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

$$\epsilon^{\pi}_{\text{MDP}} = \sum_{s} \mu_{\pi}(s) \sum_{a} \pi(a|s) |\epsilon_{\text{MDP}}(s, a)|. \tag{1}$$

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:

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha (r + \gamma \max_{a' \text{s.t.} (s', a') \in \mathcal{B}} Q(s', a')). \tag{2}$$

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:*

$$\begin{aligned} \epsilon_{\text{MDP}}(s, a) = & \sum_{s'} (p_M(s'|s, a) - p_{\mathcal{B}}(s'|s, a)) \left(r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') \left(Q^{\pi}_{\mathcal{B}}(s', a') \right) \right) \\ & + p_M(s'|s, a) \gamma \sum_{a'} \pi(a'|s') \epsilon_{\text{MDP}}(s', a'). \end{aligned} \tag{3}$$

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

$$
\begin{aligned}
\epsilon_{\text{MDP}}(s,a) &= Q^\pi(s,a) - Q^\pi_{\mathcal{B}}(s,a) \\
&= \sum_{s'} p_M(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \right) - Q^\pi_{\mathcal{B}}(s,a) \\
&= \sum_{s'} p_M(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \right) \\
&\quad - \left(\sum_{s'} p_{\mathcal{B}}(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a') \right) \right) \\
&= \sum_{s'} \left(p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a) \right) r(s,a,s') + p_M(s'|s,a) \gamma \sum_{a'} \pi(a'|s') \left(Q^\pi_{\mathcal{B}}(s',a') + \epsilon_{\text{MDP}}(s',a') \right) \\
&\quad - p_{\mathcal{B}}(s'|s,a) \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a') \\
&= \sum_{s'} \left(p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a) \right) r(s,a,s') + p_M(s'|s,a) \gamma \sum_{a'} \pi(a'|s') \left(Q^\pi_{\mathcal{B}}(s',a') + \epsilon_{\text{MDP}}(s',a') \right) \\
&\quad + p_M(s'|s,a) \gamma \sum_{a'} \pi(a'|s') \left(\epsilon_{\text{MDP}}(s',a') - \epsilon_{\text{MDP}}(s',a') \right) - p_{\mathcal{B}}(s'|s,a) \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a') \\
&= \sum_{s'} \left(p_M(s'|s,a) - p_{\mathcal{B}}(s'|s,a) \right) \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi_{\mathcal{B}}(s',a') \right) \\
&\quad + p_M(s'|s,a) \gamma \sum_{a'} \pi(a'|s') \epsilon_{\text{MDP}}(s',a')
\end{aligned}
\tag{4}
$$

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 $\pi^*(s) = \text{argmax}_{a\ s.t.(s,a)\in\mathcal{B}} Q^{\pi}_{\mathcal{B}}(s,a)$ 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:*

$$\Delta_{t+1}(x_t) = (1 - \zeta_t(x_t))\Delta_t(x_t) + \zeta_t(x_t)F_t(x_t), \tag{5}$$

where $x_t \in X$ and $t = 0, 1, 2, ...$ 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, $t = 1, 2, ...$ 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. *$||\mathbb{E}[F_t|P_t]|| \leq \kappa||\Delta_t|| + c_t$ 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*

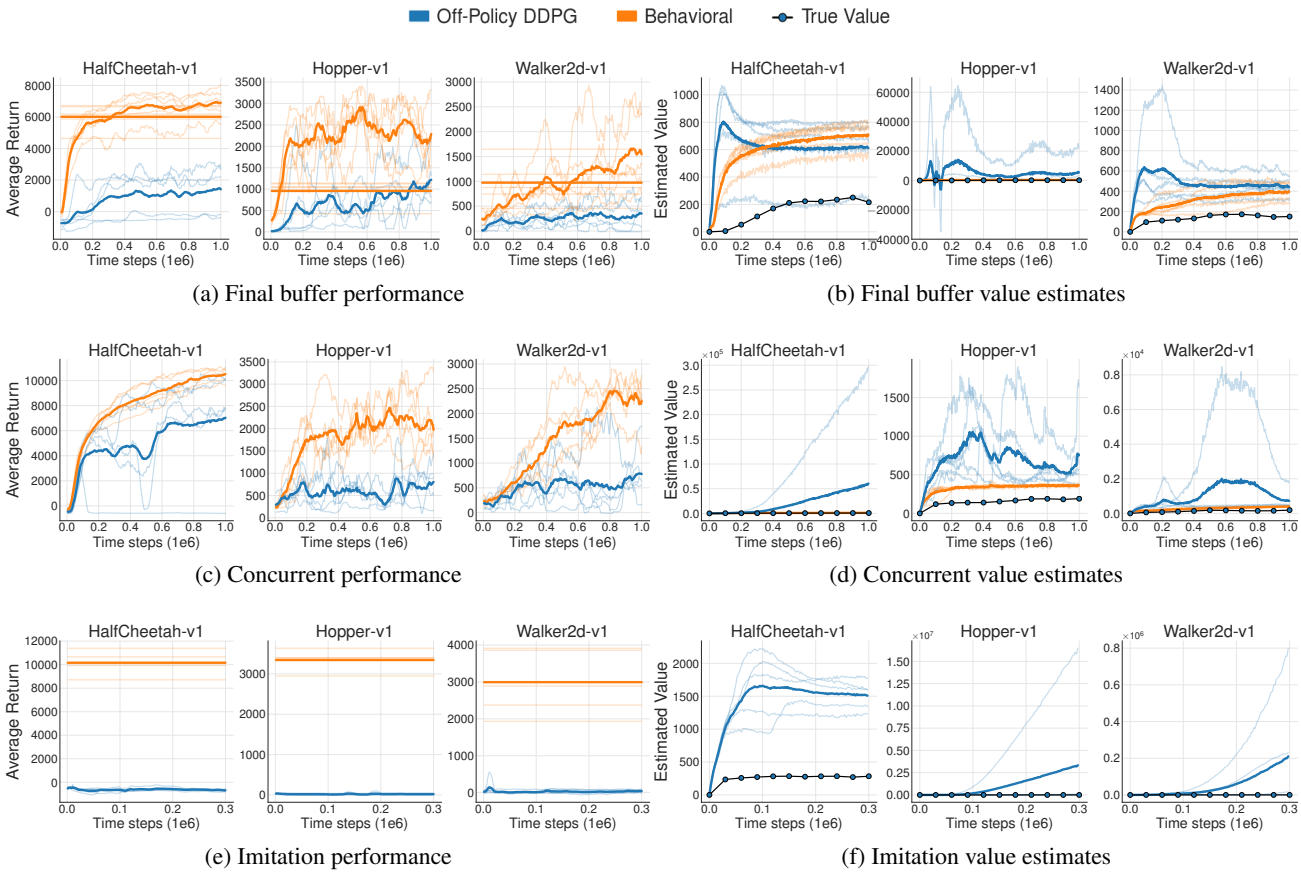
Where $||\cdot||$ denotes the maximum norm. Then Δ_t converges to 0 with probability 1.

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 $P_t = \{Q_0, s_0, a_0, \alpha_0, r_1, s_1, ...s_t, a_t\}$. 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



(a) Final buffer performance

(b) Final buffer value estimates

(c) Concurrent performance

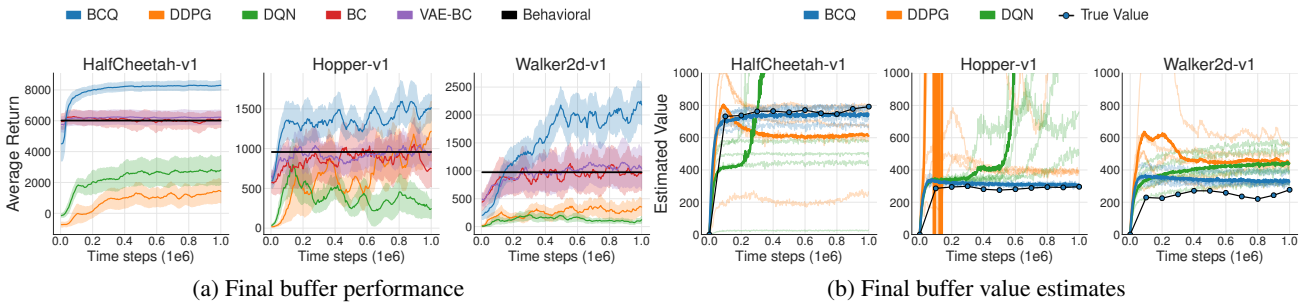
(d) Concurrent value estimates

(e) Imitation performance

(f) Imitation value estimates

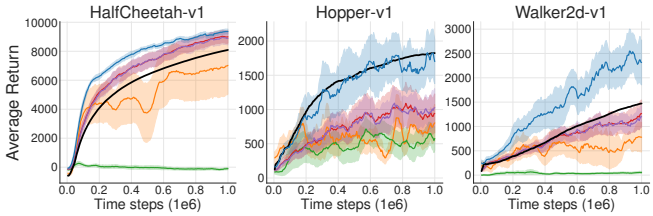
Figure 1. We examine the performance of DDPG in three batch tasks. Each individual trial is plotted with a thin line, with the mean in bold (evaluated without exploration noise). Straight lines represent the average return of episodes contained in the batch (with exploration noise). An estimate of the 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

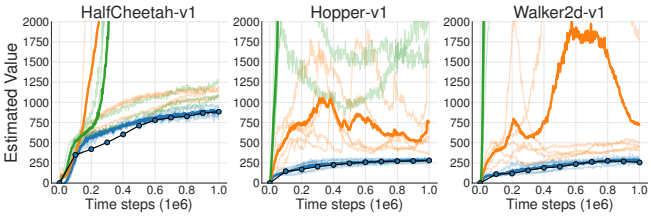


(a) Final buffer performance

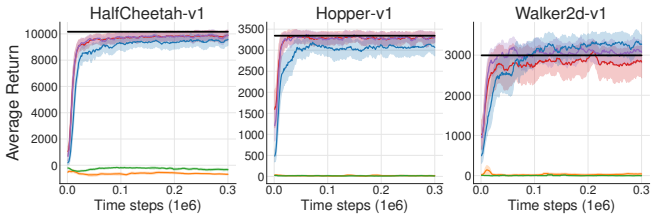
(b) Final buffer value estimates



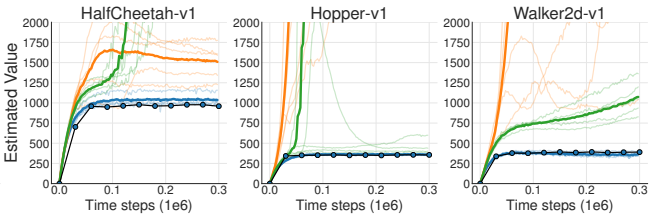
(c) Concurrent performance



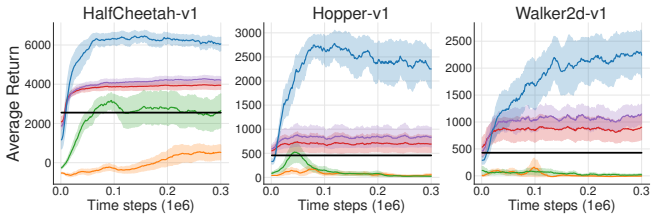
(d) Concurrent value estimates



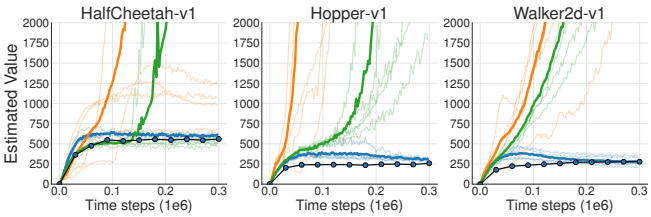
(e) Imitation performance



(f) Imitation value estimates



(g) Imperfect demonstrations performance



(h) Imperfect demonstrations value estimates

Figure 2. 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 2. 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:

$$Q(s,a) = \sum_{(s^a_{\mathcal{B}}, a, r, s'_{\mathcal{B}}) \in \mathcal{B}} \kappa^a_\tau(s, s^a_{\mathcal{B}})[r + \gamma V(s'_{\mathcal{B}})], \tag{6}$$

$$\kappa^a_\tau(s, s^a_{\mathcal{B}}) = \frac{k_\tau(s, s^a_{\mathcal{B}})}{\sum_{\tilde{s}^a_{\mathcal{B}}} k_\tau(s, \tilde{s}^a_{\mathcal{B}})}, \qquad k_\tau(s, s^a_{\mathcal{B}}) = \phi\left(\frac{||s - s^a_{\mathcal{B}}||}{\tau}\right), \tag{7}$$

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 (6), then updates $V(s'_{\mathcal{B}})$ by evaluating $Q(s'_{\mathcal{B}}, a)$ for all $s_{\mathcal{B}} \in \mathcal{B}$ and $a \in \mathcal{A}$.

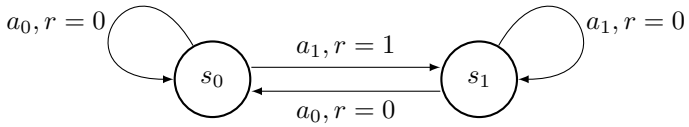


Figure 3. Toy MDP with two states s_0 and s_1, and two actions a_0 and a_1. Agent receives reward of 1 for selecting a_1 at s_0 and 0 otherwise.

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 3, 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 (6) provides the following:

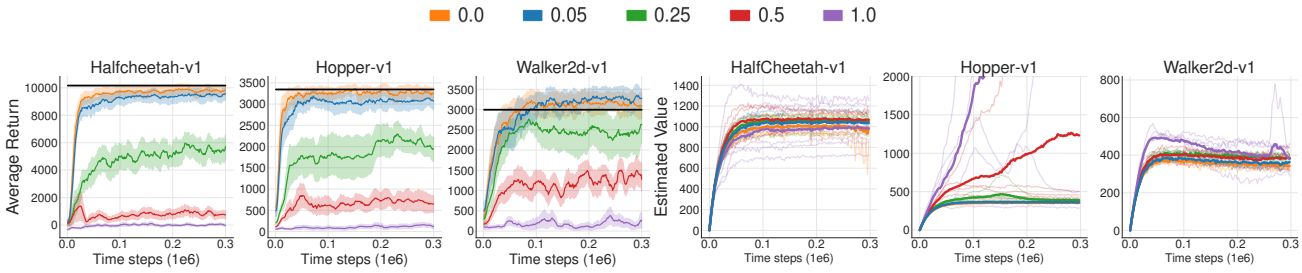
$$Q(\cdot, a_1) = 1 + \gamma V(s_1) = 1 + \gamma Q(s_1, a_0), \qquad Q(\cdot, a_0) = \gamma V(s_0) = \gamma Q(s_0, a_1). \tag{8}$$

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 $\Phi = a_{\max} - a_{\min}$, 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 4 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 4. 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

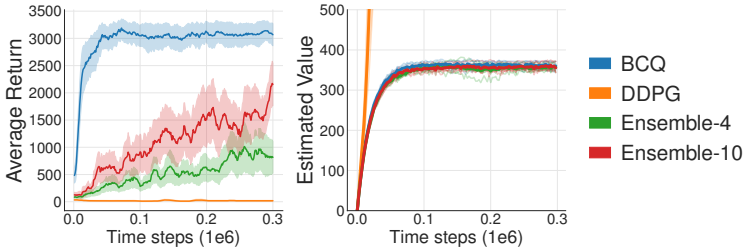
Section 4.2 proposes using generation as a method for constraining the output of the policy π to eliminate actions which are unlikely under the batch. However, a more natural approach would be through approximate uncertainty-based methods (Osband et al., 2016; Gal et al., 2016; Azizzadenesheli et al., 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:

$$\phi \leftarrow \operatorname*{argmin}_{\phi} \sum_{(s,a)\in\mathcal{B}} \sigma\left(\{Q_{\theta_i}(s,a)\}_{i=1}^{N}\right). \tag{9}$$

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 5.



(a) Imitation performance (b) Imitation value estimates

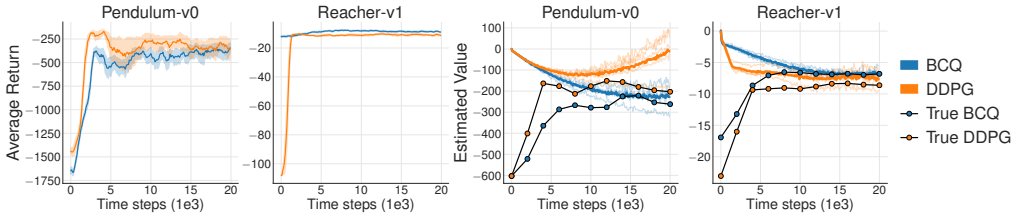
Figure 5. A comparison with uncertainty-based methods on the Hopper-v1 environment from OpenAI gym, on the imitation task. Although ensemble based methods are able to generate stable value functions (right) under these conditions, they fail to constrain the action space to the demonstrated expert actions 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 6, 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 6. 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:

$$\log p(X) \geq \mathbb{E}_{q(X|z)}[\log p(X|z)] + D_{\text{KL}}(q(z|X)||p(z)), \tag{10}$$

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):

$$\mathbb{E}_{z\sim\mathcal{N}(\mu,\sigma)}[f(z)] = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[f(\mu + \sigma\epsilon)]. \tag{11}$$

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 7 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:

$$y = \begin{cases} r & \text{if terminal } s' \\ r + \gamma Q_{\theta'}(s', a') & \text{else} \end{cases} \tag{12}$$

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 7. 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 7) 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:

$$\begin{aligned} \phi \leftarrow \operatorname*{argmax}_{\phi} \sum_{(s,a)\in\mathcal{B}} Q_{\theta_1}(s, a + \xi_\phi(s,a,\Phi)), \\ a \sim G_\omega(s). \end{aligned} \tag{13}$$

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:

$$\begin{aligned} y &= r + \gamma \max_{ai} \left[\lambda \min_{j=1,2} Q_{\theta'_j}(s', \tilde{a}_i) + (1-\lambda) \max_{j=1,2} Q_{\theta'_j}(s', \tilde{a}_i) \right], \\ &\{\tilde{a}_i = a_i + \xi_{\phi'}(s, a_i, \Phi)\}_{i=0}^n, \qquad \{a_i \sim G_\omega(s')\}_{i=0}^n, \\ \mathcal{L}_{\text{value,i}} &= \sum_{(s,a,r,s')\in\mathcal{B}} (y - Q_{\theta_i}(s,a))^2. \end{aligned} \tag{14}$$

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 7), 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:

$$\mathcal{L}_{\text{reconstruction}} = \sum_{(s,a)\in\mathcal{B}} (D_{\omega_2}(s,z) - a)^2, \quad z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0,1), \tag{15}$$

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(\mathcal{N}(\mu,\sigma) || \mathcal{N}(0,1)), \tag{16}$$

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{reconstruction}} + \lambda \mathcal{L}_{\text{KL}}. \tag{17}$$

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

$$D_{\text{KL}}(\mathcal{N}(\mu,\sigma) || \mathcal{N}(0,1)) = -\frac{1}{2} \sum_{j=1}^{J} (1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2), \tag{18}$$

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 14), 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 8), 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 8. 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:

$$y = r + \frac{\gamma}{n} \sum_{j=1}^{J} \max_i Q_{\theta'}(s', a'_{ij}), \tag{19}$$

$$\theta \leftarrow \operatorname*{argmin}_\theta \sum_{(s,a,r,s') \in \mathcal{B}} \sum_{j=1}^{J} \left(y - Q_\theta(s, a_{ij})\right)^2, \tag{20}$$

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 9.

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

Figure 9. 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.

References

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

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.

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

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

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.

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.

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.

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.

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.

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

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.

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.

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.