BATCH-CONSTRAINED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

This work examines batch reinforcement learning–the task of maximally exploiting a given batch of off-policy data, without further data collection. Due to errors introduced by extrapolation, we find that standard off-policy deep reinforcement learning algorithms such as DQN and DDPG are only capable of learning with data correlated to their current policy, making them ineffective for most off-policy applications. We introduce a novel class of off-policy algorithms, batch-constrained reinforcement learning, which restricts the action space to force the agent towards behaving on-policy with respect to a subset of the given data. We extend this notion to deep reinforcement learning, and to the best of our knowledge, present the first continuous control deep reinforcement learning algorithm which can learn effectively from uncorrelated off-policy data.

1 INTRODUCTION

Learning only from a batch of data without requiring 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 in many areas, such as robotics, imitation learning and exploration. In 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 performed to produce good policies. However, imitation learning algorithms such as behavioral cloning are known to fail when exposed to noisy or suboptimal trajectories (Ross et al., 2011), and more sophisticated approaches are generally unable to surpass the performance of the demonstrator without further interactions with the environment (Ziebart et al., 2008; Ho & Ermon, 2016; Hester et al., 2017). In most deep reinforcement learning algorithms, a single policy, or variations of a single policy, are used for both exploration and exploitation. This paradigm creates a trade-off between future learning and current performance, and limits the agent to suboptimal policies, unless careful annealing of the exploratory behavior can be performed. In contrast, batch reinforcement learning offers a mechanism for learning from decoupled exploration without suffering from degradation in current performance.

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 the process is repeated. However, we find that these off-policy algorithms are generally unsuccessful, unless the available batch contains data correlated to the current policy. We demonstrate that this inability to truly learn off-policy is largely due to *extrapolation error*, a phenomenon in which unseen state-action pairs in the learning target are erroneously estimated to have unrealistic values, which may make them more attractive than observed actions. In other words, the uncertainty of unseen state-action pairs and generalization from function approximation, can cause poor value estimates and suboptimal action selection.

To overcome extrapolation error in off-policy learning, we introduce batch-constrained reinforcement learning, in which agents are trained to maximize reward *by only selecting previously seen actions*. We prove that this paradigm induces desirable properties in the discrete MDP setting, such as converging to the optimal policy without needing to experience all possible transitions. We generalize this notion to continuous control with neural networks by training a state-conditioned generative model to produce only previously seen actions. This generative model is combined with a policy which aims to optimally perturb the generated actions in a small range, resulting in the agent only using actions similar to those previously seen.

We test our algorithm on several MuJoCo (Todorov et al., 2012) environments, where extrapolation error is particularly problematic due to the high dimensional continuous action space, which makes it impossible to sample the action space exhaustively. Unlike any previous continuous control reinforcement learning algorithm, our approach is able to learn successfully in a variety of batch reinforcement learning settings. 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. To ensure reproducibility, we provide precise experimental and implementation details and our code will be made available.

2 BACKGROUND

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

The agent selects actions with respect to a policy $\pi : \mathcal{S} \to \mathcal{A}$, which has a corresponding value function $Q^\pi(s,a) = \mathbb{E}_\pi[R_t|s,a]$, estimating the expected return when following the policy π after taking action a in state s. Given Q^π, a new policy π' of equal or better performance can be derived by greedy maximization $\pi' = \operatorname{argmax}_a Q^\pi(s,a)$ (Sutton & Barto, 1998). For a given policy π, the value function can be estimated using sampled versions of 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 DQN algorithm (Mnih et al., 2015). In DQN, the value function Q_θ is updated using target:

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

Deep Q-learning is an *off-policy* algorithm (Sutton & Barto, 1998), meaning that the target can be computed without consideration of how the experience was generated. As a result, 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'), sampled from an experience replay dataset (Lin, 1992). To further stabilize learning, a target network with frozen parameters $Q_{\theta'}$, is used in the learning target y. 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 with the deterministic policy gradient theorem (Silver et al., 2014):

$$\phi = \operatorname*{argmax}_{\tilde{\phi}} \mathbb{E}_{s \in \mathcal{B}}[Q_\theta(s, \pi_{\tilde{\phi}}(s))], \tag{3}$$

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

Off-policy algorithms commonly assume infinite state-action visitation for convergence or optimality guarantees (Watkins, 1989; Precup et al., 2001), which is generally impossible to satisfy in any

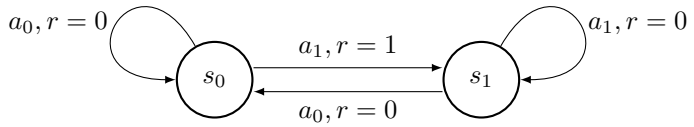


Figure 1: Toy MDP with two states s_0 and s_1, and two actions a_0 and a_1. Agent receives reward of 1 for selecting a_1 at s_0 and 0 otherwise.

practical problem. In this section, we analyze how breaking this assumption can introduce large amounts of error into the value function by a phenomenon we call *extrapolation error*. We show how this error can lead to degenerate policies for even provably convergent batch algorithms, such as kernel-based reinforcement learning (Ormoneit & Sen, 2002). Furthermore, we find that state of the art deep reinforcement learning algorithms, such as DDPG (Lillicrap et al., 2015) and DQN (Mnih et al., 2015), are generally incapable of learning from data uncorrelated to the current policy. Consequently, current deep RL algorithms are unsatisfactory solutions for the batch setting, where no assumptions can be made on the data collection process.

Informally, extrapolation error comes from training the value estimate with state-action pairs that are not present in the training data. 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). When learning off-policy or in a batch setting, this extrapolation error may never be corrected, due to the inability to collect new data, resulting in disastrous value estimation and degenerate policies.

3.1 A Simple Example

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:

$$Q(s, a) = \sum_{(s^a_{\mathcal{B}}, a, r, s'_{\mathcal{B}}) \in \mathcal{B}} \kappa^a_\tau(s, s^a_{\mathcal{B}})[r + \gamma V(s'_{\mathcal{B}})], \tag{4}$$

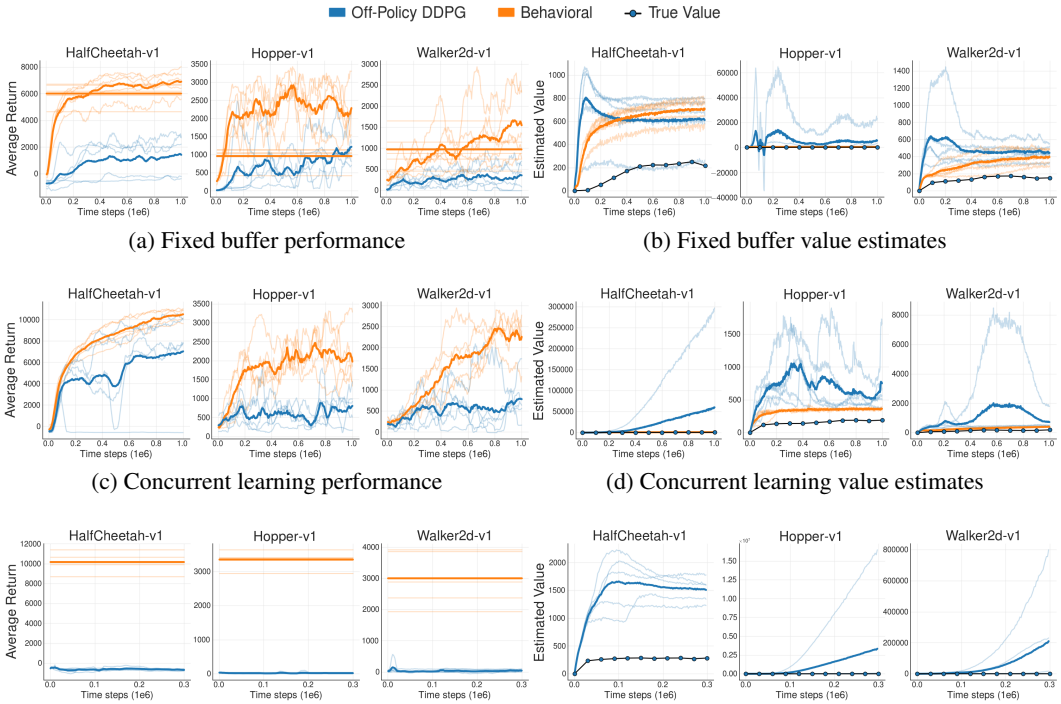
$$\kappa^a_\tau(s, s^a_{\mathcal{B}}) = \frac{k_\tau(s, s^a_{\mathcal{B}})}{\sum_{\tilde{s}^a_{\mathcal{B}}} k_\tau(s, \tilde{s}^a_{\mathcal{B}})}, \qquad k_\tau(s, s^a_{\mathcal{B}}) = \phi\left(\frac{||s - s^a_{\mathcal{B}}||}{\tau}\right), \tag{5}$$

where states $s^a_{\mathcal{B}} \in \mathcal{S}$ corresponding to the action a $(s_{\mathcal{B}}, a) \in \mathcal{B}$, 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 (4), then updates $V(s'_{\mathcal{B}})$ by evaluating $Q(s'_{\mathcal{B}}, a)$ for all $s_{\mathcal{B}} \in \mathcal{B}$ and $a \in \mathcal{A}$.

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 1, 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 (4) provides the following:

$$Q(\cdot, a_1) = 1 + \gamma V(s_1) = 1 + \gamma Q(s_1, a_0), \qquad Q(\cdot, a_0) = \gamma V(s_0) = \gamma Q(s_0, a_1). \tag{6}$$

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 can cause an increasing sequence of value estimates.



(a) Fixed buffer performance

(b) Fixed buffer value estimates

(c) Concurrent learning performance

(d) Concurrent learning value estimates

(e) Imitation learning performance

(f) Imitation learning value estimates

Figure 2: We examine the performance of DDPG in three off-policy settings. Each individual trial is plotted with a thin line, with the mean in bold. Straight lines represent the average return of episodes contained in the batch. An estimate of the true value of the off-policy agent, evaluated by Monte Carlo returns, is marked by a dotted line. In the fixed buffer experiment, the off-policy agent learns from a large, diverse dataset, but exhibits poor learning behavior and value estimation. In the concurrent learning setting the agent learns alongside a behavioral agent, with access to the same data, but suffers in performance. In the imitation learning setting, the agent receives data from an expert policy but is unable to learn, and exhibits highly divergent value estimates.

3.2 Extrapolation error in Deep Reinforcement Learning

While Section 3.1 examined a specific counterexample, this raises a key question–*what is the role of extrapolation error in a practical setting?* In this section, we examine the behavior of a state of the art deep actor-critic algorithm, DDPG (Lillicrap et al., 2015), when learned with off-policy data. We find that extrapolation error, when combined with the maximization via gradient updates in the policy updates of DDPG results in large amounts of overestimation bias and reduced performance. To examine the bias induced from off-policy learning, we examine three different batch settings in the MuJoCo environments of OpenAI gym (Todorov et al., 2012; Brockman et al., 2016), which we use to train an off-policy DDPG agent with no interaction with the environment:

Batch 1 (Fixed buffer). We train a DDPG (Lillicrap et al., 2015) agent for 1 million time steps, adding large amounts of Gaussian noise, $\mathcal{N}(0, 0.5)$, for exploration, and storing all experienced transitions. This collection procedure creates a dataset with a diverse set of states and actions.

Batch 2 (Concurrent learning). We simultaneously train the off-policy agent with the behavioral agent, for 1 million time steps. The behavioral policy performs data collection with $\mathcal{N}(0, 0.1)$ Gaussian noise added for exploration. Each transition experienced by the behavioral policy is stored in a buffer replay which both agents learn from. Experiment 2 differs from Experiment 1 as the agent learns as the buffer replay grows.

Batch 3 (Imitation). A trained DDPG agent acts as an expert, and is used to collect a dataset of 1 million transitions, without added noise.

In each experiment we graph the performance of the agents as they train (left), as well as their value estimates (right), in Figure 2. Straight lines represent the average return of episodes contained in

the batch. For better comparison, we additionally graph the learning performance of the behavioral agent for the fixed buffer experiment. Naïvely, we might expect the off-policy agent trained with Batch 1 to perform well, given the large amounts of exploration, and size of the buffer, however, we find that the agent performs far worse than the behavioral agent. Furthermore, we find the value function overestimates the true value.

In experiments with Batch 2, both agents sees the same buffer, we might expect the performance to be very similar. However, due to small differences in the initial policies, the impact of extrapolation error on the off-policy agent is observed. There is a notable difference in performance, and the value estimates of the off-policy agent is consistently higher than the behavioral value estimates.

Results of the agent trained with Batch 3 demonstrates the failure of off-policy deep reinforcement learning algorithms when the state-action coverage is limited, as exploration is hampered due to the unchanging expert policy. In this experiment the off-policy agent fails to learn any meaningful behavior and the value function demonstrates divergent behavior.

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 spaces in MuJoCo environments are contributing factors to extrapolation error, the scale of these tasks is small compared to real world settings. However, even if a sufficient amount of data collection occurs, the concerns of catastrophic forgetting (McCloskey & Cohen, 1989; Goodfellow et al., 2013) may still result in extrapolation error. Consequently, off-policy reinforcement learning algorithms used in the real-world will require practical guarantees without infinite data.

4 Off-Policy Reinforcement Learning without Infinite Data

Extrapolation error is introduced when we evaluate a learning target with an *unseen* state-action pair, during off-policy learning without infinite data. To address this concern, in this section we propose a learning agent which aims to act optimally while selecting actions from corresponding *seen* state-action pairs. In other words, for a given batch, we aim to learn the optimal agent which is on-policy with respect to a subset of the provided data. We begin by deriving the batch-constrained policy iteration algorithm in a discrete MDP setting in Section 4.1 and then extend the algorithm to a deep reinforcement learning setting in Section 4.2.

4.1 Batch-Constrained Reinforcement Learning for Discrete MDPs

While extrapolation error is induced by function approximation, this section aims to introduce theoretical concepts and explain why a constrained action-space approach is a desirable strategy, even in a discrete MDP setting without function approximation. To begin formalizing the notion of only learning with seen data, we define a batch $\mathcal{B}$ as a set of tuples (s, a, r, s'), where we further assume if $(s, a, r, s') \in \mathcal{B}$ then $s' \in \mathcal{B}$ unless s' is a terminal state. For a given batch $\mathcal{B}$, we define the set of batch-constrained policies $\Pi_{\mathcal{B}}$, where $\pi \in \Pi_{\mathcal{B}} : \mathcal{S} \to \mathcal{A}$, maps states to actions with the condition that if $s \in \mathcal{B}$ then the policy must select previously seen actions such that $(s, \pi(s)) \in \mathcal{B}$.

In a practical reinforcement learning setup, the policy can only be trained with previously collected state-action pairs, thus we define the batch Bellman operator $\mathcal{T}^{\pi}_{\mathcal{B}}$, which will produce ∞ if a state-action pair is not contained in the batch:

$$\mathcal{T}^{\pi}_{\mathcal{B}} Q(s, a) = \begin{cases} \mathbb{E}_{s'}[r + \gamma Q(s', \pi(s'))] & \text{if } (s, a) \in \mathcal{B} \\ \infty & \text{otherwise.} \end{cases} \tag{7}$$

The batch Bellman operator serves as a representation for extrapolation error, as it will produce ∞ in regions that are uncertain due to unseen data points. For a given batch $\mathcal{B}$ and policy π, we can use the batch Bellman operator to compute the value of a batch-constrained policy $\pi \in \Pi_{\mathcal{B}}$.

Lemma 1 (Batch Bellman Policy Evaluation). *For a given batch $\mathcal{B}$, MDP M, and batch-constrained policy $\pi \in \Pi_{\mathcal{B}}$, the batch Bellman operator $\mathcal{T}^{\pi}_{\mathcal{B}}$ repeatedly applied to an initial $Q_0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $Q_{k+1} = \mathcal{T}^{\pi}_{\mathcal{B}} Q_k$ converges to a fixed point $Q^{\pi}_{\mathcal{B}}(s, a)$ where $Q^{\pi}_{\mathcal{B}}(s, a) = Q^{\pi}(s, a)$ for $(s, a) \in \mathcal{B}$, as $k \to \infty$.*

Lemma 1 states if $\pi \in \Pi_{\mathcal{B}}$, the resulting value function $Q^{\pi}_{\mathcal{B}}(s, a)$ will be accurate, i.e. $Q^{\pi}_{\mathcal{B}}(s, a) = Q^{\pi}(s, a)$, at any state-action pair (s, a) contained in the batch $\mathcal{B}$. This Lemma leads us to a key result

concerning batch-constrained policies, which states only batch-constrained policies can produce accurate value functions with respect to the batch Bellman operator.

Theorem 1 (Batch Bellman Necessary and Sufficient Condition). *Let $Q^\pi_\mathcal{B}(s,a)$ be the fixed point of the batch Bellman operator $\mathcal{T}^\pi_\mathcal{B}$, then $Q^\pi_\mathcal{B}(s,a) = Q^\pi(s,a)$ for all $(s,a) \in \mathcal{B}$ if and only if $\pi \in \Pi_\mathcal{B}$.*

The batch Bellman operator overestimates the value for unseen states, analogous to the worst-case of extrapolation error in a function approximation setting. Theorem 1 implies with access to only a subset of state-action pairs in the MDP, the value function $Q^\pi_\mathcal{B}$ learned with the batch Bellman operator will be inaccurate unless π is batch-constrained. This suggests batch-constrained policies are a necessary tool for combating extrapolation bias.

Next, we demonstrate batch-constrained policies can be used to learn the optimal policy. We first generalize the policy improvement theorem to a batch-constrained setting. Given the value function of any batch-constrained policy, a policy of equal or higher value can be computed by greedily selecting actions a of highest value that satisfy the batch constraint $(s,a) \in \mathcal{B}$.

Lemma 2 (Batch-Constrained Policy Improvement). *Let $\pi \in \Pi_\mathcal{B}$ be any deterministic batch-constrained policy and $\pi'(s) = \operatorname{argmax}_{a \text{ s.t.} (s,a)\in\mathcal{B}} Q^\pi(s,a)$, then for all $s \in \mathcal{B}$, $Q^{\pi'}(s,\pi'(s)) \geq Q^\pi(s,\pi(s))$.*

The previous Lemma, along with the standard policy evaluation result (Sutton & Barto, 1998) can be brought together to provide a policy iteration result. By repeatedly evaluating an initial batch-constrained policy, and greedily maximizing with respect to value function and batch constraints, the policy converges to the optimal batch-constrained policy.

Theorem 2 (Batch-Constrained Policy Iteration). *Given a batch $\mathcal{B}$ and MDP M, then the repeated application of policy evaluation and batch-constrained policy improvement converges to a policy $\pi^*_\mathcal{B}$, such that $Q^{\pi^*_\mathcal{B}}(s,\pi^*_\mathcal{B}(s)) \geq Q^{\pi_\mathcal{B}}(s,\pi_\mathcal{B}(s))$ for all $\pi_\mathcal{B} \in \Pi_\mathcal{B}$ and $s \in \mathcal{B}$.*

Optimality is a direct result from Theorem 1, namely, if the batch contains all the state-action pairs from an optimal policy π^*, batch-constrained policy iteration converges to the optimal policy.

Corollary 1 (Batch-Constrained Optimality). *Given a batch $\mathcal{B}$, and MDP M if for all $s \in \mathcal{S}$, $(s,a^*,r,s') \in B$ where $a^* = \operatorname{argmax}_a Q^*(s,a)$, then batch-constrained policy iteration converges to a policy $\pi^* = \operatorname{argmax}_a Q^*(s,a)$ for all $s \in \mathcal{S}$.*

This result can be extended to show the direct connection from batch-constrained policy iteration and imitation learning. Given an optimal trajectory, batch-constrained policy iteration converges to a policy which imitates the optimal trajectory over the state space it covers. This result demonstrates that batch-constrained policy iteration is able to learn meaningful policies in certain regions of the state space, without access to the entire MDP.

Corollary 2 (Batch-Constrained Imitation). *Given a deterministic MDP, and trajectory $T = (s_0, a^*_0, r_0, ..., s_{i+1}, a^*_{i+1}, r_{i+1}, ...)$ where $a^*_i = \operatorname{argmax}_a Q^*(s_i,a)$, then batch-constrained policy iteration converges to $\pi^*(s_i) = \operatorname{argmax}_a Q^*(s_i,a)$ for all $s_i \in T$.*

4.2 ϵ-BATCH-CONSTRAINED REINFORCEMENT LEARNING WITH CONTINUOUS ACTIONS

In this section, we introduce an approach to batch-constrained reinforcement learning with function approximation by softening the batch-constraint. Batch-constrained reinforcement learning generalizes poorly to continuous states, as for a given state $s \in \mathcal{B}$, it is unlikely that there is more than one corresponding action a such that $(s,a) \in \mathcal{B}$. This is problematic as a batch-constrained policy will learn to simply imitate previously taken actions, which may be undesirable if the batch contains any suboptimal actions. To address this concern, we introduce ϵ-batch-constrained policies, which we implement through an approximation to the seen action distribution. We then introduce our deep batch reinforcement learning algorithm, Batch-Constrained deep Q-learning (BCQ).

ϵ-Batch-Constrained Policy. For a given batch $\mathcal{B}$, $\epsilon \geq 0$, and distance functions $D_S : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ and $D_A : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$, we define a ϵ-batch-constrained policy π:

$$\pi : \mathcal{S} \to \mathcal{A}_{s,\epsilon} \text{ where } \mathcal{A}_{s,\epsilon} = \{a \in \mathcal{A} : \min_{(s_\mathcal{B},a_\mathcal{B})\in\mathcal{B}} D_S(s,s_\mathcal{B}) + D_A(a,a_\mathcal{B}) \leq e\} \tag{8}$$

Algorithm 1 BCQ

Input: Batch $\mathcal{B}$, horizon T, target network update rate τ, mini-batch size N, max perturbation Φ
Initialize critic network Q_θ, value network V_ψ, policy network π_ϕ, and VAE G_ω, with random parameters θ, ψ, ϕ, ω, and target network $V_{\psi'}$ with $\psi' \leftarrow \psi$
for $t = 1$ **to** T **do**
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a}, \mu, \sigma = G_\omega(s)$
  Update VAE: $\omega \leftarrow \text{argmin}_\omega \, N^{-1} \sum (a - \tilde{a})^2 + D_{\text{KL}}(\mathcal{N}(\mu, \sigma) || \mathcal{N}(0, 1))$
  Update critic: $\theta \leftarrow \text{argmin}_\theta \, N^{-1} \sum ((r + \gamma V_{\psi'}(s')) - Q_\theta(s, a))^2$
  Update policy: $\phi \leftarrow \text{argmax}_\phi \, N^{-1} \sum Q(s, a + \pi(s, a; \Phi)), \quad a \sim G_\omega(s)$
  Sample n actions: $\{a_i \sim G_\omega(s')\}_{i=1,\ldots,n}$
  Perturb each action: $\{a_i = a_i + \pi_\phi(s', a_i; \Phi)\}_{i=1,\ldots,n}$
  Update value: $\psi \leftarrow \text{argmin}_\psi \, N^{-1} \sum ((\max_{a_i} Q_\theta(s', a_i)) - V_\psi(s))^2$
  Update target network: $\psi' \leftarrow \tau\psi + (1 - \tau)\psi'$
end for

If $\mathcal{A}_{s,\epsilon} = \varnothing$, then π should select a from the closest state-action pair $(s, a) \in \mathcal{B}$. Informally, for a given state, a ϵ-batch-constrained policy selects actions which are similar to actions contained in the batch. The hyper-parameter ϵ induces a trade-off between error introduced into the system and generalization in the action space.

While this may suggest a non-parametric approach, when dealing with large practical applications, it is generally preferable to leverage parametric function approximation for computational efficiency. Consequently, D_S will be defined by the implicit distance induced by the function approximation, and we consider the task of minimizing D_A as a generation task with a parametric generative model $G_\omega(s)$. We aim for $G_\omega(s)$ to generate actions from the approximated distribution of the batch $\mathcal{B}$. More formally, we would like to only sample $a_\mathcal{B} \sim G_\omega(s)$, where $(s_\mathcal{B}, a_\mathcal{B}) \in \mathcal{B}$ and $D_S(s, s_\mathcal{B}) \leq \epsilon$. To achieve this effect, we use a variational auto-encoder (VAE) (Kingma & Welling, 2013), which models the distribution by transforming an underlying latent space. For an introduction to VAEs, see Supplementary Material D. The VAE $G_\omega(s)$ is trained to minimize the distance D_a:

$$\omega = \underset{\tilde{\omega}}{\text{argmin}} \sum_{(s,a)\in\mathcal{B}} D_A(G_{\tilde{\omega}}(s), a). \tag{9}$$

We can define the simplest ϵ-batch-constrained policy $\pi = G_\omega$ by setting ϵ to be the smallest satisfiable value, $\epsilon = \text{argmin}_\epsilon \, A_{s,\epsilon}$ such that for all $s \in \mathcal{B}$, and any $a \sim G_\omega(s)$, $a \in A_{s,\epsilon}$. This generative model formulation is generally desirable, as it enables a multi-modal policy. Multi-modality may be critical if the batch $\mathcal{B}$ contains data from a set of policies or stochastic processes, both of which occur when using random exploration with an experience replay. Furthermore the VAE addresses catastrophic forgetting by sampling actions at similar frequencies to which they are observed.

ϵ-Batch-Constrained Policy Iteration. Given the generative model G_ω, the value function Q can be maximized by sampling n plausible candidate actions and selecting the highest valued action:

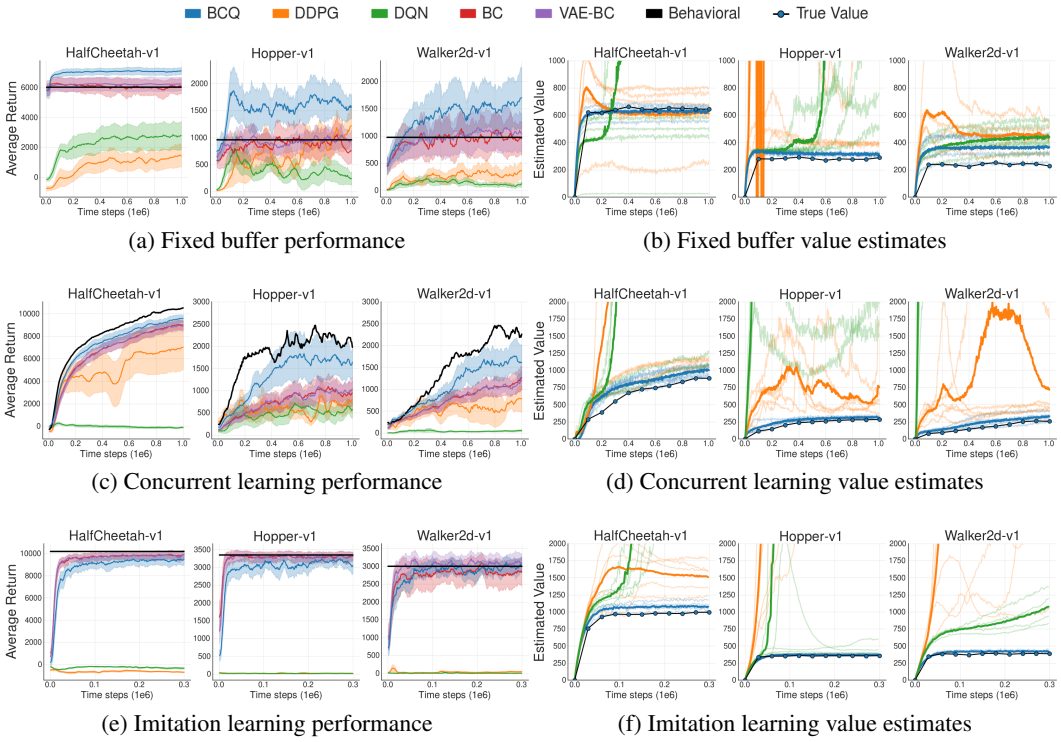
$$y = r + \gamma \max_{a_i} Q_{\theta'}(s', a_i), \quad \{a_i \sim G_\omega(s)\}_{i=1,\ldots,n}. \tag{10}$$

Assuming a sufficiently accurate G_ω, this update rule limits the error introduced into the system, but it also restricts learning as the agent is strictly limited to previously selected actions. Furthermore, even if many actions have been seen for nearby states, the generative model may need to be sampled many times to see sufficient diversity. While the hyper-parameter ϵ is bounded below by the error induced by the generative model, we can arbitrarily increase ϵ by a policy $\pi_\phi(s, a; \Phi)$ which outputs a perturbation to the action a in the range $[-\Phi, \Phi]$:

$$y = r + \gamma \max_{a_i + \pi_\phi(s', a_i)} Q_\theta(s', a_i + \pi_\phi(s', a_i; \Phi))), \quad \{a_i \sim G_\omega(s)\}_{i=1,\ldots,n}, \tag{11}$$

$$\phi = \underset{\tilde{\phi}}{\text{argmax}} \, N^{-1} \sum Q_\theta(s, a + \pi_{\tilde{\phi}}(s, a; \Phi)), \quad a \sim G_\omega(s). \tag{12}$$

Furthermore, this reduces the necessity for sampling a large number of points to provide sufficient coverage of the action space. The choice of Φ creates a trade-off between an imitation learning and



(a) Fixed buffer performance

(b) Fixed buffer value estimates

(c) Concurrent learning performance

(d) Concurrent learning value estimates

(e) Imitation learning performance

(f) Imitation learning value estimates

Figure 3: We evaluate BCQ and several baselines following the experiments from Section 3.2. The shaded area represents half a standard deviation. Value estimates include a plot of each trial, with the mean in bold. Straight lines represent the average return of episodes contained in the batch. An estimate of the true value of BCQ, evaluated by Monte Carlo returns, is marked by a dotted line. Unlike any other algorithm BCQ matches or outperforms the performance of the behavioral in all three tasks, while exhibiting a highly stable value function.

reinforcement learning algorithm. If $\Phi = 0$, and the number of sampled actions $n = 1$, then the policy resembles behavioral cloning and as $\Phi \to a_{\max} - a_{\min}$ and $n \to \infty$, then Equation (11) $\to \operatorname{argmax}_a Q(s', a)$, approaching Q-learning as the policy becomes closer to greedy maximization of the value function.

As Equation (11) includes a random sampling process, we can reduce the variance of the update by including a value network $V_\psi(s)$, with learning target y_ψ:

$$y_\psi = \max_{a_i + \pi_\phi(s', a_i)} Q_{\theta'}(s', a_i + \pi_\phi(s', a_i; \Phi))), \quad a_i \sim G_\omega(s). \tag{13}$$

This allows us to modify Equation (11) to $y = r + \gamma V_{\psi'}(s')$. This forms Batch-Constrained deep Q-learning (BCQ), which maintains four parametrized networks, an actor $\pi_\phi(s, a)$, a critic $Q_\theta(s, a)$, a value network $V_\psi(s)$, and a generative model $G_\omega(s)$. We summarize BCQ in Algorithm 1.

5 Results

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 with 10 bins per dimension, a feed-forward behavioral cloning method (BC), and a variant with a VAE (VAE-BC), mimicking $G_\omega(s)$ used by BCQ. For BCQ, we use $\Phi = 0.05$, sample $n = 10$ actions from the VAE G_ω and set the dimensionality of the latent space to twice the dimensionality of the action space for each environment. Exact network, hyper-parameter and training details of each method are provided in the Supplementary Material C. Exact experimental details are provided in the Supplementary Material B.

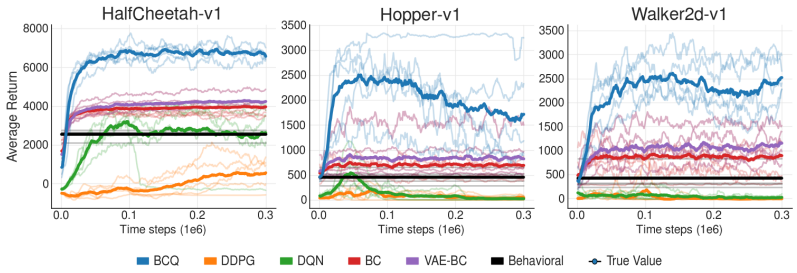


Figure 4: We examine the effectiveness of BCQ when learning from a highly noisy demonstrator. 100k transitions are provided by an expert policy with 0.3 probability of a random action and $\mathcal{N}(0, 0.3)$ added to remaining actions. Average returns include a plot of each trial, with the mean in bold. BCQ greatly outperforms behavioral cloning algorithms, as well as the behavioral policy, demonstrating robustness to suboptimal data.

We evaluate each method following the three experiments defined in Section 3.2: *fixed buffer*, learning from the final experience replay of a trained DDPG agent, *concurrent learning*, learning simultaneously with the behavioral DDPG policy, and *imitation*, learning from a dataset collected by an expert. The results of these experiments, along with 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 three tasks, matching or outperforming the behavioral policy in each instance, and outperforming all other agents, besides in imitation learning where behavioral cloning unsurprisingly performs the best. These result demonstrate our approach is capable of both performing imitation learning and off-policy learning with fixed hyper-parameters. Furthermore, unlike DDPG and DQN, BCQ exhibits a highly stable value function in the presence of off-policy samples, suggesting extrapolation error has been successfully mitigated.

Given the action space of DQN is discretized independently, it is foreseeable that it would perform poorly on these environments with multi-dimensional actions. However, this discretization reduces the action space to a more manageable scale, allowing for greater coverage of the action space, and potentially reducing the impact of extrapolation error. Unfortunately, it is clear that this hypothesis is false, and the greedy maximization of DQN produces a highly overestimated value function. Alongside extrapolation error, a likely cause is the error introduced by the independent discretization. These results suggest that naïvely reducing the size of the action space of the environment is an inadequate solution to extrapolation error.

To study the robustness of BCQ to imperfect transitions and multi-modal data, we examine its performance 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. We report the results in Figure 4. We find that both deep reinforcement learning and imitation learning algorithms preform poorly at this task. BCQ, however, is able to strongly outperform the performance of the noisy demonstrator, disentangling poor and expert actions. Furthermore, compared to current deep reinforcement learning algorithms, such as the DDPG demonstrator in Figure 3.c, BCQ attains a high performance in remarkably few iterations. As DDPG is known to be data-efficient compared to other state of the art algorithms (Henderson et al., 2017), this suggests our approach effectively leverages expert transitions.

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 methods such as fitted Q-iteration have used decision trees (Ernst et al., 2005) and neural networks (Riedmiller, 2005) but come without convergence guarantees. Decoupling exploration and exploitation has been previously attempted by achieving sufficient diversity in data in an exploratory phase (Colas et al., 2018). Due to fewer assumptions on the data collection, off-policy algorithms which rely on importance sampling (Precup et al., 2001; Munos et al., 2016) may not be applicable in a batch setting, and scale poorly to multi-dimensional action spaces. Reinforcement learning with the experience replay (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 larger experience replay (Lin, 1992) 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 agent was strongest when the distribution of data in the replay buffer matches 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). In recent years, combining imitation with reinforcement learning, via learning from demonstration (Kim et al., 2013; Piot et al., 2014; Chemali & Lazaric, 2015) has grown in popularity with extensions to deep reinforcement learning (Hester et al., 2017; Večerík et al., 2017). While effective, these methods require either explicit labeling of expert data, or further on-policy data collection. Research in imitation and inverse reinforcement learning, with robustness to noise is an emerging area (Evans, 2016; Nair et al., 2017), but require expert data. Gao et al. (2018) introduced an imitation learning algorithm which learned from highly imperfect demonstrations, by favoring previously seen actions, but is limited to discrete actions.

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. It has been previously noted that non-linear function approximation itself induces an implicit exploration without an explicit exploration policy (Dauparas et al., 2018). Our results suggest that this exploration may be the result of an extrapolation error due to unseen state-action pairs. In model-based reinforcement learning, uncertainty has been used for exploration, but also the opposite effect–to push the policy towards regions of certainty in the model. This is present in policy search methods such as PILCO (Deisenroth & Rasmussen, 2011; Gal et al., 2016; Higuera et al., 2018), combined with trajectory optimization (Chua et al., 2018), or value-based methods (Buckman et al., 2018), to combat the well-known problems with compounding model errors. Our work connects to policy methods with conservative updates (Kakade & Langford, 2002), such as trust region (Schulman et al., 2015; Achiam et al., 2017) and information-theoretic methods (Peters & Mülling, 2010; Van Hoof et al., 2017) which aim to keep an 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 an off-policy variant, where the policy aims to be kept close, in output space, to any combination of previous policies which performed the data collection.

7 Conclusion

In this work we demonstrate a critical problem in off-policy reinforcement learning with finite data and function approximation, where the value target introduces error by including an estimate of unseen data. This phenomenon, which we denote *extrapolation error*, has important implications for batch and off-policy reinforcement learning, as it is generally implausible to have complete state-action coverage in any practical setting. We derive batch-constrained reinforcement learning–acting 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 capable of learning from arbitrary batch data, without assumptions on the collection procedure.

References

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

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

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

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

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

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

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1039–1048. PMLR, 2018.

Justas Dauparas, Ryota Tomioka, and Katja Hofmann. Depth and nonlinearity induce implicit exploration for RL. *arXiv preprint arXiv:1805.11711*, 2018.

Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*, 2015.

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

Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *AAAI/IAAI*, pp. 761–768, 1998.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

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

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

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1587–1596. PMLR, 2018.

Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*, 2016.

Yang Gao, Ji Lin, Fisher Yu, Sergey Levine, Trevor Darrell, et al. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

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

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. *arXiv preprint arXiv:1709.06560*, 2017.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, et al. Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017.

Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek. Synthesizing neural network controllers with probabilistic model based reinforcement learning. *arXiv preprint arXiv:1803.02291*, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

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

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

Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, pp. 2859–2867, 2013.

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

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

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

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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

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

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

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. *arXiv preprint arXiv:1709.10089*, 2017.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Vlad Mnih. The uncertainty Bellman equation and exploration. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 3839–3848. PMLR, 2018.

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

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pp. 4026–4034, 2016.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *arXiv preprint arXiv:1806.03335*, 2018.

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

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

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

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Martin Riedmiller. 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.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, 2011.

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

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

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

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

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

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

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

Ahmed Touati, Harsh Satija, Joshua Romoff, Joelle Pineau, and Pascal Vincent. Randomized value functions via multiplicative normalizing flows. *arXiv preprint arXiv:1806.02315*, 2018.

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

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

Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

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

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

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pp. 1433–1438, 2008.

A MISSING PROOFS

Lemma 1 (Batch Bellman Policy Evaluation). *For a given batch $\mathcal{B}$, MDP M, and batch-constrained policy $\pi \in \Pi_{\mathcal{B}}$, the batch Bellman operator $\mathcal{T}^{\pi}_{\mathcal{B}}$ repeatedly applied to an initial $Q_0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $Q_{k+1} = \mathcal{T}^{\pi}_{\mathcal{B}} Q_k$ converges to a fixed point $Q^{\pi}_{\mathcal{B}}(s,a)$ where $Q^{\pi}_{\mathcal{B}}(s,a) = Q^{\pi}(s,a)$ for $(s,a) \in \mathcal{B}$, as $k \to \infty$.*

Proof. Define a new MDP $M_{\mathcal{B}}$ by restricting the full MDP M to a sub-MDP with a set of states and actions with respect to $\mathcal{B}$, then the theorem follows naturally from the contraction proof of the Bellman operator (Bertsekas & Tsitsiklis, 1996).

Theorem 1 (Batch Bellman Necessary and Sufficient Condition). *Let $Q^{\pi}_{\mathcal{B}}(s,a)$ be the fixed point of the batch Bellman operator $\mathcal{T}^{\pi}_{\mathcal{B}}$, then $Q^{\pi}_{\mathcal{B}}(s,a) = Q^{\pi}(s,a)$ for all $(s,a) \in \mathcal{B}$ if and only if $\pi \in \Pi_{\mathcal{B}}$.*

Proof. ($\Rightarrow$) Follows from the definition of $\mathcal{T}^{\pi}_{\mathcal{B}}$, as if $\pi \notin \Pi_{\mathcal{B}}$ then for some s we have $(s, \pi(s)) \notin \mathcal{B}$ and it follows that $Q^{\pi}_{\mathcal{B}}(s, \pi(s)) = \infty$. ($\Leftarrow$) Follows directly from Lemma 1.

Lemma 2 (Batch-Constrained Policy Improvement). *Let $\pi \in \Pi_{\mathcal{B}}$ be any deterministic batch-constrained policy and $\pi'(s) = \operatorname{argmax}_{a \text{ s.t.} (s,a) \in \mathcal{B}} Q^{\pi}(s,a)$, then for all $s \in \mathcal{B}$, $Q^{\pi'}(s, \pi'(s)) \geq Q^{\pi}(s, \pi(s))$.*

Proof. Follows from policy improvement theorem (Sutton & Barto, 1998) by restricting the MDP to a sub-MDP with the set of states and actions contained in $\mathcal{B}$ and noting $Q^{\pi}(s, \pi^*_{\mathcal{B}}(s)) \geq Q^{\pi}(s, \pi(s))$ for all $s \in \mathcal{B}$.

Theorem 2 (Batch-Constrained Policy Iteration). *Given a batch $\mathcal{B}$ and MDP M, then the repeated application of policy evaluation and batch-constrained policy improvement converges to a policy $\pi^*_{\mathcal{B}}$, such that $Q^{\pi^*_{\mathcal{B}}}(s, \pi^*_{\mathcal{B}}(s)) \geq Q^{\pi_{\mathcal{B}}}(s, \pi_{\mathcal{B}}(s))$ for all $\pi_{\mathcal{B}} \in \Pi_{\mathcal{B}}$ and $s \in \mathcal{B}$.*

Proof. Denote π_i the policy at iteration i. By Lemma 2, we have the monotonically increasing sequence $Q^{\pi_{i+1}} \geq Q^{\pi_i}$. Noting that Q is bounded above as r is bounded, then policy iteration must converge to some maximum $Q^{\pi'}$, with corresponding policy $\pi' \in \Pi_{\mathcal{B}}$, such that $\pi'(s) = \operatorname{argmax}_{a \text{ s.t.} (s,a) \in \mathcal{B}} Q^{\pi'}(s,a)$ for all $s \in \mathcal{B}$. It follows that for all $\pi \in \Pi_{\mathcal{B}}$ we have $\mathcal{T}^{\pi}_{\mathcal{B}} Q^{\pi'} \leq Q^{\pi'}$, otherwise contradicting the definition of π'. Then from a repeated application of Lemma 1 we have that $Q^{\pi} \leq Q^{\pi'}$, and π' must be the optimal batch-constrained policy.

Corollary 1 (Batch-Constrained Optimality). *Given a batch $\mathcal{B}$, and MDP M if for all $s \in \mathcal{S}$, $(s, a^*, r, s') \in B$ where $a^* = \operatorname{argmax}_a Q^*(s,a)$, then batch-constrained policy iteration converges to a policy $\pi^* = \operatorname{argmax}_a Q^*(s,a)$ for all $s \in \mathcal{S}$.*

Proof. Follows directly from Theorem 1 as $\pi^* \in \Pi_{\mathcal{B}}$.

Corollary 2 (Batch-Constrained Imitation). *Given a deterministic MDP, and trajectory $T = (s_0, a^*_0, r_0, ..., s_{i+1}, a^*_{i+1}, r_{i+1}, ...)$ where $a^*_i = \operatorname{argmax}_a Q^*(s_i, a)$, then batch-constrained policy iteration converges to $\pi^*(s_i) = \operatorname{argmax}_a Q^*(s_i, a)$ for all $s_i \in T$.*

Proof. Follows directly from Theorem 1 by setting $\mathcal{B} = T$.

B 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 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 settings as defined in Supplementary Material C are used.

Batch 1 (Fixed 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. Experiment 2 differs from Experiment 1 as the agents are trained with the same version of the buffer, while in Experiment 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 (Robust Imitation). The expert policies from experiment 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.

C Implementation Details

Across all methods and experiments, for fair comparison, each network generally uses the hyper-parameters and architectures, these are defined in Table 1 and Figure 5 respectively. Critics and 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:

$$y = \begin{cases} r & \text{if terminal } s' \\ r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s')) & \text{else} \end{cases} \tag{14}$$

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
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	$\mathcal{N}(0, 0.1)$

BCQ. BCQ uses four networks: an actor $\pi_\phi(s, a)$, a critic $Q_\theta(s, a)$, a value network $V_\psi(s)$, and a state-conditioned VAE $G_\omega(s)$, along with a target value network $V_{\psi'}(s)$. Each network in BCQ follows the default architecture (Figure 5) and default hyper-parameters found in Table 1.

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

Figure 5: Default Network Architecture. All actor networks are followed by a `tanh` $\cdot$ `max action size`

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. The VAE is trained with respect to the mean-squared error of the reconstruction along with a KL regularization term:

$$\mathcal{L}_{\text{reconstruction}} = \sum_{(s,a)\in\mathcal{B}} (D_{\omega_2}(s, z) - a)^2, \quad z = \text{clip}(\mu + \sigma \cdot \epsilon, -0.5, -0.5), \quad \epsilon \sim \mathcal{N}(0, 1), \tag{15}$$

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

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

To limit generalization beyond previously seen actions, the latent vector is clipped to a range of $[-0.5, 0.5]$. Noting the Gaussian form of both distributions, the KL divergence term can be simplified (Kingma & Welling, 2013):

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

where J denotes the dimensionality of z. For each experiment we use $J = 2 \cdot$ action dimension. To normalize $\mathcal{L}_{\text{VAE}}$ across experiments we use $\lambda = \frac{1}{J}$. While the choice of J is sufficiently large to exactly reconstruct any possible action, we found that when z was clipped to a sufficiently small range the VAE only sampled previously seen actions. This is exemplified by the experimental results of the VAE behavioral cloning method.

In the value network update (Equation (13)), the VAE is sampled multiple times and passed to both the actor and critic. 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 actor and critic. When running the agent in the environment, we sample from the VAE 100 times.

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 network at the second layer, (Figure 6), 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 6: DDPG Critic Network Architecture.

As done in 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.

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_i) for $i = \{1, ..., J\}$. In each DQN update, all state-sub-action pairs (s, a_i) are updated with respect to the average value of the target state-sub-action pairs (s', a'_j). The learning update of the discretized DQN is as follows:

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

$$\theta = \underset{\tilde{\theta}}{\operatorname{argmin}} \sum_{(s,a,r,s') \in \mathcal{B}} \sum_{i} (y - Q_{\tilde{\theta}}(s, a_i))^2. \tag{20}$$

For clarity we provide the exact DQN network architecture in Figure 7.

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

Figure 7: 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.

D Variational Auto-Encoder Background

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:

$$\log p(X) \geq \mathbb{E}_{t(X|z)}[\log p(X|z)] - D_{\text{KL}}(q(z|X)||p(z)), \tag{21}$$

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 the 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):

$$\mathbb{E}_{z \sim \mathcal{N}(\mu,\sigma)}[f(z)] = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}[f(\mu + \sigma\epsilon)]. \tag{22}$$

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.