[Re] Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction

Seungwon Kim
Incheon International Airport Corporation,
Georgia Institute of Technology
skim3222@gatech.edu

1 Introduction

Recent Reinforcement Learning(RL) algorithms have achieved great success on different tasks with the comparable performance with humans or sometimes surpassing human ability. Nevertheless, compared to the area of computer vision or NLP which can learn from large-scale datasets and have a generalizable model, the application of RL algorithms to the real-world is still limited since it requires active data collection. In principle, Off-policy RL algorithms such as Q-Learning can remedy this issue since it can learn from the dataset collected from any policy. However, in practice, off-policy Q-Learning is so sensitive to the training data distribution that it often fails to learn without data collection that directly comes from the interaction with environments.

The authors have investigated off-policy Q-learning which learns from static dataset and identified the instability of Q-learning is due to bootstrapping error, which results from selecting action lies out-of-distribution of the dataset, in Bellman backup operator [4]. The authors theoretically analyze bootstrapping error and suggest a way to reduce bootstrapping error by carefully constraining action selection in backup operation. Finally, the authors present a practical algorithm, bootstrapping error accumulation reduction (BEAR), which is a robust off-policy Q-learning algorithm, and empirically show that BEAR Q-Learning (BEAR-QL) is able to learn from a variety of datasets including suboptimal and random datasets in continuous control tasks.

In this work, we reproduce the main results of the paper, Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction, including the performance of baseline algorithms as well as BEAR-QL [4]. We analyze and compare our results with those in the paper and provide practical suggestions to reproduce the results of the paper. The implementation of BEAR-QL is done with Tensorflow and we make the code and dataset available online [1].

2 Background

In reinforcement learning, Markov Decision Process (MDP) can be expressed as a tuple of (S, A, P, R), which consists of state S, action A, reward function R, and trainsition distirubtion P that is probability distribution of next state S' given that action a is executed in state S, i.e., $P(s'|s, a)$. The goal of reinforcement learning is to maximize expected return, i.e., $\sum_{t=1} \gamma^{t-1} R_t$, where reward R at time t is discounted by discount factor γ that is between 0 and 1. In continuous action spaces, the objective can be transformed into finding the optimal policy π parameterized by θ that maximizes

[1] https://github.com/seungwon1/BEAR-QL

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

expected return as below equation (1). Here τ is the trajectory of states and actions induced by π, and $p_\theta(\tau)$ is the distribution of τ defined as $p_\theta(\tau) = p(s_1) \prod_{t=1} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$

$$\theta^* = arg\max_\theta \mathbb{E}_{\tau\sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] \tag{1}$$

Q-learning is off-policy RL algorithm that allows target policy which is different from behavior policy and the optimal Q function, state-action value $Q^*(s, a)$, can be learned by iteratively updating Q functions using Bellman optimality equation with backup operator T in equation (2).

$$(T\hat{Q}) := R(s, a) + \gamma\, \mathbb{E}_{T(s'|s,a)}[\max_{a'} \hat{Q}(s', a')] \tag{2}$$

On the other hand, equation (2) is only valid when the action space is discrete. When the state or action is in continuous space, it is necessary to approximate state or action to estimate Q functions or directly finding the optimal policy π^*. Deep neural networks can be used to approximate $Q(s, a)$ and the policy, $\pi(\cdot|s)$, which directly outputs action values if the policy is deterministic or mean and variance of the policy function, typically Gaussian function, if the policy is stochastic.

3 Analysis of BEAR Q-Learning

The authors point out that Q-learning fails to learn with a static dataset due to the bootstrapping error. Bootstrapping error comes from the inappropriate action selection when performing Bellman update. As shown in equation (2), the action should be selected to estimate the Q function in the next state, $\hat{Q}(s', a')$. However, since the action is selected based on the current Q function, a' does not necessarily lie in the distribution of the training set and it is more likely to be far out of distribution of the dataset if one continually updates Q-values. Out-Of-Distribution(OOD) action in Q-learning makes learning process unstable, resulting in poor performance.

To mitigate this problem, BEAR Q-Learning makes the policy, $\pi(\cdot|s)$, output actions that lie on the support of the training set. Note that this is not the same as Batch Constrained Q-Learning (BCQ), which constrains the policy more strictly to be close to the distribution of the training set [1]. It is less strict than BCQ, and it has benefits to find better solutions than BCQ does especially given that the training set is suboptimal. Consider that the training set is collected by a random uniform policy. In this case, the policy of BEAR Q-Learning is less restrictive than those of BCQ and it is more likely to find better solutions by learning from training sets, whereas BCQ tends to imitate the distribution of training set. The overall description of BEAR-Q Learning built on actor-critic style is presented in algorithm 1. The description of algorithm 1 is slightly different from those in the paper. Details and specific architectures of the network are provided to help reproducibility.

Algorithm 1 BEAR Q-Learning (BEAR-QL)

Input: Dataset D, target network update rate τ , mini-batch size N , sampled actions p, m, n, MMD threshold ϵ, minimum λ

1: Initialize Q-ensemble $\{Q_{\theta_i}\}_{i=1}^K$, actor π_ϕ , VAE network $G_w = \{E_{w1}, D_{w2}\}$, Lagrange multiplier α, target networks $\{Q_{\theta'_i}\}_{i=1}^K$, and a target actor $\pi_{\phi'}$ with $\phi' \leftarrow \phi$, $\theta'_i \leftarrow \theta_i$
2: **for** t in $\{1, ..., N\}$ **do**
3: Sample mini-batch of transitions $(s, a, r, s') \sim D$
4: $\mu, \sigma = E_{w1}(s, a)$, $\tilde{a} = D_{w2}(s, z)$, $z \sim N(\mu, \sigma)$
$w \leftarrow arg\min_w \sum (a - \tilde{a})^2 + D_{KL}(N(\mu, \sigma)||N(0, 1))$
Q-update:
5: Sample p action samples, $\{a_i \sim G_w(s')\}_{i=1}^p$
6: Define $y(s, a) := \max_{ai}[\lambda \min_{j=1,...,K} Q_{\theta'_j}(s', a_i) + (1 - \lambda) \max_{j=1,...,K} Q_{\theta'_j}(s', a_i)]$
7: $\forall i, \theta_i \leftarrow arg\min_{\theta_i}(Q_{\theta_i}(s, a) - (r + \gamma y(s, a))^2$
Policy-update:
8: Sample action samples, $\{\hat{a_i} \sim \pi_\phi(\cdot|s)\}_{i=1}^m$ and $\{a_j \sim G_w(s)\}_{j=1}^n$, m, n between (1-10)
9: Update π, α by minimizing equation 3 using dual gradient descent with lagrange multiplier α
10: **end for**

There are several components used in BEAR Q-Learning. First of all, it uses Variational Auto-Encoder(VAE) to approximate the behavior policy of dataset to sample actions from the dataset and to compute the target function for updating Q functions (line 4, 5, 6). Note that VAE in algorithm 1 is not provided in the original paper [4]. In their paper, there is no specific guidance for training behavior policy of dataset to sample actions and how to sample actions for calculating target $y(s, a)$. However, after examing several networks such as naive Behavior Cloning(BC) and VAE based BC and through communication with authors, VAE-BC turned out to work well and they also confirmed that VAE-BC is used for training behavior policy of the training set D. Using VAE to sample actions for calculating target also makes sense since it reduces bootstrapping error by selecting the action is likely to lie in the distribution of the dataset. The other component is an ensemble of Q functions to estimate target values. This is similar to recent works in Q-learning with the static dataset and stabilizes learning by reducing overestimation bias from the maximization step in Q-learning.

After updating Q functions, actions are sampled from dataset D and online policy networks π_ϕ to update the policy. The policy is updated by minimizing equation 3 using dual gradient descent.

$$\begin{aligned} \pi_\phi(s) = \max_{\pi \in \Pi_\epsilon} \mathbb{E}_{a \sim \pi(\cdot|s)}[Q_k(\hat{s}, a)] - \lambda \sqrt{var_k\, \mathbb{E}_{a \sim \pi(\cdot|s)}[Q_k(\hat{s}, a)]} \\ s.t.\, \mathbb{E}_{s \sim D}[MMD(D(s), \pi(\cdot|s))] \leq \epsilon \end{aligned} \tag{3}$$

The policy update in BEAR-QL is a constrained optimization problem that the objective $\pi_\phi(s)$ is maximized while satisfying the MMD between sample actions. Here, the constraints of the objective is maximum mean discrepancy(MMD), which measures the distance between two sample actions from different distributions. We provide implementation details of this constrained optimization problem using dual gradient descent in section 4.3. In summary, BEAR-QL tries to maximize Q values with respect to current policy while constraining the actions to lie on the support of the distribution of training set, resulting in reducing bootstrapping error and better performance than other methods such as BCQ that has more strict constraints.

4 Reproducibility

4.1 Data generation

The authors made three kinds of datasets: optimal, medium quality, and random datasets; for four different Mujoco environments: HalfCheetah-v2, Walker2d-v2, Hopper-v2, and Ant-v2. The optimal and medium quality dataset are generated by rolling the policy trained by Soft Actor-Critic (SAC) algorithm [3]. They used near-optimal policy to generate the optimal dataset and partially trained policy to generate the medium quality dataset. The random dataset is made by rolling uniform policy. Each dataset contains 1e6 number of tuples and each tuple contains S, A, R, S', and T, where S is state, A is action, R is reward, S' is the next state, and T terminal value that indicates whether the episode terminates. As denoted by the paper, we used the official implementation of Soft Actor-Critic (SAC) algorithm and trained SAC on four environments, logged the performance of the agent in the evaluation stage, and saved all the policy variables trained by SAC for every certain time step interval, e.g., 100 iterations [3]. To make optimal and medium quality datasets, we selected several policy candidates in which the average return during the evaluation stage is close to those of the behavior policy of the dataset presented in the paper. This is because there is no information about the standard deviation of the performance of the behavior policy that generates each dataset and the authors only provided the average return of the behavior policy of the dataset. We rolled out several policies for 100 episodes, picked the best one which has the closest average return with lower standard deviation. We assumed that selecting the dataset has consistent performance between individual trials reduces errors and helps reproducibility. Then, we simulated the final policy and saves the experiences of which consist of (S, A, R, S', T) for 1e6 time steps. For the random dataset, since the standard deviation of the performance of the behavior policy is negligible, we just simulated uniform policy for 1e6 time steps. We provide the average return with a standard deviation ($\pm\%$) of the policy in 1e6 time steps for each dataset in table 1 for future reproducibility.

Quality	HalfCheeta-v2	Walker2d-v2	Hopper-v2	Ant-v2
Optimal	12254 $\pm 2\%$	3146 $\pm 22\%$	2695 $\pm 0.7\%$	5193 $\pm 9\%$
Medium	3995 $\pm 11\%$	510 $\pm 46\%$	1119 $\pm 8\%$	614 $\pm 45\%$

Table 1: Performance of behavior policy in data generation

4.2 Experimental Setup

Using above datasets, we first performed experiments of baseline to match the performance with those in the paper. Baseline experiments were done using the official implementations of BCQ, TD3, and BC-VAE as denoted in the paper [1, 2]. We did not change hyper-parameters as well as network architecture and used the default value, which is the same as those used in the paper for BCQ, BC-VAE, and TD3 [1, 2]. To train BEAR-QL, there is a lack of information for some hyper-parameters. Since BEAR-QL is similar to BCQ, we used similar values for missing hyper-parameters such as learning rate, optimizer, and target update rate and adjusted some others such as the number of actions to sample through communication with authors. Also, we observed that the standard deviation of the Q-values across the ensemble model often makes learning unstable and policy variables diverge. Details of hyper-parameters as well as some clipping values to stabilize learning used in BEAR-QL are summarized in table 2.

Hyper-parameter	value
Number of ensemble k	2
Threshold ϵ for MMD	0.05
Learning rate	1e-3
Optimizer	Adam
Batch size	100
gamma	0.99
λ for estimating target Q	0.75
Target Update Rate τ	5e-3
Number of samples p, m, n	5, 5, 5
Clipping dual variable α	[-5, 10]
Clipping standard deviation of Q functions	[0, 10]

Table 2: Default hyper-parameter used in BEAR-QL

4.3 Implementation details

Here we provide practical implementation details of network architecture for critic and actor, VAE used in BEAR-QL as well as how the dual gradient descent is performed using Lagrange multiplier α. Since BEAR-QL shares similar components with BCQ, the same network architecture used in BCQ is selected for the critic, actor, and VAE network. For critic, there are two separate networks and each output unique value of Q function, $Q(s, a)$, given observation and action. Each Q function network is a 3-layer network where hidden units are 400, 300 respectively with relu activation function except for the output layer. For actor, there is a 3-layer network where hidden units are 400, 300 respectively with relu activation function. It is connected to two individual output components respectively, the mean and standard deviation of gaussian policy. Note that mean and variance have a shared network. The encoder network of VAE is similar to the actor network but different number of hidden units with 750, 750 and it outputs mean and variance of gaussian policy. The decoder network is a 3-layer network outputs action where hidden units are 750, 750. Latent dimension for VAE is twice as big as the dimension of action space.

In algorithm 1, BEAR-QL has to solve a constrained optimization problem using dual gradient descent. Dual gradient descent is a popular technique for maximizing the objective while constraining the policy. The objective and constraints is defined in equation (3). Using equation (3), the Lagrange L with Lagrange multiplier α is defined as

$$L(\phi, \alpha) = -\pi_\phi(s) + \alpha(\mathbb{E}_{s\sim D}[MMD(D(s), \pi(\cdot|s))] - \epsilon) \tag{4}$$

Note that $\pi_\phi(s)$ is flipped in sign since it has to be maximized. We alternately performed gradient descent on Lagrange L with respect to the policy variable ϕ and gradient ascent with respect to Lagrange multiplier α to maximize dual loss $\alpha(\mathbb{E}_{s\sim D}[MMD(D(s), \pi(\cdot|s))] - \epsilon)$. The learning rate for each gradient step is the same as shown in table 2.

5 Results

The results of experiments are shown in Figures 1, 2, 3 for the medium quality dataset, random dataset, optimal dataset respectively. We evaluated the policy of the agent in every 1000 episodes and used the average return of 10 trials for plotting as done in their paper. Here to make the graph more legible, we applied the exponential moving average with window size 50 to ours. The average return of behavior policy of the dataset is plotted as a dashed magneta line in the figure.

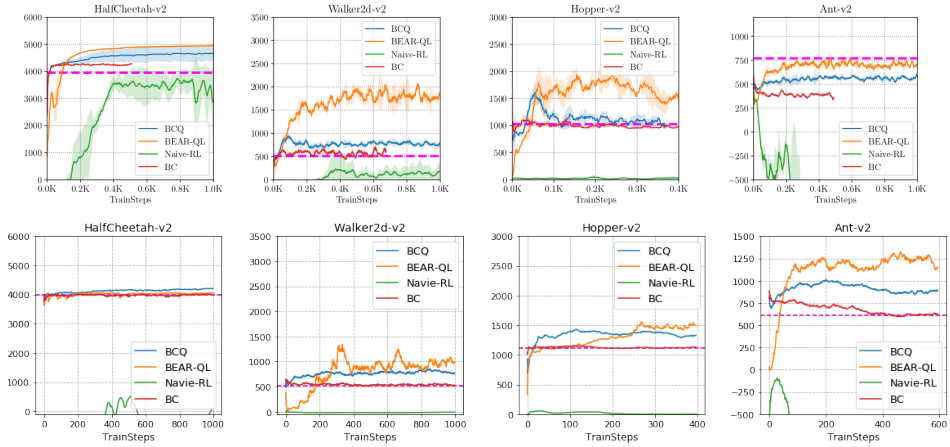

Figure 1: Performance on medium quality dataset. Top (paper) and Bottom (Ours)

As shown in figure 1-bottom, our BEAR-QL outperforms any other baseline algorithms in all tasks except for HalfCheetah-v2. In HalfCheetah-v2, BCQ outperforms BEAR-QL with a small margin, which is different from the original results. We found that it is difficult to make BEAR-QL outperform BCQ in HalfCheetah-v2 since the margin between the performance of BEAR-QL and BCQ is quite small. However, in Ant-v2, our BEAR-QL outperforms all baseline algorithms including BCQ and the behavior policy of dataset by far large margins, demonstrating the main claim of the paper, which is in medium quality dataset, BEAR-QL tries to find a policy that is more superior than behavior policy of dataset, whereas BCQ and BC tend to stay within the data distribution.

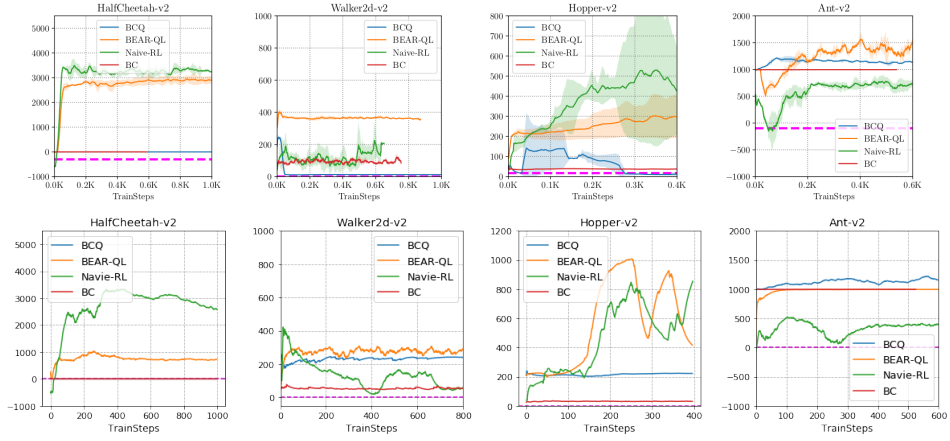

Figure 2: Performance on random dataset. Top (paper) and Bottom (Ours)

Figure 2 shows the performance of algorithms in the random dataset is similar to those in the paper. Although the performance of our BEAR-QL is way better or not comparable to those of the original results in Hopper-v2 and HalfCheetah-v2 on medium quality dataset, it demonstrates that BEAR-QL consistently achieves better results than the average return of dataset in all domains, whereas BCQ performs poorly in some tasks such as HalfCheetah-v2 because of the strict constraints.

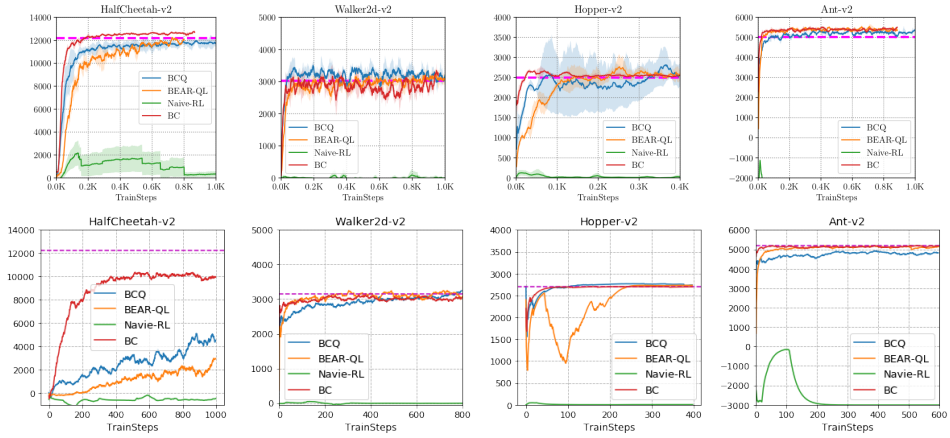

Figure 3: Performance on optimal dataset. Top (paper) and Bottom (Ours)

The performance of BEAR-QL and baseline algorithms in the optimal dataset is presented in figure 3. In all domains, BEAR-QL performs comparably with BCQ and it achieved near-optimal performance except for HalfCheetah-v2. In HalfCheetah-v2, there is a large gap between the performance of BC-VAE and BEAR-QL. Since our BEAR-QL shows better performance in some domains such as Hopper-v2 random dataset or Ant-v2 medium quality dataset, the reason that the performance of BEAR-QL is different from the original results in some domains might result from hyper-parameters or datasets which are different from the paper. Further hyper-parameter tuning or generating additional datasets with a different standard deviation of average return seems to mitigate this issue, resulting in more similar results.

6 Conclusion

In this paper, we study off-policy Q-Learning with static datasets and explored BEAR-QL as well as other baseline algorithms, reproduced the main results in the paper [4]. Although our results are not perfectly matched with the paper, those are roughly same and we empirically show that BEAR-QL can consistently work well on all kinds of datasets in different control tasks: it can learn a better policy from the random dataset, achieve near-optimal or suboptimal performance with optimal or medium quality datasets.

References

[1] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

[2] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[4] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.