True Off-Policy Batch Constrained Reinforcement Learning

Samaahita Belavadi
A53284059
Department of Electrical and Computer Engineering
University of California, San Diego
sbelavad@ucsd.edu

Sutej Kulgod
A53264802
Department of Electrical and Computer Engineering
University of California, San Diego
skulgod@ucsd.edu

***Abstract*—Batch constrained reinforcement learning is a particularly interesting reinforcement learning domain since it offers the possibility of learning from a batch of data while requiring no interaction with the environment. Since standard off-policy reinforcement learning algorithms like TD3 and DDPG fail in a batch constrained setting, a new class of algorithms that can learn in this true off-policy setting have been introduced. In this paper, a novel batch constrained Q-learning algorithm is introduced in which the action space is restricted to push the agent towards taking actions present in the batch while also making sure that expected rewards are maximized. This ensures that while the agent mostly stays close to the data in the batch, when required the agent is allowed to maximize the Q-value function by taking actions which are farther away from the batch. This approach is evaluated against Batch Constrained Q-learning (BCQ), which is a state of the art algorithm in a batch constrained setting. Experimental results over several tasks prove that the presented approach outperforms BCQ, especially when presented with data that contains mostly poor actions.**

***Index Terms*—Reinforcement Learning, Batch Constrained Q-learning**

I. Introduction

Many practical applications of reinforcement learning involve learning from a fixed batch of data that has been collected by an expert or some other policy without offering further possibilities for data gathering. Most modern off-policy deep reinforcement learning algorithms like DDPG [1] and TD3 [2] fall into the category of growing batch learning, in which data is collected and stored into an experience replay dataset, which is used to train the agent before further data collection occurs. However, these algorithms fail while training if they are constrained to a fixed batch of data, without access to the environment due to *extrapolation error* as elaborated in [3]–[6]. Hence, there is a necessity for a new class of algorithms which are suitable for batch constrained settings.

Current off-policy deep reinforcement learning algorithms select actions with respect to a learned Q-value [7] estimate, without consideration of the accuracy of the estimate. As a result, certain actions which are not present in the batch of data can be erroneously extrapolated to higher Q-values in a fixed data setting. This reduction in performance can be seen in Fig. 1, where two DDPG agents are trained using the same data with orange agent having access to interact with the environment, while the blue agent does not get to interact with the environment. However, as shown by Fujimoto et al. [3], the value of an off-policy agent can be accurately evaluated only in the regions where data is available. Keeping this in mind, Batch Constrained Q-learning (BCQ) [3], Bootstrapping Error Accumulation Reduction (BEAR) [4] and Behavior Regularized Actor Critic (BRAC) [5] are formulated, in which the current policy during training is driven towards inducing state-action pairs which are close to the ones present in the batch. These algorithms are shown to outperform state of the art off-policy and imitation learning algorithms in a true off policy setting.

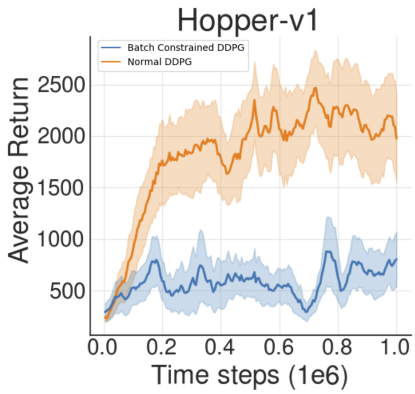

Fig. 1: Performance of two DDPG agents trained in a standard RL setting and in a batch constrained setting [3]

We take inspiration from Batch Constrained Q-Learning proposed by Fujimoto et al. in [3] for our work. Due to the hard constraint of choosing actions that are close to the data present in the batch, BCQ fails to generalize and chooses a suboptimal action when presented with data containing only suboptimal actions. In order to solve this problem, an improved batch constrained algorithm is proposed which can learn to choose good actions that may not be present in the batch of data.

II. Background

Batch constrained reinforcement learning involves learning the reward maximising behavior using an agent which cannot interact with the environment and only has access to data generated by some other policy. The data consists of tuples of state of the environment s, selected action a, reward obtained from the environment r and next state s' i.e. (s, a, r, s'). The goal of the agent is to maximize the

expectation of the sum of discounted rewards, known as the return $R_t = \sum_{i=t+1}^{\infty} \gamma^i r(s_i, a_i, s_{i+1})$, which weighs future rewards with respect to the discount factor $\gamma \in [0, 1)$.

Q-learning is an off-policy algorithm, meaning the target can be computed without consideration of how the experience was generated [8]. For Q-learning in large or continuous state and action space, Deep Q-networks are used as function approximators [9] and the loss is minimized over mini-batches of tuples of the agent's past data, $(s, a, r, s') \in \mathcal{B}$ sampled from an experience buffer $\mathcal{B}$. Hence, for batch constrained deep reinforcement leaning the batch $\mathcal{B}$ is fixed and no more addition of data is allowed. In recent work, to further stabilize learning, a target network $Q_{\theta'}$ with frozen parameters θ', is used as the learning target. The parameters of the target network are updated using the following equation:

$$\theta_i' \leftarrow \tau\theta + (1 - \tau)\theta_i' \tag{1}$$

where τ is constant and the value of the Q-network is updated using the target:

$$y \leftarrow r + \gamma Q_{\theta'}(s', \pi(s')) \tag{2}$$

Since the function approximation is done for a Q-network, the final policy can be obtained by determining the action a which maximises the Q value at the given state s. The maximum Q value for s is determined by sampling a few possible actions and taking argmax over the Q-value at these state-action pairs as shown in the equation below:

$$\pi(s) \leftarrow \text{argmax}_a Q_\theta(s, a) \tag{3}$$

It can be seen from (1) and (2) that an error in Q-value estimation gets carried over to the target network and hence the Q-network from the following iterations. The estimation of Q-value is poor in the regions where data is not present, and this phenomenon is explained by Fujimoto et al. in [3] and is labeled as extrapolation error. Extrapolation error is an error in off-policy value learning which is introduced by the mismatch between the dataset and true state-action visitation of the current policy. The cause of extrapolation error can be attributed to the following factors:

- Absent Data: Any state-action pair (s, a) not available in the batch data $\mathcal{B}$ which results in arbitrarily bad $Q_\theta(s', \pi(s'))$
- Model Bias: Bias originating due to computing the value of $Q(s, a)$ with respect to the batch $\mathcal{B}$ and not with the true MDP
- Training Mismatch: Bias corresponding to the difference in the distribution of actions in the batch and its distribution under the current policy during training.

To avoid extrapolation error and to optimise off-policy learning for a given batch, the policies are trained to choose actions based on the following conditions:

- Minimize the distance of the selected actions to the data in the batch.
- Lead to states where familiar data can be observed.
- Maximize the value function.

A variational auto-encoder (VAE) [10] is a generative model which aims to maximize the marginal log-likelihood $\log p(X) = \sum_{i=1}^{N} \log p(x_i)$ where $X = \{x_1, \ldots, x_N\}$, is the dataset. A VAE consists of an encoder E_{ω_1} which generates a mean μ and variance σ of a normal distribution using an input x. A value sampled from this distribution $\mathcal{N}(\mu, \sigma)$ is passed through a decoder D_{ω_2} which is trained to recover the value x.

III. Problem Definition

Given a batch $\mathcal{B}$ containing tuples of state s, action a, reward r and next state s' i.e. (s, a, r, s'), obtain a policy π^* which maximises the value $V^\pi(s)$ for all s in state space $\mathcal{S}$.

$$\pi^* = \arg\max_\pi V^\pi(s), \qquad \forall s \in \mathcal{S} \tag{4}$$

IV. Methods

As described in section II, we develop an algorithm in which the actions generated are close to the ones present in the dataset, while exploring the vicinity to obtain higher long term rewards. A similarity metric is introduced by assuming that for a given state s and action a, the similarity between (s, a) and state-action pairs in the batch $\mathcal{B}$ can be modelled as $P_{\mathcal{B}}^G(a|s)$. Since estimating $P_{\mathcal{B}}^G(a|s)$ for a high-dimensional continuous space is difficult, a parametric generative model $G_\omega(s)$ is trained and is used to sample actions as a reasonable approximation to $argmax_a P_{\mathcal{B}}^G(a|s)$.

A variational auto-encoder (VAE) is used to model $G_\omega(s)$. The update equation for the VAE is used to control the importance given to attaining higher Q-value given by $\sum Q_{\theta_1}(s, G_\omega(s))$ and staying close to the data present in the batch given by $\sum(a - \tilde{a})^2 + D_{\text{KL}}(\mathcal{N}(\mu, \sigma) \| \mathcal{N}(0, 1))$ while generating an action. These characteristics are controlled by the parameters k_1 and k_2 as seen in the update equation (5), where $\tilde{a}$ is the action generated from the VAE:

$$\begin{aligned}\omega \leftarrow \text{argmin}_\omega -k_1 \sum Q_{\theta_1}(s, G_\omega(s)) + k_2 \sum (a - \tilde{a})^2 \\ + D_{\text{KL}}(\mathcal{N}(\mu, \sigma) \| \mathcal{N}(0, 1))\end{aligned} \tag{5}$$

A target VAE with parameters ω' is used to decrease variance in the estimated action a. The update for the target network is similar to the equation (1), and the same constant τ is used. The update equation is given by:

$$\omega' \leftarrow \tau\omega + (1 - \tau)\omega' \tag{6}$$

For the Q-networks, Modified Clipped Double Q-learning is used. This is done to penalize uncertainty over future states, as the minimum operator penalizes high variance estimates in the region of uncertainty and pushes the policy towards taking actions which produce states in the known regions. A convex combination of the minimum and maximum values between the two target Q-values is taken to form a learning target y which is used by both the Q-networks.

$$y \leftarrow r + \gamma[\lambda \min_{j=1,2} Q_{\theta_j'}(s', a') + (1 - \lambda) \max_{j=1,2} Q_{\theta_j'}(s', a')] \tag{7}$$

The degree of penalization of uncertainty at future states can be set by choosing λ and $\lambda = 1$ corresponds to Clipped Double

Q-learning where the penalty is highest and corresponds to the vanilla Clipped Double Q-Learning.

Algorithm 1 describes the proposed Improved BCQ algorithm. It can be seen in the algorithm that Q-network and VAE are initialised to random and these wights are used to initialise their corresponding target networks. The algorithm iterates for T epochs as seen in line 1. Line 2 describes the sampling from the batch $\mathcal{B}$. Line3-4 describe updating the VAE. Line 5-7 describe sampling actions a' from the target VAE and using that in obtaining the value target y and updating the two Q network parameters θ_1 and θ_2. Line 8-9 describe the updation of the Q-network targets and the VAE target.

Algorithm 1 Improved BCQ

Input: Batch $\mathcal{B}$, training epochs T, target network update rate τ, mini-batch size N, minimum weighting λ, weights for VAE loss function k_1 and k_2.
Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$ with random parameters $\theta_1\theta_2, \omega$, and target networks $Q_{\theta'_1}, Q_{\theta'_2}, G_{\omega'}$ with $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\omega' \leftarrow \omega$.

1: **for** $t = 1$ to T **do**
2: Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
3: $\mu, \sigma = E_{\omega_1}(s, a)$, $\tilde{a} = D_{\omega_2}(s, z)$, $z \sim \mathcal{N}(\mu, \sigma)$
4: $\omega \leftarrow \text{argmin}_\omega -k_1 \sum Q_{\theta_1}(s, G_\omega(s)) + k_2 \sum (a - \tilde{a})^2 + D_{\text{KL}}(\mathcal{N}(\mu, \sigma) \| \mathcal{N}(0, 1))$
5: $a' \sim G_{\omega'}(s')$
6: Set value target: $y \leftarrow r + \gamma[\lambda \min_{j=1,2} Q_{\theta'_j}(s', a') + (1 - \lambda) \max_{j=1,2} Q_{\theta'_j}(s', a')]$
7: $\theta_i \leftarrow \text{argmin}_{\theta_i} \sum \left(y - Q_{\theta_i}(s, a)\right)^2$ for $i = 1, 2$
8: Update target networks: $\theta'_i \leftarrow \tau\theta + (1 - \tau)\theta'_i$ for $i = 1, 2$
9: $\omega' \leftarrow \tau\omega + (1 - \tau)\omega'$ for $i = 1, 2$

V. Results

To show the effectiveness of our algorithm in high dimensional continuous space setting, we use MuJoCo [11] environments in OpenAI Gym [12]. We compare our algorithm with Batch Constrained Q-Learning (BCQ) which is the current state of the art. Two different batch settings are used on "Reacher-v2" and "Hopper-v2" environments are used to make the comparison. The two batch settings are as follows:

1) **Concurrent Batch**: Buffer data of size 10^6 is collected and stored while training a TD3 behavioral agent.
2) **Reversed Batch**: Buffer data of size 10^6 is generated and stored using a trained TD3 behavioral agent. This data is generated by obtaining an action a from the trained TD3 policy and reversing it in a particular region of state space with a probability of 0.9. This is done to ensure reversed data is present even from the inner parts of the selected region. The reverse action $-a$ is passed through the simulation environment to generate the tuple $(s, -a, r, s')$ which is stored in the dataset. It is made sure that at least 20% of the data contains actions opposite to the ones obtained from the behavioral agent. This is done to ensure that there is sufficient amount of reversed data in the batch.

The parameter values used for training and evaluation are shown in table I.

Parameter	Value
λ	0.75
k_1	0.05
k_2	0.5
τ	0.005
γ	0.99
N	100

TABLE I: Table of parameter values

λ is chosen to be 0.75 so that higher weight is given to the minimum Q-value to avoid over-estimation bias while also giving sufficient weight to the maximum Q-value instead of ignoring the maximum Q-value as in standard double clipped Q-learning. k_1 is set to a small value of 0.05 and k_2 is set to a higher value of 0.5 so that the reconstruction error of the VAE is given more importance than maximizing Q-value. This ensures that more weight is given to reducing extrapolation error while also giving a chance for the agents to explore actions with higher Q-values.

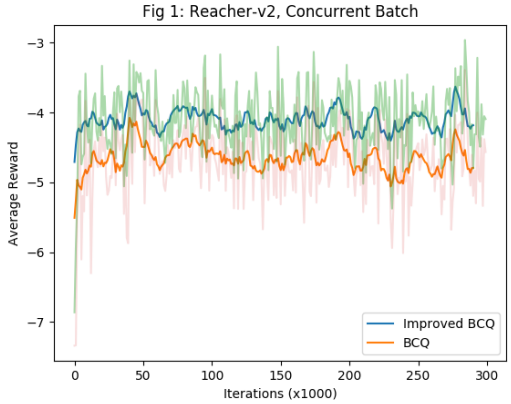

Fig. 2: Average reward vs training iterations for Concurrent batch setting on Reacher-v2 environment

It can be seen from the above results that the proposed algorithm outperforms BCQ for both the batches in both the environments. In the concurrent batch setting, as expected both algorithms have similar performance with the improved BCQ algorithm having slightly better performance. This is because, in improved BCQ, as the VAE is trained while keeping in mind that Q-values need to be maximized, in general, the sampled actions from the VAE have better rewards than the sampled actions obtained in BCQ.

In the reversed batch setting, it is observed that the proposed algorithm has a significant improvement in performance over BCQ. This is because, even in regions where only reversed actions are available i.e. the data only specifies what not to do, the BCQ algorithm picks the relatively better action from among the poor actions in the dataset. On the other hand, since in the proposed algorithm the VAE does not have a hard

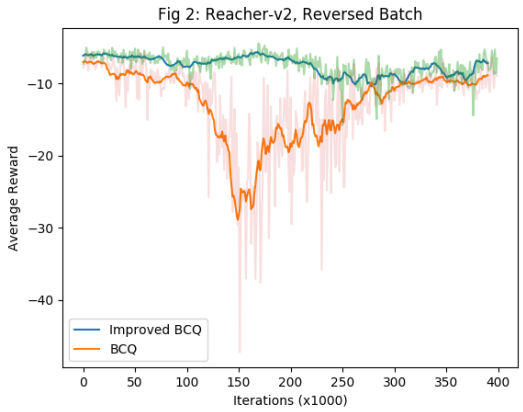

Fig. 3: Average reward vs training iterations for Reversed batch setting on Reacher-v2 environment

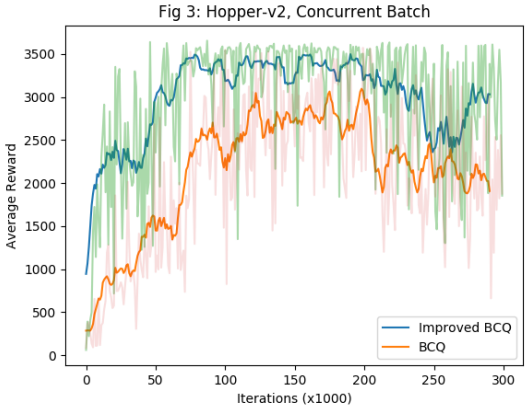

Fig. 4: Average reward vs training iterations for Concurrent batch setting on Hopper-v2 environment

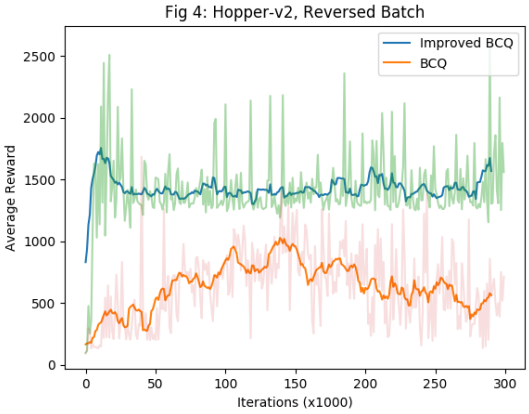

Fig. 5: Average reward vs training iterations for Reversed batch setting on Hopper-v2 environment

constraint to choose actions close to the ones in the dataset, it attempts to maximize the Q function and picks an action with better long term rewards even though the action is farther away from the dataset.

VI. Conclusion

In this work, we demonstrate a problem in batch constrained off-policy learning algorithms where when a batch of data contains only poor actions for a particular state, algorithms like BCQ still pick one of the poor actions since they have a hard constraint of taking actions which are close to the batch. This has important implications for off-policy and batch reinforcement learning as, in any practical setting, it is generally implausible to ensure that good actions are present in the batch for all states in the state space.

We present an improved batch-constrained reinforcement learning algorithm in which the hard constraint of taking actions that are close to the batch is relaxed. Instead, the agent is trained to trade off between maximizing Q-values and staying close to the batch of data, where higher importance is given to the latter to ensure that extrapolation error is minimized. This improved BCQ algorithm is able to generalize to states where only poor actions are present in the batch of data. The performance of our approach is shown to surpass the state of the art BCQ algorithm, both in a setting where the batch contains only good actions and in a setting where the batch contains poor actions for some states. One possibility for improving the algorithm further is to replace the variational auto-encoder with an approximator which can select an action for a given state without random sampling.

References

[1] 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. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[2] 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, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1582–1591, 2018.

[3] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 2052–2062, 2019.

[4] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. *arXiv e-prints*, page arXiv:1906.00949, Jun 2019.

[5] Yifan Wu, George Tucker, and Ofir Nachum. Behavior Regularized Offline Reinforcement Learning. *arXiv e-prints*, page arXiv:1911.11361, Nov 2019.

[6] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for Simplicity in Off-Policy Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1907.04543, Jul 2019.

[7] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. The MIT Press, 2018.

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

[10] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019.

[11] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

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