pi_{\mathcal{B}}(s,a)$ where $\pi^*(s) = \operatorname{argmax}_{a \text{ s.t.} (s,a) \in \mathcal{B}} Q^\pi_{\mathcal{B}}(s,a)$ is the optimal batch-constrained policy.*

This means that BCQL is guaranteed to outperform any behavioral policy when starting from any state contained in the batch, effectively outperforming imitation learning. Unlike standard Q-learning, there is no condition on state-action visitation, other than coherency in the batch.

4.2. Batch-Constrained Deep Reinforcement Learning

We introduce our approach to off-policy batch reinforcement learning, Batch-Constrained deep Q-learning (BCQ). BCQ approaches the notion of batch-constrained through a generative model. For a given state, BCQ generates plausible candidate actions with high similarity to the batch, and then selects the highest valued action through a learned Q-network. Furthermore, we bias this value estimate to penalize rare, or unseen, states through a modification to Clipped Double Q-learning (Fujimoto et al., 2018). As a result, BCQ learns a policy with a similar state-action visitation to the data in the batch, as inspired by the theoretical benefits of its tabular counterpart.

To maintain the notion of batch-constraint, we define a similarity metric by making the assumption that for a given state s, the similarity between (s,a) and the state-action pairs in the batch $\mathcal{B}$ can be modelled using a learned state-conditioned marginal likelihood $P^G_{\mathcal{B}}(a|s)$. In this case, it follows that the policy maximizing $P^G_{\mathcal{B}}(a|s)$ would minimize the error induced by extrapolation from distant, or unseen, state-action pairs, by only selecting the most likely actions in the batch with respect to a given state. Given the difficulty of estimating $P^G_{\mathcal{B}}(a|s)$ in high-dimensional continuous spaces, we instead train a parametric generative model of the batch $G_\omega(s)$, which we can sample actions from, as a reasonable approximation to $\operatorname{argmax}_a P^G_{\mathcal{B}}(a|s)$.

For our generative model we use a conditional variational auto-encoder (VAE) (Kingma & Welling, 2013; Sohn et al., 2015), which models the distribution by transforming an underlying latent space[1]. The generative model G_ω, alongside the value function Q_θ, can be used as a policy by sampling n actions from G_ω and selecting the highest valued action according to the value estimate Q_θ. To increase the diversity of seen actions, we introduce a perturbation model $\xi_\phi(s,a,\Phi)$, which outputs an adjustment to an action a in the range $[-\Phi,\Phi]$. This enables access to actions in a constrained region, without having to sample from the generative model a prohibitive number of times. This results in the policy π:

$$\begin{aligned}\pi(s) = &\operatorname*{argmax}_{a_i+\xi_\phi(s,a_i,\Phi)} Q_\theta(s, a_i + \xi_\phi(s,a_i,\Phi)),\\ &\{a_i \sim G_\omega(s)\}_{i=1}^n.\end{aligned} \tag{11}$$

The choice of n and Φ creates a trade-off between an imitation learning and reinforcement learning algorithm. If $\Phi = 0$, and the number of sampled actions $n = 1$, then the policy resembles behavioral cloning and as $\Phi \to a_{\text{max}} - a_{\text{min}}$ and $n \to \infty$, then the algorithm approaches Q-learning, as the policy begins to greedily maximize the value function over the entire action space.

The perturbation model ξ_ϕ can be trained to maximize $Q_\theta(s,a)$ through the deterministic policy gradient algorithm (Silver et al., 2014) by sampling $a \sim G_\omega(s)$:

$$\phi \leftarrow \operatorname*{argmax}_\phi \sum_{(s,a)\in\mathcal{B}} Q_\theta(s, a + \xi_\phi(s,a,\Phi)). \tag{12}$$

To penalize uncertainty over future states, we modify Clipped Double Q-learning (Fujimoto et al., 2018), which estimates the value by taking the minimum between two Q-networks $\{Q_{\theta_1}, Q_{\theta_2}\}$. Although originally used as a countermeasure to overestimation bias (Thrun & Schwartz, 1993; Van Hasselt, 2010), the minimum operator also penalizes high variance estimates in regions of uncertainty, and pushes the policy to favor actions which lead to states contained in the batch. In particular, we take a convex combination of the two values, with a higher weight on the minimum, to form a learning target which is used by both Q-networks:

$$r + \gamma \max_{a_i}\left[\lambda \min_{j=1,2} Q_{\theta'_j}(s', a_i) + (1-\lambda)\max_{j=1,2} Q_{\theta'_j}(s', a_i)\right] \tag{13}$$

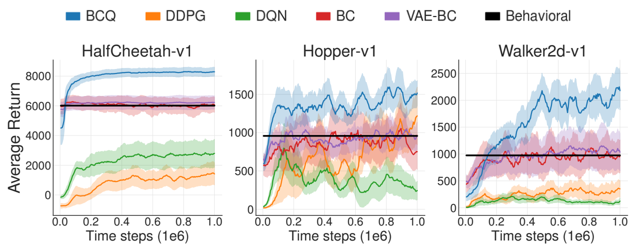
where a_i corresponds to the perturbed actions, sampled from the generative model. If we set $\lambda = 1$, this update corresponds to Clipped Double Q-learning. We use this weighted minimum as the constrained updates produces less overestimation bias than a purely greedy policy update, and enables control over how heavily uncertainty at future time steps is penalized through the choice of λ.

This forms Batch-Constrained deep Q-learning (BCQ), which maintains four parametrized networks: a generative model $G_\omega(s)$, a perturbation model $\xi_\phi(s,a)$, and two Q-networks $Q_{\theta_1}(s,a), Q_{\theta_2}(s,a)$. We summarize BCQ in Algorithm 1. In the following section, we demonstrate BCQ results in stable value learning and a strong performance in the batch setting. Furthermore, we find that only a single choice of hyper-parameters is necessary for a wide range of tasks and environments.

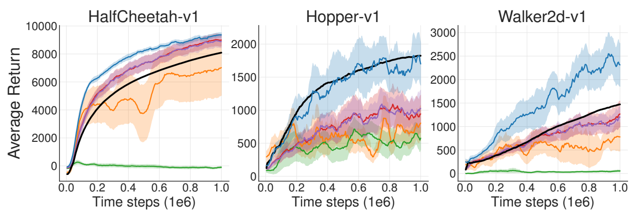
Algorithm 1 BCQ

```
Input: Batch $\mathcal{B}$, horizon $T$, target network update rate $\tau$, mini-batch size $N$, max perturbation $\Phi$, number of sampled actions $n$, minimum weighting $\lambda$.
Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, perturbation network $\xi_\phi$, and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, with random parameters $\theta_1$, $\theta_2$, $\phi$, $\omega$, and target networks $Q_{\theta'_1}, Q_{\theta'_2}$, $\xi_{\phi'}$ with $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$.
for $t = 1$ to $T$ do
    Sample mini-batch of $N$ transitions $(s,a,r,s')$ from $\mathcal{B}$
    $\mu, \sigma = E_{\omega_1}(s,a), \quad \tilde{a} = D_{\omega_2}(s,z), \quad z \sim \mathcal{N}(\mu,\sigma)$
    $\omega \leftarrow \operatorname{argmin}_\omega \sum (a-\tilde{a})^2 + D_{\text{KL}}(\mathcal{N}(\mu,\sigma)||\mathcal{N}(0,1))$
    Sample $n$ actions: $\{a_i \sim G_\omega(s')\}_{i=1}^n$
    Perturb each action: $\{a_i = a_i + \xi_\phi(s', a_i, \Phi)\}_{i=1}^n$
    Set value target $y$ (Eqn. 13)
    $\theta \leftarrow \operatorname{argmin}_\theta \sum (y - Q_\theta(s,a))^2$
    $\phi \leftarrow \operatorname{argmax}_\phi \sum Q_{\theta_1}(s, a + \xi_\phi(s,a,\Phi)), a \sim G_\omega(s)$
    Update target networks: $\theta'_i \leftarrow \tau\theta + (1-\tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
end for
```

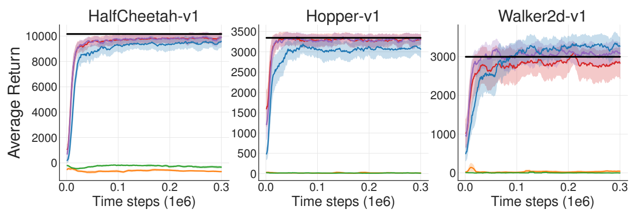
[1]See the Supplementary Material for an introduction to VAEs.



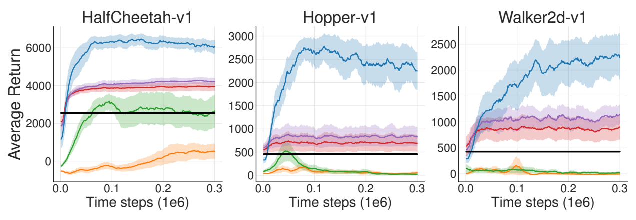
(a) Final buffer performance



(b) Concurrent performance



(c) Imitation performance



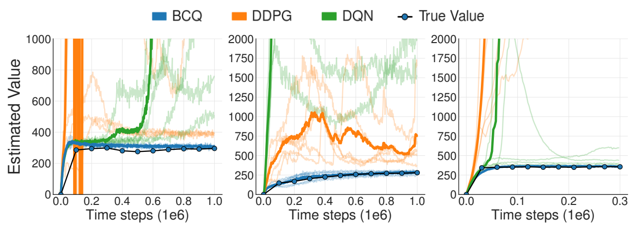
(d) Imperfect demonstrations performance

Figure 2. We evaluate BCQ and several baselines on the experiments from Section 3.1, as well as the imperfect demonstrations task. The shaded area represents half a standard deviation. The bold black line measures the average return of episodes contained in the batch. Only BCQ matches or outperforms the performance of the behavioral policy in all tasks.

5. Experiments

To evaluate the effectiveness of Batch-Constrained deep Q-learning (BCQ) in a high-dimensional setting, we focus on MuJoCo environments in OpenAI gym (Todorov et al., 2012; Brockman et al., 2016). For reproducibility, we make no modifications to the original environments or reward functions. We compare our method with DDPG (Lillicrap et al., 2015), DQN (Mnih et al., 2015) using an independently discretized action space, a feed-forward behavioral cloning method (BC), and a variant with a VAE (VAE-BC), using $G_\omega(s)$ from BCQ. Exact implementation and experimental details are provided in the Supplementary Material.

We evaluate each method following the three experiments defined in Section 3.1. In *final buffer* the off-policy agents



(a) Final Buffer (b) Concurrent (c) Imitation

Figure 3. We examine the value estimates of BCQ, along with DDPG and DQN on the experiments from Section 3.1 in the Hopper-v1 environment. Each individual trial is plotted, with the mean in bold. An estimate of the true value of BCQ, evaluated by Monte Carlo returns, is marked by a dotted line. Unlike the state of the art baselines, BCQ exhibits a highly stable value function in each task. Graphs for the other environments and imperfect demonstrations task can be found in the Supplementary Material.

learn from the final replay buffer gathered by training a DDPG agent over a million time steps. In *concurrent* the off-policy agents learn concurrently, with the same replay buffer, as the behavioral DDPG policy, and in *imitation*, the agents learn from a dataset collected by an expert policy. Additionally, to study the robustness of BCQ to noisy and multi-modal data, we include an *imperfect demonstrations* task, in which the agents are trained with a batch of 100k transitions collected by an expert policy, with two sources of noise. The behavioral policy selects actions randomly with probability 0.3 and with high exploratory noise $\mathcal{N}(0, 0.3)$ added to the remaining actions. The experimental results for these tasks are reported in Figure 2. Furthermore, the estimated values of BCQ, DDPG and DQN, and the true value of BCQ are displayed in Figure 3.

Our approach, BCQ, is the only algorithm which succeeds at all tasks, matching or outperforming the behavioral policy in each instance, and outperforming all other agents, besides in the imitation learning task where behavioral cloning unsurprisingly performs the best. These results demonstrate that our algorithm can be used as a single approach for both imitation learning and off-policy reinforcement learning, with a single set of fixed hyper-parameters. Furthermore, unlike the deep reinforcement learning algorithms, DDPG and DQN, BCQ exhibits a highly stable value function in the presence of off-policy samples, suggesting extrapolation error has been successfully mitigated through the batch-constraint. In the imperfect demonstrations task, we find that both deep reinforcement learning and imitation learning algorithms perform poorly. BCQ, however, is able to strongly outperform the noisy demonstrator, disentangling poor and expert actions. Furthermore, compared to current deep reinforcement learning algorithms, which can require millions of time steps (Duan et al., 2016; Henderson et al., 2017), BCQ attains a high performance in remarkably few iterations. This suggests our approach effectively leverages expert transitions, even in the presence of noise.

6. Related Work

Batch Reinforcement Learning. While batch reinforcement learning algorithms have been shown to be convergent with non-parametric function approximators such as averagers (Gordon, 1995) and kernel methods (Ormoneit & Sen, 2002), they make no guarantees on the quality of the policy without infinite data. Other batch algorithms, such as fitted Q-iteration, have used other function approximators, including decision trees (Ernst et al., 2005) and neural networks (Riedmiller, 2005), but come without convergence guarantees. Unlike many previous approaches to off-policy policy evaluation (Peshkin & Shelton, 2002; Thomas et al., 2015; Liu et al., 2018), our work focuses on constraining the policy to a subset of policies which can be adequately evaluated, rather than the process of evaluation itself. Additionally, off-policy algorithms which rely on importance sampling (Precup et al., 2001; Jiang & Li, 2016; Munos et al., 2016) may not be applicable in a batch setting, requiring access to the action probabilities under the behavioral policy, and scale poorly to multi-dimensional action spaces. Reinforcement learning with a replay buffer (Lin, 1992) can be considered a form of batch reinforcement learning, and is a standard tool for off-policy deep reinforcement learning algorithms (Mnih et al., 2015). It has been observed that a large replay buffer can be detrimental to performance (de Bruin et al., 2015; Zhang & Sutton, 2017) and the diversity of states in the buffer is an important factor for performance (de Bruin et al., 2016). Isele & Cosgun (2018) observed the performance of an agent was strongest when the distribution of data in the replay buffer matched the test distribution. These results defend the notion that extrapolation error is an important factor in the performance off-policy reinforcement learning.

Imitation Learning. Imitation learning and its variants are well studied problems (Schaal, 1999; Argall et al., 2009; Hussein et al., 2017). Imitation has been combined with reinforcement learning, via learning from demonstrations methods (Kim et al., 2013; Piot et al., 2014; Chemali & Lazaric, 2015), with deep reinforcement learning extensions (Hester et al., 2017; Večerík et al., 2017), and modified policy gradient approaches (Ho et al., 2016; Sun et al., 2017; Cheng et al., 2018; Sun et al., 2018). While effective, these interactive methods are inadequate for batch reinforcement learning as they require either an explicit distinction between expert and non-expert data, further on-policy data collection or access to an oracle. Research in imitation, and inverse reinforcement learning, with robustness to noise is an emerging area (Evans, 2016; Nair et al., 2018), but relies on some form of expert data. Gao et al. (2018) introduced an imitation learning algorithm which learned from imperfect demonstrations, by favoring seen actions, but is limited to discrete actions. Our work also connects to residual policy learning (Johannink et al., 2018; Silver et al., 2018), where the initial policy is the generative model, rather than an expert or feedback controller.

Uncertainty in Reinforcement Learning. Uncertainty estimates in deep reinforcement learning have generally been used to encourage exploration (Dearden et al., 1998; Strehl & Littman, 2008; O'Donoghue et al., 2018; Azizzadenesheli et al., 2018). Other methods have examined approximating the Bayesian posterior of the value function (Osband et al., 2016; 2018; Touati et al., 2018), again using the variance to encourage exploration to unseen regions of the state space. In model-based reinforcement learning, uncertainty has been used for exploration, but also for the opposite effect–to push the policy towards regions of certainty in the model. This is used to combat the well-known problems with compounding model errors, and is present in policy search methods (Deisenroth & Rasmussen, 2011; Gal et al., 2016; Higuera et al., 2018; Xu et al., 2018), or combined with trajectory optimization (Chua et al., 2018) or value-based methods (Buckman et al., 2018). Our work connects to policy methods with conservative updates (Kakade & Langford, 2002), such as trust region (Schulman et al., 2015; Achiam et al., 2017; Pham et al., 2018) and information-theoretic methods (Peters & Mülling, 2010; Van Hoof et al., 2017), which aim to keep the updated policy similar to the previous policy. These methods avoid explicit uncertainty estimates, and rather force policy updates into a constrained range before collecting new data, limiting errors introduced by large changes in the policy. Similarly, our approach can be thought of as an off-policy variant, where the policy aims to be kept close, in output space, to any combination of the previous policies which performed data collection.

7. Conclusion

In this work, we demonstrate a critical problem in off-policy reinforcement learning with finite data, where the value target introduces error by including an estimate of unseen state-action pairs. This phenomenon, which we denote *extrapolation error*, has important implications for off-policy and batch reinforcement learning, as it is generally implausible to have complete state-action coverage in any practical setting. We present batch-constrained reinforcement learning–acting close to on-policy with respect to the available data, as an answer to extrapolation error. When extended to a deep reinforcement learning setting, our algorithm, Batch-Constrained deep Q-learning (BCQ), is the first continuous control algorithm capable of learning from arbitrary batch data, without exploration. Due to the importance of batch reinforcement learning for practical applications, we believe BCQ will be a strong foothold for future algorithms to build on, while furthering our understanding of the systematic risks in Q-learning (Thrun & Schwartz, 1993; Lu et al., 2018).

References

Achiam, J., Held, D., Tamar, A., and Abbeel, P. Constrained policy optimization. In *International Conference on Machine Learning*, pp. 22–31, 2017.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

Azizzadenesheli, K., Brunskill, E., and Anandkumar, A. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

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

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pp. 8234–8244, 2018.

Chemali, J. and Lazaric, A. Direct policy iteration with demonstrations. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Cheng, C.-A., Yan, X., Wagener, N., and Boots, B. Fast policy learning through imitation and reinforcement. *arXiv preprint arXiv:1805.10413*, 2018.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems 31*, pp. 4759–4770, 2018.

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

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*, 2015.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. Improved deep reinforcement learning for robotics through distribution-based experience retention. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3947–3952. IEEE, 2016.

Dearden, R., Friedman, N., and Russell, S. Bayesian q-learning. In *AAAI/IAAI*, pp. 761–768, 1998.

Deisenroth, M. and Rasmussen, C. E. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, pp. 465–472, 2011.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

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

Evans, O. Learning the preferences of ignorant, inconsistent agents. In *AAAI*, pp. 323–329, 2016.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, volume 80, pp. 1587–1596. PMLR, 2018.

Gal, Y., McAllister, R., and Rasmussen, C. E. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*, 2016.

Gao, Y., Lin, J., Yu, F., Levine, S., and Darrell, T. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

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

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep Reinforcement Learning that Matters. *arXiv preprint arXiv:1709.06560*, 2017.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., et al. Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017.

Higuera, J. C. G., Meger, D., and Dudek, G. Synthesizing neural network controllers with probabilistic model based reinforcement learning. *arXiv preprint arXiv:1803.02291*, 2018.

Ho, J., Gupta, J., and Ermon, S. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*, pp. 2760–2769, 2016.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

Isele, D. and Cosgun, A. Selective experience replay for lifelong learning. *arXiv preprint arXiv:1802.10269*, 2018.

Jaksch, T., Ortner, R., and Auer, P. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

Jiang, N. and Li, L. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pp. 652–661, 2016.

Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. Residual reinforcement learning for robot control. *arXiv preprint arXiv:1812.03201*, 2018.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pp. 267–274, 2002.

Kim, B., Farahmand, A.-m., Pineau, J., and Precup, D. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, pp. 2859–2867, 2013.

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

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Konda, V. R. and Tsitsiklis, J. N. On actor-critic algorithms. *SIAM journal on Control and Optimization*, 42(4):1143–1166, 2003.

Lai, T. L. and Robbins, H. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1): 4–22, 1985.

Lange, S., Gabel, T., and Riedmiller, M. Batch reinforcement learning. In *Reinforcement learning*, pp. 45–73. Springer, 2012.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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

Liu, Y., Gottesman, O., Raghu, A., Komorowski, M., Faisal, A. A., Doshi-Velez, F., and Brunskill, E. Representation balancing mdps for off-policy policy evaluation. In *Advances in Neural Information Processing Systems*, pp. 2644–2653, 2018.

Lu, T., Schuurmans, D., and Boutilier, C. Non-delusional q-learning and value-iteration. In *Advances in Neural Information Processing Systems*, pp. 9971–9981, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Melo, F. S. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. The uncertainty Bellman equation and exploration. In *International Conference on Machine Learning*, volume 80, pp. 3839–3848. PMLR, 2018.

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

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pp. 4026–4034, 2016.

Osband, I., Aslanides, J., and Cassirer, A. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems 31*, pp. 8626–8638, 2018.

Peshkin, L. and Shelton, C. R. Learning from scarce experience. In *International Conference on Machine Learning*, pp. 498–505, 2002.

Peters, J. and Mülling, K. Relative entropy policy search. In *AAAI*, pp. 1607–1612, 2010.

Pham, T.-H., De Magistris, G., Agravante, D. J., Chaudhury, S., Munawar, A., and Tachibana, R. Constrained exploration and recovery from experience shaping. *arXiv preprint arXiv:1809.08925*, 2018.

Piot, B., Geist, M., and Pietquin, O. Boosted bellman residual minimization handling expert demonstrations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 549–564. Springer, 2014.

Precup, D., Sutton, R. S., and Dasgupta, S. Off-policy temporal-difference learning with function approximation. In *International Conference on Machine Learning*, pp. 417–424, 2001.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

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

Schaal, S. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pp. 387–395, 2014.

Silver, T., Allen, K., Tenenbaum, J., and Kaelbling, L. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pp. 3483–3491, 2015.

Strehl, A. L. and Littman, M. L. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., and Bagnell, J. A. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International Conference on Machine Learning*, pp. 3309–3318, 2017.

Sun, W., Bagnell, J. A., and Boots, B. Truncated horizon policy search: Combining reinforcement learning & imitation learning. *arXiv preprint arXiv:1805.11240*, 2018.

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

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

Thomas, P., Theocharous, G., and Ghavamzadeh, M. High confidence policy improvement. In *International Conference on Machine Learning*, pp. 2380–2388, 2015.

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

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033. IEEE, 2012.

Touati, A., Satija, H., Romoff, J., Pineau, J., and Vincent, P. Randomized value functions via multiplicative normalizing flows. *arXiv preprint arXiv:1806.02315*, 2018.

Van Hasselt, H. Double q-learning. In *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016.

Van Hoof, H., Neumann, G., and Peters, J. Non-parametric policy search with limited information loss. *The Journal of Machine Learning Research*, 18(1):2472–2517, 2017.

Večerík, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Watkins, C. J. C. H. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. Algorithmic framework for model-based reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

Zhang, S. and Sutton, R. S. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

Off-Policy Deep Reinforcement Learning without Exploration: Supplementary Material

A. Missing Proofs

A.1. Proofs and Details from Section 4.1

Definition 1. We define a coherent batch $\mathcal{B}$ as a batch such that if $(s, a, s') \in \mathcal{B}$ then $s' \in \mathcal{B}$ unless s' is a terminal state.

Definition 2. We define $\epsilon_{\text{MDP}}(s, a) = Q^\pi(s, a) - Q^\pi_{\mathcal{B}}(s, a)$ as the error between the true value of a policy π in the MDP M and the value of π when learned with a batch $\mathcal{B}$.

Definition 3. For simplicity in notation, we denote

$$\epsilon^\pi_{\text{MDP}} = \sum_s \mu_\pi(s) \sum_a \pi(a|s)|\epsilon_{\text{MDP}}(s, a)|. \tag{14}$$

To evaluate a policy π exactly at relevant state-action pairs, only $\epsilon^\pi_{\text{MDP}} = 0$ is required.

Definition 4. We define the optimal batch-constrained policy $\pi^* \in \Pi_{\mathcal{B}}$ such that $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$ for all $\pi \in \Pi_{\mathcal{B}}$ and $(s, a) \in \mathcal{B}$.

Algorithm 1. Batch-Constrained Q-learning (BCQL) maintains a tabular value function $Q(s, a)$ for each possible state-action pair (s, a). A transition tuple (s, a, r, s') is sampled from the batch $\mathcal{B}$ with uniform probability and the following update rule is applied, with learning rate α:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a' \text{s.t.} (s',a') \in \mathcal{B}} Q(s', a')). \tag{15}$$

Theorem 1. *Performing Q-learning by sampling from a batch $\mathcal{B}$ converges to the optimal value function under the MDP $M_{\mathcal{B}}$.*

Proof. Again, the MDP $M_{\mathcal{B}}$ is defined by the same action and state space as M, with an additional terminal state s_{init}. $M_{\mathcal{B}}$ has transition probabilities $p_{\mathcal{B}}(s'|s, a) = \frac{N(s,a,s')}{\sum_{\tilde{s}} N(s,a,\tilde{s})}$, where $N(s, a, s')$ is the number of times the tuple (s, a, s') is observed in $\mathcal{B}$. If $\sum_{\tilde{s}} N(s, a, \tilde{s}) = 0$, then $p_{\mathcal{B}}(s_{\text{init}}|s, a) = 1$, where $r(s_{\text{init}}, s, a)$ is to the initialized value of $Q(s, a)$.

For any given MDP Q-learning converges to the optimal value function given infinite state-action visitation and some standard assumptions (see Section A.2). Now note that sampling under a batch $\mathcal{B}$ with uniform probability satisfies the infinite state-action visitation assumptions of the MDP $M_{\mathcal{B}}$, where given (s, a), the probability of sampling (s, a, s') corresponds to $p(s'|s, a) = \frac{N(s,a,s')}{\sum_{\tilde{s}} N(s,a,\tilde{s})}$ in the limit. We remark that for $(s, a) \notin \mathcal{B}$, $Q(s, a)$ will never be updated, and will return the initialized value, which corresponds to the terminal transition s_{init}. It follows that sampling from $\mathcal{B}$ is equivalent to sampling from the MDP $M_{\mathcal{B}}$, and Q-learning converges to the optimal value function under $M_{\mathcal{B}}$.

Remark 1. *For any policy π and state-action pair (s, a), the error term $\epsilon_{MDP}(s, a)$ satisfies the following Bellman-like equation:*

$$\begin{aligned}\epsilon_{\text{MDP}}(s, a) = & \sum_{s'} (p_M(s'|s, a) - p_{\mathcal{B}}(s'|s, a)) \left(r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') \left(Q^\pi_{\mathcal{B}}(s', a') \right) \right) \\ & + p_M(s'|s, a)\gamma \sum_{a'} \pi(a'|s')\epsilon_{\text{MDP}}(s', a').\end{aligned} \tag{16}$$

Proof. Proof follows by expanding each Q, rearranging terms and then simplifying the expression.

$$\begin{aligned}
\epsilon_{\text{MDP}}(s,a) &= Q^\pi(s,a) - Q^\pi_{\mathcal{B}}(s,a) \\
&= \sum_{s'} p_M(s'|s,a)\left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')\right) - Q^\pi_{\mathcal{B}}(s,a) \\
&= \sum_{s'} p_M(s'|s,a)\left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')\right) \\
&\quad - \left(\sum_{s'} p_{\mathcal{B}}(s'|s,a)\left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a')\right)\right) \\
&= \sum_{s'} \left(p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a)\right) r(s,a,s') + p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s')\left(Q^\pi_{\mathcal{B}}(s',a') + \epsilon_{\text{MDP}}(s',a')\right) \\
&\quad - p_{\mathcal{B}}(s'|s,a)\gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a') \\
&= \sum_{s'} \left(p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a)\right) r(s,a,s') + p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s')\left(Q^\pi_{\mathcal{B}}(s',a') + \epsilon_{\text{MDP}}(s',a')\right) \\
&\quad + p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s')\left(\epsilon_{\text{MDP}}(s',a') - \epsilon_{\text{MDP}}(s',a')\right) - p_{\mathcal{B}}(s'|s,a)\gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a') \\
&= \sum_{s'} \left(p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a)\right)\left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a')\right) \\
&\quad + p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s') \epsilon_{\text{MDP}}(s',a')
\end{aligned} \tag{17}$$

Lemma 1. *For all reward functions, $\epsilon^\pi_{MDP} = 0$ if and only if $p_{\mathcal{B}}(s'|s,a) = p_M(s'|s,a)$ for all $s' \in \mathcal{S}$ and (s,a) such that $\mu_\pi(s) > 0$ and $\pi(a|s) > 0$.*

Proof. From Remark 1, we note that the form of $\epsilon_{\text{MDP}}(s,a)$, since no assumptions can be made on the reward function and therefore the expression $r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a')$, we have that $\epsilon_{\text{MDP}}(s,a) = 0$ if and only if $p_{\mathcal{B}}(s'|s,a) = p_M(s'|s,a)$ for all $s' \in \mathcal{S}$ and $p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s') \epsilon_{\text{MDP}}(s',a') = 0$.

($\Rightarrow$) Now we note that if $\epsilon_{\text{MDP}}(s,a) = 0$ then $p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s') \epsilon_{\text{MDP}}(s',a') = 0$ by the relationship defined by Remark 1 and the condition on the reward function. It follows that we must have $p_{\mathcal{B}}(s'|s,a) = p_M(s'|s,a)$ for all $s' \in \mathcal{S}$.

($\Leftarrow$) If we have $\sum_{s'} |p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a)| = 0$ for all (s,a) such that $\mu_\pi(s) > 0$ and $\pi(a|s) > 0$, then for any (s,a) under the given conditions, we have $\epsilon(s,a) = \sum_{s'} p_M(s'|s,a)\gamma \sum_{a'} \pi(a'|s') \epsilon(s',a')$. Recursively expanding the ϵ term, we arrive at $\epsilon(s,a) = 0 + \gamma 0 + \gamma^2 0 + ... = 0$.

Theorem 2. *For a deterministic MDP and all reward functions, $\epsilon^\pi_{MDP} = 0$ if and only if the policy π is batch-constrained. Furthermore, if $\mathcal{B}$ is coherent, then such a policy must exist if the start state $s_0 \in \mathcal{B}$.*

Proof. The first part of the Theorem follows from Lemma 1, noting that for a deterministic policy π, if $(s,a) \in \mathcal{B}$ then we must have $p_{\mathcal{B}}(s'|s,a) = p_M(s'|s,a)$ for all $s' \in \mathcal{S}$.

We can construct the batch-constrained policy by selecting a in the state $s \in \mathcal{B}$, such that $(s,a) \in \mathcal{B}$. Since the MDP is deterministic and the batch is coherent, when starting from s_0, we must be able to follow at least one trajectory until termination.

Theorem 3. *Given the Robbins-Monro stochastic convergence conditions on the learning rate α, and standard sampling requirements from the environment, BCQL converges to the optimal value function Q^*.*

Proof. Follows from proof of convergence of Q-learning (see Section A.2), noting the batch-constraint is non-restrictive with a batch which contains all possible transitions.

Theorem 4. *Given a deterministic MDP and coherent batch $\mathcal{B}$, along with the Robbins-Monro stochastic convergence conditions on the learning rate α and standard sampling requirements on the batch $\mathcal{B}$, BCQL converges to $Q^{\pi}_{\mathcal{B}}(s,a)$ where $\pi^*(s) = \text{argmax}_{a \text{ s.t.} (s,a) \in \mathcal{B}} Q^{\pi}_{\mathcal{B}}(s,a)$ is the optimal batch-constrained policy.*

Proof. Results follows from Theorem 1, which states Q-learning learns the optimal value for the MDP $M_{\mathcal{B}}$ for state-action pairs in (s,a). However, for a deterministic MDP $M_{\mathcal{B}}$ corresponds to the true MDP in all seen state-action pairs. Noting that batch-constrained policies operate only on state-action pairs where $M_{\mathcal{B}}$ corresponds to the true MDP, it follows that π^* will be the optimal batch-constrained policy from the optimality of Q-learning.

A.2. Sketch of the Proof of Convergence of Q-Learning

The proof of convergence of Q-learning relies large on the following lemma (Singh et al., 2000):

Lemma 2. *Consider a stochastic process $(\zeta_t, \Delta_t, F_t), t \geq 0$ where $\zeta_t, \Delta_t, F_t : X \to \mathbb{R}$ satisfy the equation:*

$$\Delta_{t+1}(x_t) = (1 - \zeta_t(x_t))\Delta_t(x_t) + \zeta_t(x_t)F_t(x_t), \tag{18}$$

where $x_t \in X$ and $t = 0, 1, 2, \ldots$. Let P_t be a sequence of increasing σ-fields such that ζ_0 and Δ_0 are P_0-measurable and ζ_t, Δ_t and F_{t-1} are P_t-measurable, $t = 1, 2, \ldots$. Assume that the following hold:

1. *The set X is finite.*
2. *$\zeta_t(x_t) \in [0,1], \sum_t \zeta_t(x_t) = \infty, \sum_t (\zeta_t(x_t))^2 < \infty$ with probability 1 and $\forall x \neq x_t : \zeta(x) = 0$.*
3. *$||\mathbb{E}\left[F_t|P_t\right]|| \leq \kappa||\Delta_t|| + c_t$ where $\kappa \in [0,1)$ and c_t converges to 0 with probability 1.*
4. *$Var[F_t(x_t)|P_t] \leq K(1 + \kappa||\Delta_t||)^2$, where K is some constant*

Where $||\cdot||$ denotes the maximum norm. Then Δ_t converges to 0 with probability 1.

Sketch of Proof of Convergence of Q-Learning. We set $\Delta_t = Q_t(s,a) - Q^*(s,a)$. Then convergence follows by satisfying the conditions of Lemma 2. Condition 1 is satisfied by the finite MDP, setting $X = \mathcal{S} \times \mathcal{A}$. Condition 2 is satisfied by the assumption of Robbins-Monro stochastic convergence conditions on the learning rate α_t, setting $\zeta_t = \alpha_t$. Condition 4 is satisfied by the bounded reward function, where $F_t(s,a) = r(s,a,s') + \gamma \max_{a'} Q(s',a') - Q^*(s,a)$, and the sequence $P_t = \{Q_0, s_0, a_0, \alpha_0, r_1, s_1, \ldots s_t, a_t\}$. Finally, Condition 3 follows from the contraction of the Bellman Operator $\mathcal{T}$, requiring infinite state-action visitation, infinite updates and $\gamma < 1$.

Additional and more complete details can be found in numerous resources (Dayan & Watkins, 1992; Singh et al., 2000; Melo, 2001).

B. Missing Graphs

B.1. Extrapolation Error in Deep Reinforcement Learning

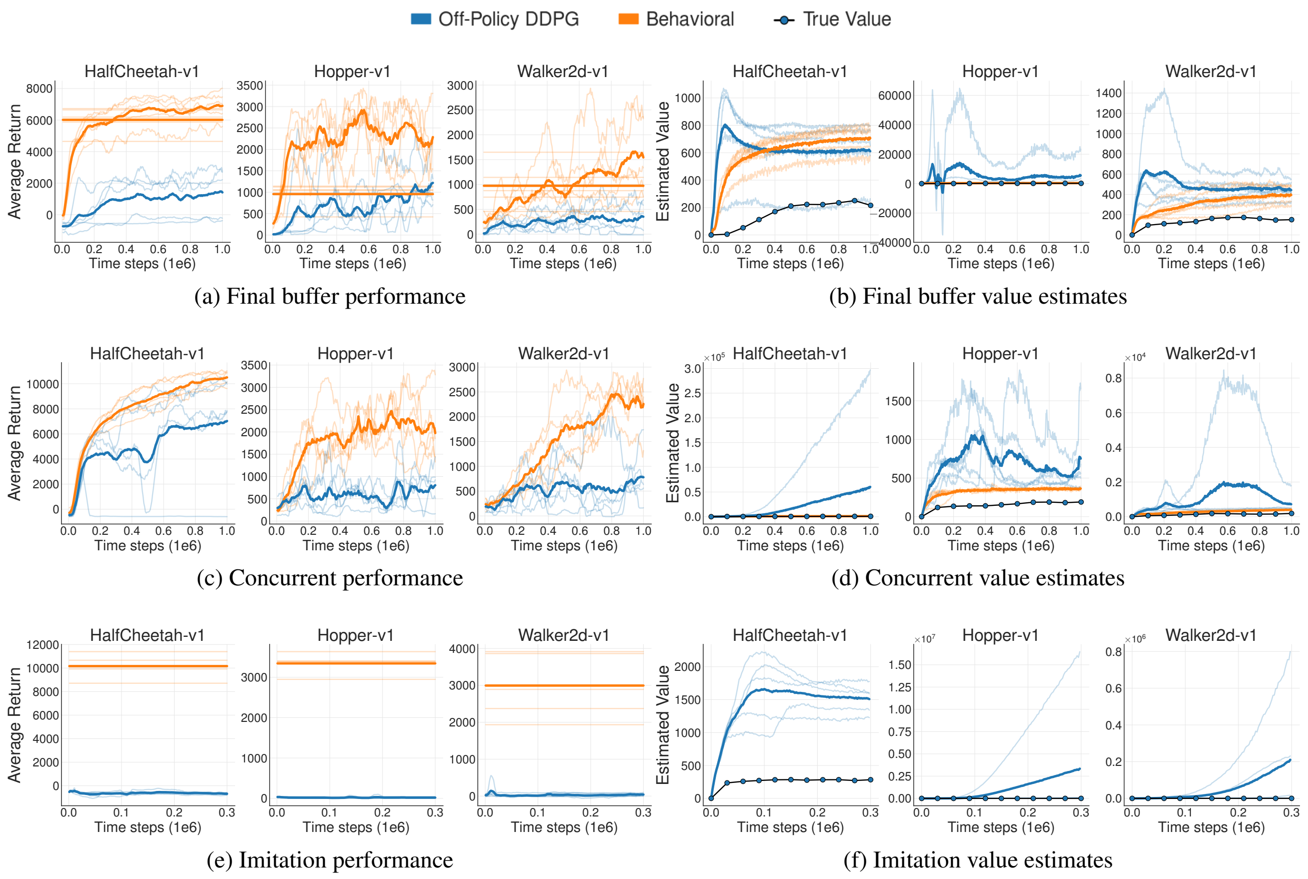
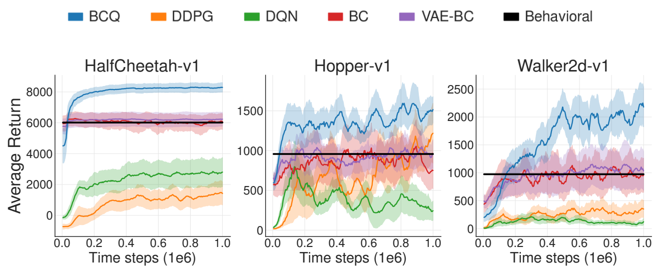
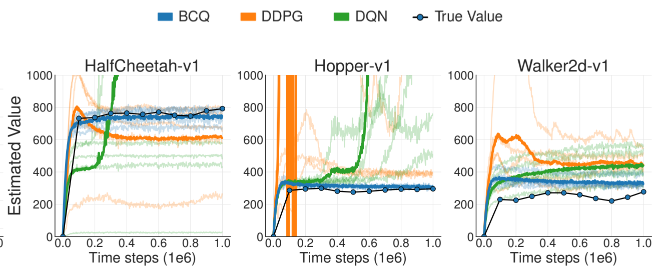


Figure 4. We examine the performance of DDPG in three batch tasks. Each individual trial is plotted with a thin line, with the mean in bold (evaluated without exploration noise). Straight lines represent the average return of episodes contained in the batch (with exploration noise). An estimate of the true value of the off-policy agent, evaluated by Monte Carlo returns, is marked by a dotted line. In the final buffer experiment, the off-policy agent learns from a large, diverse dataset, but exhibits poor learning behavior and value estimation. In the concurrent setting the agent learns alongside a behavioral agent, with access to the same data, but suffers in performance. In the imitation setting, the agent receives data from an expert policy but is unable to learn, and exhibits highly divergent value estimates.

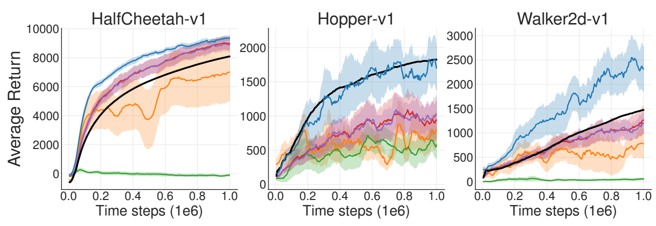
B.2. Complete Experimental Results



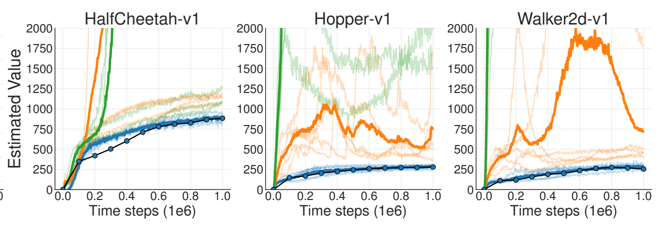
(a) Final buffer performance



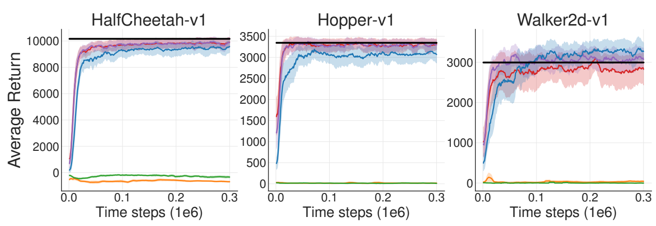
(b) Final buffer value estimates



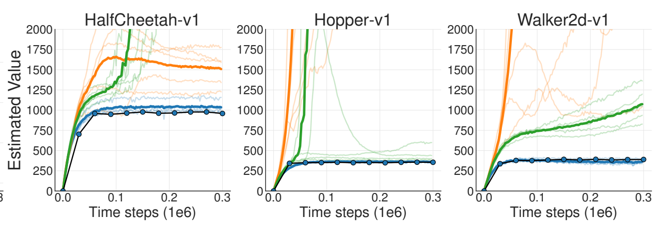
(c) Concurrent performance



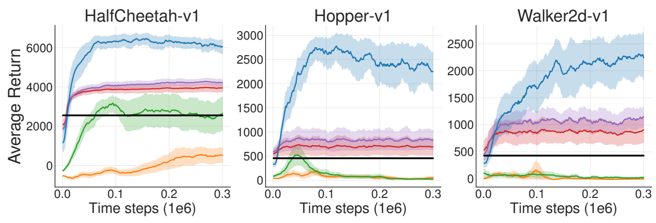
(d) Concurrent value estimates



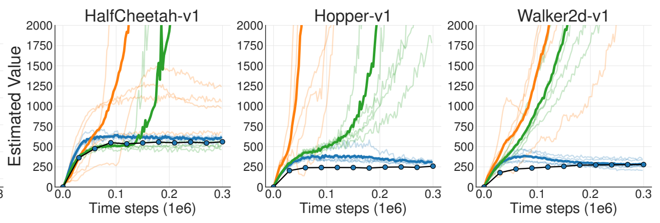
(e) Imitation performance



(f) Imitation value estimates



(g) Imperfect demonstrations performance



(h) Imperfect demonstrations value estimates

Figure 5. We evaluate BCQ and several baselines on the experiments from Section 3.1, as well as a new imperfect demonstration task. Performance is graphed on the left, and value estimates are graphed on the right. The shaded area represents half a standard deviation. The bold black line measures the average return of episodes contained in the batch. For the value estimates, each individual trial is plotted, with the mean in bold. An estimate of the true value of BCQ, evaluated by Monte Carlo returns, is marked by a dotted line. Only BCQ matches or outperforms the performance of the behavioral policy in all tasks, while exhibiting a highly stable value function in each task.

We present the complete set of results across each task and environment in Figure 5. These results show that BCQ successful mitigates extrapolation error and learns from a variety of fixed batch settings. Although BCQ with our current hyper-parameters was never found to fail, we noted with slight changes to hyper-parameters, BCQ failed periodically on the concurrent learning task in the HalfCheetah-v1 environment, exhibiting instability in the value function after $750,000$ or more iterations on some seeds. We hypothesize that this instability could occur if the generative model failed to output in-distribution actions, and could be corrected through additional training or improvements to the vanilla VAE. Interestingly, BCQ still performs well in these instances, due to the behavioral cloning-like elements in the algorithm.

C. Extrapolation Error in Kernel-Based Reinforcement Learning

This problem of extrapolation persists in traditional batch reinforcement learning algorithms, such as kernel-based reinforcement learning (KBRL) (Ormoneit & Sen, 2002). For a given batch $\mathcal{B}$ of transitions (s, a, r, s'), non-negative density function $\phi : \mathbb{R}^+ \to \mathbb{R}^+$, hyper-parameter $\tau \in \mathbb{R}$, and norm $||\cdot||$, KBRL evaluates the value of a state-action pair (s,a) as follows:

$$Q(s,a) = \sum_{(s^a_{\mathcal{B}}, a, r, s'_{\mathcal{B}}) \in \mathcal{B}} \kappa^a_\tau(s, s^a_{\mathcal{B}})[r + \gamma V(s'_{\mathcal{B}})], \tag{19}$$

$$\kappa^a_\tau(s, s^a_{\mathcal{B}}) = \frac{k_\tau(s, s^a_{\mathcal{B}})}{\sum_{\tilde{s}^a_{\mathcal{B}}} k_\tau(s, \tilde{s}^a_{\mathcal{B}})}, \qquad k_\tau(s, s^a_{\mathcal{B}}) = \phi\left(\frac{||s - s^a_{\mathcal{B}}||}{\tau}\right), \tag{20}$$

where $s^a_{\mathcal{B}} \in \mathcal{S}$ represents states corresponding to the action a for some tuple $(s_{\mathcal{B}}, a) \in \mathcal{B}$, and $V(s'_{\mathcal{B}}) = \max_{a \text{ s.t.}(s'_{\mathcal{B}}, a) \in \mathcal{B}} Q(s'_{\mathcal{B}}, a)$. At each iteration, KBRL updates the estimates of $Q(s_{\mathcal{B}}, a_{\mathcal{B}})$ for all $(s_{\mathcal{B}}, a_{\mathcal{B}}) \in \mathcal{B}$ following Equation (19), then updates $V(s'_{\mathcal{B}})$ by evaluating $Q(s'_{\mathcal{B}}, a)$ for all $s_{\mathcal{B}} \in \mathcal{B}$ and $a \in \mathcal{A}$.

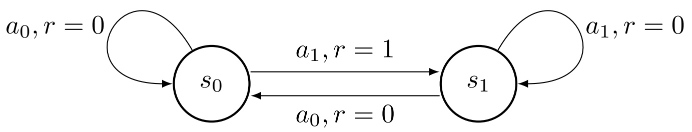


Figure 6. Toy MDP with two states s_0 and s_1, and two actions a_0 and a_1. Agent receives reward of 1 for selecting a_1 at s_0 and 0 otherwise.

Given access to the entire deterministic MDP, KBRL will provable converge to the optimal value, however when limited to only a subset, we find the value estimation susceptible to extrapolation. In Figure 6, we provide a deterministic two state, two action MDP in which KBRL fails to learn the optimal policy when provided with state-action pairs from the optimal policy. Given the batch $\{(s_0, a_1, r = 1, s_1), (s_1, a_0, r = 0, s_0)\}$, corresponding to the optimal behavior, and noting that there is only one example of each action, Equation (19) provides the following:

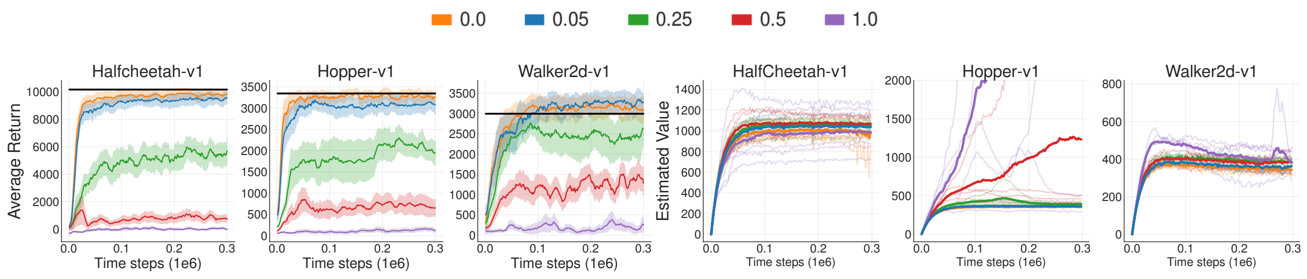
$$Q(\cdot, a_1) = 1 + \gamma V(s_1) = 1 + \gamma Q(s_1, a_0), \qquad Q(\cdot, a_0) = \gamma V(s_0) = \gamma Q(s_0, a_1). \tag{21}$$

After sufficient iterations KBRL will converge correctly to $Q(s_0, a_1) = \frac{1}{1-\gamma^2}$, $Q(s_1, a_0) = \frac{\gamma}{1-\gamma^2}$. However, when evaluating actions, KBRL erroneously extrapolates the values of each action $Q(\cdot, a_1) = \frac{1}{1-\gamma^2}$, $Q(\cdot, a_0) = \frac{\gamma}{1-\gamma^2}$, and its behavior, $\operatorname{argmax}_a Q(s, a)$, will result in the degenerate policy of continually selecting a_1. KBRL fails this example by estimating the values of unseen state-action pairs. In methods where the extrapolated estimates can be included into the learning update, such fitted Q-iteration or DQN (Ernst et al., 2005; Mnih et al., 2015), this could cause an unbounded sequence of value estimates, as demonstrated by our results in Section 3.1.

D. Additional Experiments

D.1. Ablation Study of Perturbation Model

BCQ includes a perturbation model $\xi_\theta(s, a, \Phi)$ which outputs a small residual update to the actions sampled by the generative model in the range $[-\Phi, \Phi]$. This enables the policy to select actions which may not have been sampled by the generative model. If $\Phi = a_{\text{max}} - a_{\text{min}}$, then all actions can be plausibly selected by the model, similar to standard deep reinforcement learning algorithms, such as DQN and DDPG (Mnih et al., 2015; Lillicrap et al., 2015). In Figure 7 we examine the performance and value estimates of BCQ when varying the hyper-parameter Φ, which corresponds to how much the model is able to move away from the actions sampled by the generative model.



(a) Imitation performance (b) Imitation value estimates

Figure 7. We perform an ablation study on the perturbation model of BCQ, on the imitation task from Section 3.1. Performance is graphed on the left, and value estimates are graphed on the right. The shaded area represents half a standard deviation. The bold black line measures the average return of episodes contained in the batch. For the value estimates, each individual trial is plotted, with the mean in bold.

We observe a clear drop in performance with the increase of Φ, along with an increase in instability in the value function. Given that the data is based on expert performance, this is consistent with our understanding of extrapolation error. With larger Φ the agent learns to take actions that are further away from the data in the batch after erroneously overestimating the value of suboptimal actions. This suggests the ideal value of Φ should be small enough to stay close to the generated actions, but large enough such that learning can be performed when exploratory actions are included in the dataset.

D.2. Uncertainty Estimation for Batch-Constrained Reinforcement Learning

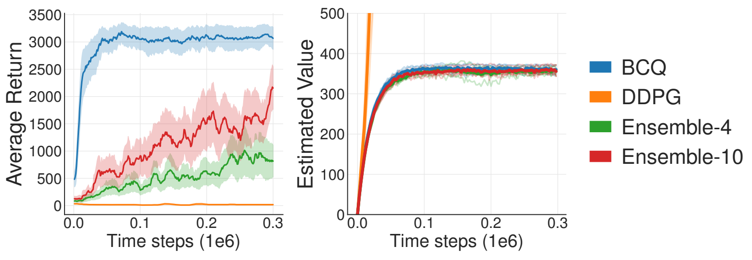
Section 4.2 proposes using generation as a method for constraining the output of the policy π to eliminate actions which are unlikely under the batch. However, a more natural approach would be through approximate uncertainty-based methods (Osband et al., 2016; Gal et al., 2016; Azizzadenesheli et al., 2018). These methods are well-known to be effective for *exploration*, however we examine their properties for the *exploitation* where we would like to avoid uncertain actions.

To measure the uncertainty of the value network, we use ensemble-based methods, with an ensemble of size 4 and 10 to mimic the models used by Buckman et al. (2018) and Osband et al. (2016) respectively. Each network is trained with separate mini-batches, following the standard deep Q-learning update with a target network. These networks use the default architecture and hyper-parameter choices as defined in Section G. The policy π_ϕ is trained to minimize the standard deviation σ across the ensemble:

$$\phi \leftarrow \underset{\phi}{\operatorname{argmin}} \sum_{(s,a)\in\mathcal{B}} \sigma\left(\{Q_{\theta_i}(s,a)\}_{i=1}^{N}\right). \tag{22}$$

If the ensembles were a perfect estimate of the uncertainty, the policy would learn to select the most certain action for a given state, minimizing the extrapolation error and effectively imitating the data in the batch.

To test these uncertainty-based methods, we examine their performance on the *imitation* task in the Hopper-v1 environment (Todorov et al., 2012; Brockman et al., 2016). In which a dataset of 1 million expert transitions are provided to the agents. Additional experimental details can be found in Section F. The performance, alongside the value estimates of the agents are displayed in Figure 8.



(a) Imitation performance (b) Imitation value estimates

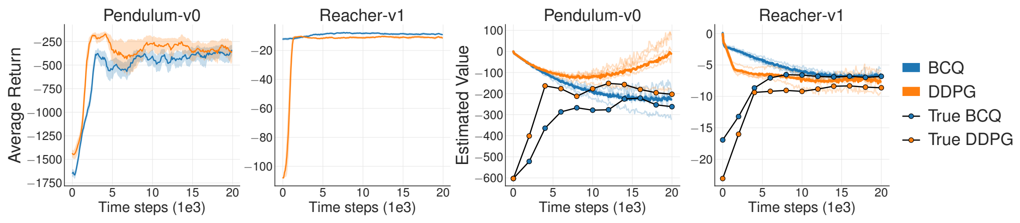
Figure 8. A comparison with uncertainty-based methods on the Hopper-v1 environment from OpenAI gym, on the imitation task. Although ensemble based methods are able to generate stable value functions (right) under these conditions, they fail to constrain the action space to the demonstrated expert actions and suffer in performance compared to our approach, BCQ (left).

We find that neither ensemble method is sufficient to constrain the action space to only the expert actions. However, the value function is stabilized, suggesting that ensembles are an effective strategy for eliminating outliers or large deviations in the value from erroneous extrapolation. Unsurprisingly, the large ensemble provides a more accurate estimate of the uncertainty. While scaling the size of the ensemble to larger values could possibly enable an effective batch-constraint, increasing the size of the ensemble induces a large computational cost. Finally, in this task, where only expert data is provided, the policy can attempt to imitate the data without consideration of the value, however in other tasks, a weighting between value and uncertainty would need to be carefully tuned. On the other hand, BCQ offers a computational cheap approach without requirements for difficult hyper-parameter tuning.

D.3. Random Behavioral Policy Study

The experiments in Section 3.1 and 5 use a learned, or partially learned, behavioral policy for data collection. This is a necessary requirement for learning meaningful behavior, as a random policy generally fails to provide sufficient coverage over the state space. However, in simple toy problems, such as the pendulum swing-up task and the reaching task with a two-joint arm from OpenAI gym (Brockman et al., 2016), a random policy can sufficiently explore the environment, enabling us to examine the properties of algorithms with entirely non-expert data.

In Figure 9, we examine the performance of our algorithm, BCQ, as well as DDPG (Lillicrap et al., 2015), on these two toy problems, when learning off-policy from a small batch of 5000 time steps, collected entirely by a random policy. We find that both BCQ and DDPG are able to learn successfully in these off-policy tasks. These results suggest that BCQ is less restrictive than imitation learning algorithms, which require expert data to learn. We also find that unlike previous environments, given the small scale of the state and action space, the random policy is able to provide sufficient coverage for DDPG to learn successfully.



(a) Random behavioral performance (b) Random behavioral value estimates

Figure 9. We evaluate BCQ and DDPG on a batch collected by a random behavioral policy. The shaded area represents half a standard deviation. Value estimates include a plot of each trial, with the mean in bold. An estimate of the true value of BCQ and DDPG, evaluated by Monte Carlo returns, is marked by a dotted line. While both BCQ and DDPG perform well, BCQ exhibits more stability in the value function for the Pendulum task, and outperforms DDPG in the Reacher task. These tasks demonstrate examples where any imitation learning would fail as the data collection procedure is entirely random.

E. Missing Background

E.1. Variational Auto-Encoder

A variational auto-encoder (VAE) (Kingma & Welling, 2013) is a generative model which aims to maximize the marginal log-likelihood $\log p(X) = \sum_{i=1}^N \log p(x_i)$ where $X = \{x_1, ..., x_N\}$, the dataset. While computing the marginal likelihood is intractable in nature, we can instead optimize the variational lower-bound:

$$\log p(X) \geq \mathbb{E}_{q(X|z)}[\log p(X|z)] + D_{\text{KL}}(q(z|X)||p(z)), \tag{23}$$

where $p(z)$ is chosen a prior, generally the multivariate normal distribution $\mathcal{N}(0, I)$. We define the posterior $q(z|X) = \mathcal{N}(z|\mu(X), \sigma^2(X)I)$ as the encoder and $p(X|z)$ as the decoder. Simply put, this means a VAE is an auto-encoder, where a given sample x is passed through the encoder to produce a random latent vector z, which is given to the decoder to reconstruct the original sample x. The VAE is trained on a reconstruction loss, and a KL-divergence term according to the distribution of the latent vectors. To perform gradient descent on the variational lower bound we can use the re-parametrization trick (Kingma & Welling, 2013; Rezende et al., 2014):

$$\mathbb{E}_{z\sim\mathcal{N}(\mu,\sigma)}[f(z)] = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[f(\mu + \sigma\epsilon)]. \tag{24}$$

This formulation allows for back-propagation through stochastic nodes, by noting μ and σ can be represented by deterministic functions. During inference, random values of z are sampled from the multivariate normal and passed through the decoder to produce samples x.

F. Experimental Details

Each environment is run for 1 million time steps, unless stated otherwise, with evaluations every 5000 time steps, where an evaluation measures the average reward from 10 episodes with no exploration noise. Our results are reported over 5 random seeds of the behavioral policy, OpenAI Gym simulator and network initialization. Value estimates are averaged over mini-batches of 100 and sampled every 2500 iterations. The *true* value is estimated by sampling 100 state-action pairs from the buffer replay and computing the discounted return by running the episode until completion while following the current policy.

Each agent is trained after each episode by applying one training iteration per each time step in the episode. The agent is trained with transition tuples (s, a, r, s') sampled from an experience replay that is defined by each experiment. We define four possible experiments. Unless stated otherwise, default implementation settings, as defined in Section G, are used.

Batch 1 (Final buffer). We train a DDPG (Lillicrap et al., 2015) agent for 1 million time steps, adding large amounts of Gaussian noise ($\mathcal{N}(0, 0.5)$) to induce exploration, and store all experienced transitions in a buffer replay. This training procedure creates a buffer replay with a diverse set of states and actions. A second, randomly initialized agent is trained using the 1 million stored transitions.

Batch 2 (Concurrent learning). We simultaneously train two agents for 1 million time steps, the first DDPG agent, performs data collection and each transition is stored in a buffer replay which both agents learn from. This means the behavioral agent learns from the standard training regime for most off-policy deep reinforcement learning algorithms, and the second agent is learning off-policy, as the data is collected without direct relationship to its current policy. Batch 2 differs from Batch 1 as the agents are trained with the same version of the buffer, while in Batch 1 the agent learns from the final buffer replay after the behavioral agent has finished training.

Batch 3 (Imitation). A DDPG agent is trained for 1 million time steps. The trained agent then acts as an expert policy, and is used to collect a dataset of 1 million transitions. This dataset is used to train a second, randomly initialized agent. In particular, we train DDPG across 15 seeds, and select the 5 top performing seeds as the expert policies.

Batch 4 (Imperfect demonstrations). The expert policies from Batch 3 are used to collect a dataset of 100k transitions, while selecting actions randomly with probability 0.3 and adding Gaussian noise $\mathcal{N}(0, 0.3)$ to the remaining actions. This dataset is used to train a second, randomly initialized agent.

G. Implementation Details

Across all methods and experiments, for fair comparison, each network generally uses the same hyper-parameters and architecture, which are defined in Table 1 and Figure 10 respectively. The value functions follow the standard practice (Mnih et al., 2015) in which the Bellman update differs for terminal transitions. When the episode ends by reaching some terminal state, the value is set to 0 in the learning target y:

$$y = \begin{cases} r & \text{if terminal } s' \\ r + \gamma Q_{\theta'}(s', a') & \text{else} \end{cases} \tag{25}$$

Where the termination signal from time-limited environments is ignored, thus we only consider a state s_t terminal if $t <$ `max horizon`.

Table 1. Default hyper-parameters.

Hyper-parameter	Value
Optimizer	Adam (Kingma & Ba, 2014)
Learning Rate	10^{-3}
Batch Size	100
Normalized Observations	False
Gradient Clipping	False
Discount Factor	0.99
Target Update Rate (τ)	0.005
Exploration Policy	$\mathcal{N}(0, 0.1)$

```
(input dimension, 400)
ReLU
(400, 300)
RelU
(300, output dimension)
```

Figure 10. Default network architecture, based on DDPG (Lillicrap et al., 2015). All actor networks are followed by a `tanh` $\cdot$ `max action size`

BCQ. BCQ uses four main networks: a perturbation model $\xi_\phi(s,a)$, a state-conditioned VAE $G_\omega(s)$ and a pair of value networks $Q_{\theta_1}(s,a), Q_{\theta_2}(s,a)$. Along with three corresponding target networks $\xi_{\phi'}(s,a), Q_{\theta'_1}(s,a), Q_{\theta'_2}(s,a)$. Each network, other than the VAE follows the default architecture (Figure 10) and the default hyper-parameters (Table 1). For $\xi_\phi(s,a,\Phi)$, the constraint Φ is implemented through a tanh activation multiplied by $I \cdot \Phi$ following the final layer.

As suggested by Fujimoto et al. (2018), the perturbation model is trained only with respect to Q_{θ_1}, following the deterministic policy gradient algorithm (Silver et al., 2014), by performing a residual update on a single action sampled from the generative model:

$$\begin{aligned}
&\phi \leftarrow \operatorname*{argmax}_{\phi} \sum_{(s,a)\in\mathcal{B}} Q_{\theta_1}(s, a + \xi_\phi(s,a,\Phi)), \\
&a \sim G_\omega(s).
\end{aligned} \tag{26}$$

To penalize uncertainty over future states, we train a pair of value estimates $\{Q_{\theta_1}, Q_{\theta_2}\}$ and take a weighted minimum between the two values as a learning target y for both networks. First n actions are sampled with respect to the generative model, and then adjusted by the target perturbation model, before passed to each target Q-network:

$$\begin{aligned}
&y = r + \gamma \max_{ai} \left[\lambda \min_{j=1,2} Q_{\theta'_j}(s', \tilde{a}_i) + (1-\lambda) \max_{j=1,2} Q_{\theta'_j}(s', \tilde{a}_i) \right], \\
&\{\tilde{a}_i = a_i + \xi_{\phi'}(s, a_i, \Phi)\}_{i=0}^n, \qquad \{a_i \sim G_\omega(s')\}_{i=0}^n, \\
&\mathcal{L}_{\text{value,i}} = \sum_{(s,a,r,s')\in\mathcal{B}} (y - Q_{\theta_i}(s,a))^2.
\end{aligned} \tag{27}$$

Both networks are trained with the same target y, where $\lambda = 0.75$.

The VAE G_ω is defined by two networks, an encoder $E_{\omega_1}(s,a)$ and decoder $D_{\omega_2}(s,z)$, where $\omega = \{\omega_1, \omega_2\}$. The encoder takes a state-action pair and outputs the mean μ and standard deviation σ of a Gaussian distribution $\mathcal{N}(\mu,\sigma)$. The state s, along with a latent vector z is sampled from the Gaussian, is passed to the decoder $D_{\omega_2}(s,z)$ which outputs an action. Each network follows the default architecture (Figure 10), with two hidden layers of size 750, rather than 400 and 300. The VAE is trained with respect to the mean squared error of the reconstruction along with a KL regularization term:

$$\mathcal{L}_{\text{reconstruction}} = \sum_{(s,a)\in\mathcal{B}} (D_{\omega_2}(s,z) - a)^2, \quad z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0,1), \tag{28}$$

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(\mathcal{N}(\mu,\sigma)||\mathcal{N}(0,1)), \tag{29}$$

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{reconstruction}} + \lambda \mathcal{L}_{\text{KL}}. \tag{30}$$

Noting the Gaussian form of both distributions, the KL divergence term can be simplified (Kingma & Welling, 2013):

$$D_{\text{KL}}(\mathcal{N}(\mu,\sigma)||\mathcal{N}(0,1)) = -\frac{1}{2}\sum_{j=1}^{J}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2), \tag{31}$$

where J denotes the dimensionality of z. For each experiment, J is set to twice the dimensionality of the action space. The KL divergence term in $\mathcal{L}_{\text{VAE}}$ is normalized across experiments by setting $\lambda = \frac{1}{2J}$. During inference with the VAE, the latent vector z is clipped to a range of $[-0.5, 0.5]$ to limit generalization beyond previously seen actions. For the small scale experiments in Supplementary Material D.3, L_2 regularization with weight 10^{-3} was used for the VAE to compensate for the small number of samples. The other networks remain unchanged. In the value network update (Equation 27), the VAE is sampled multiple times and passed to both the perturbation model and each Q-network. For each state in the mini-batch the VAE is sampled $n = 10$ times. This can be implemented efficiently by passing a latent vector with batch size $10 \cdot$ `batch size`, effectively 1000, to the VAE and treating the output as a new mini-batch for the perturbation model and each Q-network. When running the agent in the environment, we sample from the VAE 10 times, perturb each action with ξ_ϕ and sample the highest valued action with respect to Q_{θ_1}.

DDPG. Our DDPG implementation deviates from some of the default architecture and hyper-parameters to mimic the original implementation more closely (Lillicrap et al., 2015). In particular, the action is only passed to the critic at the second layer (Figure 11), the critic uses L_2 regularization with weight 10^{-2}, and the actor uses a reduced learning rate of 10^{-4}.

```
(state dimension, 400)
ReLU
(400 + action dimension, 300)
RelU
(300, 1)
```

Figure 11. DDPG Critic Network Architecture.

As done by Fujimoto et al. (2018), our DDPG agent randomly selects actions for the first 10k time steps for HalfCheetah-v1, and 1k time steps for Hopper-v1 and Walker2d-v1. This was found to improve performance and reduce the likelihood of local minima in the policy during early iterations.

DQN. Given the high dimensional nature of the action space of the experimental environments, our DQN implementation selects actions over an independently discretized action space. Each action dimension is discretized separately into 10 possible actions, giving $10J$ possible actions, where J is the dimensionality of the action-space. A given state-action pair (s, a) then corresponds to a set of state-sub-action pairs (s, a_{ij}), where $i \in \{1, ..., 10\}$ bins and $j = \{1, ..., J\}$ dimensions. In each DQN update, all state-sub-action pairs (s, a_{ij}) are updated with respect to the average value of the target state-sub-action pairs (s', a'_{ij}). The learning update of the discretized DQN is as follows:

$$y = r + \frac{\gamma}{n} \sum_{j=1}^{J} \max_i Q_{\theta'}(s', a'_{ij}), \tag{32}$$

$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \sum_{(s,a,r,s') \in \mathcal{B}} \sum_{j=1}^{J} \left(y - Q_\theta(s, a_{ij})\right)^2, \tag{33}$$

Where a_{ij} is chosen by selecting the corresponding bin i in the discretized action space for each dimension j. For clarity, we provide the exact DQN network architecture in Figure 12.

```
(state dimension, 400)
ReLU
(400, 300)
RelU
(300, 10 action dimension)
```

Figure 12. DQN Network Architecture.

Behavioral Cloning. We use two behavioral cloning methods, VAE-BC and BC. VAE-BC is implemented and trained exactly as $G_\omega(s)$ defined for BCQ. BC uses a feed-forward network with the default architecture and hyper-parameters, and trained with a mean-squared error reconstruction loss.