Boosting Offline Reinforcement Learning with Residual Generative Modeling

Hua Wei, Deheng Ye, Zhao Liu, Hao Wu, Bo Yuan, Qiang Fu, Wei Yang, and Zhenhui (Jessie) Li

1Tencent AI Lab, Shenzhen, China
2The Pennsylvania State University, University Park, USA

{harriswei, dericye, ricardoliu, alberthwu, jerryyuan, leonfu, willyang}@tencent.com
jessieli@ist.psu.edu

Abstract

Offline reinforcement learning (RL) tries to learn the near-optimal policy with recorded offline experience without online exploration. Current offline RL research includes: 1) generative modeling, i.e., approximating a policy using fixed data; and 2) learning the state-action value function. While most research focuses on the state-action function part through reducing the bootstrapping error in value function approximation induced by the distribution shift of training data, the effects of error propagation in generative modeling have been neglected. In this paper, we analyze the error in generative modeling. We propose AQL (action-conditioned Q-learning), a residual generative model to reduce policy approximation error for offline RL. We show that our method can learn more accurate policy approximations in different benchmark datasets. In addition, we show that the proposed offline RL method can learn more competitive AI agents in complex control tasks under the multiplayer online battle arena (MOBA) game Honor of Kings.

1 Introduction

Reinforcement learning (RL) has achieved promising results in many domains [Mnih et al., 2015; Silver et al., 2017; Kalashnikov et al., 2018; Berner et al., 2019; Vinyals et al., 2019; Ye et al., 2020b; Ye et al., 2020a]. However, being fettered by the online and trial-and-error nature, applying RL in real-world cases is difficult [Dulac-Arnold et al., 2019]. Unlike supervised learning which directly benefits from large offline datasets like ImageNet [Deng et al., 2009], current RL has not made full use of offline data [Levine et al., 2020]. In many real-world scenarios, online exploration of the environment may be unsafe or expensive. For example, in recommendation [Li et al., 2011] or healthcare domains [Gottesman et al., 2019], a new policy may only be deployed at a low frequency after extensive testing. In these cases, the offline dataset is often large, potentially consisting of years of logged experiences. Even in applications where online exploration is feasible, offline data is still beneficial. For example, in strategy video games (Dota or Honor of Kings) that require highly complex action control, current RL methods still learn from the scratch, resulting in a long time to master human-level skills [Berner et al., 2019; Ye et al., 2020c], where there is a large amount of logged replay data from vast players to be utilized.

Although off-policy RL methods [Mnih et al., 2015; Lillicrap et al., 2015; Haarnoja et al., 2018] may be executed in the offline scenario directly, their performance was shown to be poor [Fujimoto et al., 2019] with a fixed batch of data and no interactions with the environment. The poor performance is suspected due to incorrect value estimation of actions outside the training data distribution [Kumar et al., 2019]. The error of estimated values will accumulate and amplify during the Bellman backup process in RL. Typical off-policy RL would improve the learned policy by trying out the policy in the environment to correct the erroneous estimations, which is not applicable in purely offline scenarios.

Instead of correcting erroneous estimations by interacting with the environment, offline RL provides an alternative for the offline scenario. Current representative offline RL methods [Fujimoto et al., 2019; Kumar et al., 2019; Wu et al., 2019; Kumar et al., 2020] mainly study how to reduce the error with conservative estimations, i.e., constraining the action or state-action distribution around the given dataset’s distribution when learning value functions. With such estimations, the learned RL policy can approach or exceed the original policy (also called behavioral policy) that generates the fixed dataset. As most offline RL methods assume that the behavioral policy is unknown, typical offline RL methods would first approximate the behavioral policy through generative modeling [Levine et al., 2020], i.e., learn to output actions for given states. Then, the objective is to learn approximations for selecting the highest valued actions that are similar to the approximated behavioral policy.

Different from the aforementioned methods, this paper focuses on investigating the impact of errors from generative modeling on offline RL. As we will show, in value-based off-policy scenarios, the error of generative modeling will accumulate in the process of learning the Q-value function during Bellman backup. Our main contribution involves studying the accumulating process of generative modeling error in offline RL and developing a practical method to mitigate this error. To expand, we first examine the error accumulation process during offline RL learning in detail and analyze the influence
of generative modeling error on the final offline RL error on the Q-value function. Then we propose an error reduction method via residual learning [Huang et al., 2017]. Through experiments on a set of benchmark datasets, we verify the effectiveness of our method in boosting offline RL performance over state-of-the-art methods. Furthermore, in the scenario of the multiplayer online battle arena (MOBA) game Honor of Kings, which involves large state-action space, our proposed method can also achieve excellent performance.

2 Related Work

Under fully offline settings where no additional online data collection is performed, both offline RL methods and imitation learning methods can be used to learn a policy from pre-recorded trajectories.

Offline RL. Offline RL describes the setting in which a learner has access to only a fixed dataset of experience, in contrast to online RL which allows interactions with the environment. Existing offline RL methods suffer from issues pertaining to OOD actions [Fujimoto et al., 2019; Kumar et al., 2019; Wu et al., 2019; Kumar et al., 2020]. Prior works aim to make conservative value function estimations around the given dataset’s distribution, and then only use action sampled from this constrained policy in Bellman backups for applying value penalty. Different from these works, this paper focuses on how the errors propagate throughout the whole process of offline RL, from generative modeling to value approximation.

Imitation learning. Imitation learning (IL) is to learn behavior policies from demonstration data [Schaal, 1999; Hussein et al., 2017]. Though effective, these methods are not suitable for offline RL setting because they require either on-policy data collection or oracle policy. Different from offline RL, imitation learning methods do not necessarily consider modeling the long-term values of actions or states like reinforcement learning methods.

3 Preliminaries

3.1 Reinforcement Learning

We consider the environment as a Markov Decision Process (MDP) \((S, A, P, R, \gamma)\), where \(S\) is the state space, \(A\) is the action space, \(P : S \times A \times S \to [0, 1]\) denotes the state transition probability, \(R : S \times A \to \mathbb{R}\) represents the reward function, and \(\gamma \in (0, 1]\) is the discount factor. A policy \(\pi\) is a mapping \(S \times A \to [0, 1]\). A value function provides an estimate of the expected cumulative reward that will be obtained by following some policy \(\pi(a_t|s_t)\) when starting from a state-action tuple \((s_t, a_t)\) in the case of the state-action value function \(Q^\pi(s_t, a_t)\):

\[
Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1}, a_{t+1}}[Q^\pi(s_{t+1}, a_{t+1})]
\]

In the classic off-policy setting, the learning of Q-function is based on the agent’s replay buffer \(D\) that gathers the experience of the agent in the form of \((s_t, a_t, r_t, s_{t+1})\), and each new policy \(\pi_k\) collects additional data by exploring the environment. Then \(D\), which consists of the samples from \(\pi_0, \ldots, \pi_k\), is used to train a new updated policy \(\pi_{k+1}\).

3.2 Offline Reinforcement Learning

Offline RL additionally considers the problem of learning policy \(\pi\) from a fixed dataset \(D\) consisting of single-step transitions \((s_t, a_t, r_t, s_{t+1})\), without interactions with the environment. This is in contrast to many off-policy RL algorithms that assume further interactions with the current policy rolling out in the environment. In this paper, we define the behavioral policy \(\pi_b\) as the conditional distribution \(p(a|s)\) in the dataset \(D\), which is treated unknown. In real-world cases, the training dataset could be generated by a collection of policies. For simplicity, we refer to them as a single behavioral policy \(\pi_b\).

Value Function Approximation

For large or continuous state and action spaces, the value function can be approximated with neural networks \(Q_b\), parameterized by \(\theta\). With the notion of Bellman operator \(\mathcal{T}\), we can denote Equation (1) as \(\hat{Q}^\pi = \mathcal{T}^* \hat{Q}^\pi\) with \(\pi \in [0, 1]\). This Bellman operator has a unique fixed point that corresponds to the true state-value function for \(\pi(a|s)\), which can be obtained by repeating the iteration \(Q_{k+1} = \mathcal{T}^k \hat{Q}^\pi\), and it can be shown that \(\lim_{k \to \infty} Q_{b}^k = \mathcal{T}^* \hat{Q}^\pi\).

Offline RL algorithms based on this value function approximation with the iterative update are shown to suffer from action distribution shift [Fujimoto et al., 2019] during training. Since the policy in next iterate is computed by choosing actions that greedily maximize the state-action value function at each state, \(\pi_{k+1}(a|s) = \arg \max_a Q_{b}^k(s, a)\), it may be biased towards out-of-distribution (OOD) actions with erroneously high Q-values. In RL with explorations, such errors can be corrected by rolling out the action in the environment and observing its true value. In contrast, an offline RL agent is unable to query such information from the environment. To mitigate the problem caused by OOD actions, typical offline RL methods focus on constraining the learned policy to output actions that lie in the training distribution \(\pi_b\).

Generative Modeling

Because we do not assume direct access to \(\pi_b\), it is common in previous work to approximate this behavior policy via a generative model \(G_\omega(s)\), with max-likelihood over \(D\) [Fujimoto et al., 2019; Kumar et al., 2019; Wu et al., 2019; Levine et al., 2020]:

\[
G_\omega(s) = \hat{\pi}_b := \arg \max_{\pi} E_{(s, a, r, s') \sim \mathcal{D}}[\log \hat{\pi}(a|s)]
\]

We denote the approximated policy as \(\hat{\pi}_b\) and refer to it as “cloned policy” to distinguish it from \(\pi_b\).

In Section 4, we will analyze how the errors propagate from generative modeling to value function approximation, resulting in the overall errors for offline RL. Then we will introduce how to reduce the overall errors of offline RL by reducing the generative modeling error, with theoretical proofs and implemented models in Section 5.

4 Generative Modeling Error Propagation

In this section, we define and analyze how the generative modeling error propagates during the process of value estimation in offline RL. We derive bounds which depend on the
generative modeling. This motivates further focusing on mitigating generative modeling error.

4.1 Generative Modeling Error

As discussed in the last section, we need to approximate \( \pi_D(s, a) \) with a generative model \( G_\omega(s) \). If we train \( G_\omega(s) \) with supervised learning (i.e., standard likelihood maximization) on \( D \), we have the following result from [Ross et al., 2011], whose proof can be found in [Levine et al., 2020].

**Lemma 1** (Behavioral cloning error bound). If \( \pi_{\pi_b}(s) \) is the state distribution induced by \( \pi_b \) and \( \pi(a|s) \) is trained via standard likelihood maximization on \( s \sim \tau_{\pi_b}(s) \) and optimal labels \( a \), and attains generalization error \( \epsilon_g \) on \( s \sim \tau_{\pi_b}(s) \), then \( l(\pi) \leq C + H^2\epsilon_g \) is the best possible bound on the expected error of the learned policy, where \( C \) is the true accumulated reward of \( \pi_b \).

This means that even with optimal action labels, we still get an error bound at least quadratic in the time horizon \( H \) in the offline case. Intuitively, the policy \( \pi_b \) learned with generative model \( G_\omega(s) \) may enter into states that are far outside of the training distribution, where the generalization error bound \( \epsilon_g \) no longer holds on unseen states during training. Once the policy enters one OOD state, it will keep making mistakes and remain OOD for the remainder of the testing phase, accumulating \( O(H) \) error. Since there is a non-trivial chance of entering an OOD state at every one of the \( H \) steps, the overall error scales as \( O(H^2) \).

4.2 Error Propagation on Value Estimation

**Definition 1** (Value function approximation error). We define \( \epsilon_D^v(s, a) \) as the value function approximation error between the true state-action value function \( Q^*_D \) computed from the dataset \( D \) and the true state-action value function \( Q^* \):

\[
\epsilon_D^v(s, a) = Q^*(s, a) - Q^*_D(s, a)
\]

**Theorem 1.** Given a policy \( \pi_b \) that generates the dataset \( D \), if we model its cloned policy \( \pi_b \) from \( D \) with a generative modeling error of \( \epsilon_g \), assume that \( \delta(s, a) = \sup_D \epsilon_D^v(s, a) \) and \( \eta(s, a) = \sup_D \epsilon_D^v(a|s) \), with the action space of dimension \( |A| \), \( \delta(s, a) \) satisfies the following:

\[
\delta(s, a) \geq \sum_{s'} (p^*(s'|s, a) - p_D(s'|s, a))
\]

\[
+ [r(s, a) + \gamma (\sum_{a'} (\pi_b(a'|s') + |A|\eta)Q^*_D(s', a'))] +
\]

\[
+ p^*(s'|s, a) \cdot \gamma (\sum_{a'} \pi_b(a'|s') + |A|\eta)\epsilon_D^v(s, a)
\]

**Proof.** Firstly, we have the following:

\[
\epsilon_D^v(s, a) = \sum_{s'} (p^*(s'|s, a) - p_D(s'|s, a))
\]

\[
+ [r(s, a) + \gamma (\sum_{a'} (\pi_b(a'|s') + \epsilon_g(a'|s'))Q^*_D(s', a'))] +
\]

\[
+ p^*(s'|s, a) \cdot \gamma (\sum_{a'} (\pi_b(a'|s') + \epsilon_g(a'|s'))\epsilon_D^v(s', a'))
\]

The proof follows by expanding each \( Q \), rearranging terms, simplifying the expression and then representing cloned policy \( \pi \) with behavior policy \( \pi_b \) with a generative error \( \epsilon_g \).

Based on above equation, we take the supremum of \( \epsilon_D^v(s, a) \) and have the following:

\[
\sup_D \epsilon_D^v(s, a) \geq \sum_{s'} (p^*(s'|s, a) - p_D(s'|s, a))
\]

\[
+ [r(s, a) + \gamma (\sum_{a'} (\pi_b(a'|s') + |A|\eta)Q^*_D(s', a'))] +
\]

\[
+ p^*(s'|s, a) \cdot \gamma (\sum_{a'} \pi_b(a'|s') + |A|\eta)\epsilon_D^v(s, a)
\]

For a finite, deterministic MDP, if all possible state transitions are captured in \( D \), \( p_D(s'|s, a) \) will be equivalent to \( p^*(s'|s, a) \), we will have \( \epsilon_D^v(s, a) = 0 \). However, in infinite or stochastic MDP, it might require an infinite number of samples to cover the true distribution. Therefore, \( \delta(s, a) = 0 \) if and only if the following strong assumptions holds: the true MDP is finite and deterministic, and all possible state transitions are captured in \( D \). Otherwise, we have \( \delta(s, a) > 0 \).

From Theorem 1, we have \( \delta(s, a) \) scales as \( O(|A|H^2) \), where \( \eta(s, a) = \sup_D \epsilon_D^v(a|s) \). Intuitively, this means we can prevent an increase in the state-action value function error by learning a generative model \( G_\omega(s) \) with smaller \( \epsilon_g \). Meanwhile, for settings where the action space is small, the \( \epsilon_g \) will have smaller influences in inducing the state-action value function error. Overall, the time horizon \( H \), since the generative error \( \epsilon_g \) scales as \( O(H^2) \), \( \delta(s, a) = \sup_D \epsilon_D^v(s, a) \) scales as \( O(|A|H^2) \).

5 Residual Generative Modeling

We begin by analyzing the theoretical properties of residual generative modeling in a deterministic policy setting, where we are able to measure the monotonically decreasing loss over sampled actions precisely. We then introduce our deep reinforcement learning model in detail, by drawing inspirations from the deterministic analog.

5.1 Addressing Generative Modeling Error with Residual Learning

In this section, we will provide a short theoretical illustration of how residual learning can be used in addressing the generative modeling under deterministic policies. In this example, we will assume that \( \rho_\omega(s) \) is the original generative model in mimicking the policy \( \pi_b \) in dataset \( D \), and the residual generative model is denoted as \( \tilde{a} = G_\omega(s, \rho_\omega(s)) \), where \( \omega \) stands for the parameters used to combine input \( s \) and generative model \( \rho_\omega(s) \).

Without loss of generality, we can re-write the final output layer by assuming that \( G_\omega \) is parameterized by a linear weight vector \( w \) and a weight matrix \( M \), and previous layers can be represented as \( G_{\omega_2}(s) \). Thus, \( \tilde{a} \) can be denoted as \( \tilde{a} = w^T(MG_{\omega_2}(s) + \rho_\omega(s)) \). We have the following result from [Shamir, 2018]:

\[
\epsilon_D^v(s, a) = \sum_{s'} (p^*(s'|s, a) - p_D(s'|s, a))
\]

\[
+ [r(s, a) + \gamma (\sum_{a'} (\pi_b(a'|s') + |A|\eta)Q^*_D(s', a'))] +
\]

\[
+ p^*(s'|s, a) \cdot \gamma (\sum_{a'} \pi_b(a'|s') + |A|\eta)\epsilon_D^v(s, a)
\]
Lemma 2. Suppose we have a function defined as
\[ \Gamma_\psi(a, B) = \Gamma(a, B, \psi) \doteq \mathbb{E}_{x,y} \left[ l \left( a^T (H(x) + BF_{\psi}(x), y) \right) \right] \]
where \( l \) is the defined loss, \( a, B \) are weight vector and matrix respectively, and \( \psi \) is the parameters of a neural network. Then, every local minimum of \( \Gamma \) satisfies
\[ \Gamma(a, B, \psi) \leq \inf_{a} \Gamma(a, 0, \psi) \]
if following conditions are satisfied: (1) loss \( l(\hat{y}, y) \) is twice differentiable and convex in \( \hat{y} \); (2) \( \Gamma_\psi(a, B) \), \( \nabla \Gamma(a, B) \), and \( \nabla^2 \Gamma(a, B) \) are Lipschitz continuous in \( (a, B) \).

Theorem 2. When using log loss or squared loss for deterministic policy and linear or convolution layers for \( F_{\omega_2} \), every local optimum of \( \hat{a} = w^T (MG_{\omega_2}(s) + \rho_\phi(s)) \) will be no worse than \( \hat{a} = w^T \rho_\phi(s) \).

Proof. For deterministic policies, \( G_{\omega_2}(x) \) is Lipschitz continuous when using linear or convolution layers [Virmaux and Scaman, 2018]. Since log loss and squared loss are twice differentiable in \( a \), we have
\[ \Gamma(w, M, \theta) \leq \inf_{w} \Gamma(0, 0, \theta) \]
where \( \Gamma(a, B) = \Gamma(a, B, \psi) \doteq \mathbb{E}_{x,y}[l(a^T (H(x) + BF_{\psi}(x), y)) \right] \), \( l \) is the defined loss, \( a, B \) are weight vector and matrix respectively, and \( \theta \) is the parameters of a neural network.

That is, the action-conditioned model has no spurious local minimum that is above that of the original generative model \( \rho_\phi(s) \).

5.2 Residual Generative Model

The main difference between our method and existing offline RL methods is that we design a residual modeling part for the generative model when approximating the \( \pi_D \). Therefore, in this section, we mainly introduce our approach to offline reinforcement learning, AQL (action-conditioned Q-learning), which uses action-conditioned residual modeling to reduce the generative modeling error.

Our generative model consists of two major components: a conditional variational auto-encoder (VAE) that models the distribution by transforming an underlying latent space, and a residual neural network that models the state-action distribution residuals on the output of the VAE.

Conditional VAE

To model the generative process of predicting actions given certain states, analogous to existing literatures like Batch-Constrained Q-learning (BCQ) [Fujimoto et al., 2019] and Bootstrapping error reduction (BEAR) [Kumar et al., 2019], we use a conditional VAE that takes state and action as input and outputs the reconstructed action. Given the raw state, we first embed the state observation with a state embedding module in conditional VAE. Then in the encoder part, we concatenate the state embedding with action input and output the distribution of latent space (assumed to be Gaussian for continuous action space and Categorical for discrete action space). In the decoder part, we concatenate the state embedding and the latent variable \( z \) from the learned distribution and outputs the reconstructed action. The overall training loss for the conditional VAE is:
\[ L_{VAE} = -\mathbb{E}_{z \sim q(z,s,a)} [\log p(a|s, z)] + D_{KL}(q(z|s, a)||p(z)) \]
where the first term is the reconstructed loss, and the second term is the regularizer that constrains the latent space distribution, \( s \) is the state input to the conditional VAE, \( a \) is the action in the dataset in pair with \( s \).

Residual Network

Unlike BCQ and BEAR which take VAE or a single feedforward neural network as the generative model, we propose to use the reconstructed action from VAE as an additional input to learn the residual of action output. This residual mechanism is motivated to boost offline RL by reducing the generative model’s error. The overall loss for the generative network is:
\[ L_o = \mathbb{E}_{o}[d(\hat{a}, a)] + L_{VAE} \]
where \( \hat{a} = G_o(s) \) is the output of the residual network, and \( d(\hat{a}, a) \) is the distance measure between two actions. For continuous actions, \( d \) could be defined as the mean squared error; for discrete actions, \( d \) could be defined as the cross-entropy. Intuitively, this loss function includes the original generative modeling loss function (usually treated the same as a behavior cloning loss) and a VAE loss, optimized at the same time.

5.3 Training Process

We now describe the practical offline RL method based on BEAR, a similar variant on BCQ or other methods can be similarly derived. Empirically we find that our method based on BEAR performs better. We’ve described our generative model in previous sections, here we briefly introduce the other part of the offline RL algorithm, i.e., value function approximation process, which is similar to BCQ and BEAR. To compute the target Q-value for policy improvement, we use the weighted combination of the maximum and the minimum Q-value of \( K \) state-action value functions, which is also adopted in BCQ and BEAR and shown to be useful to penalize uncertainty over future states in existing literature [Fujimoto et al., 2019]:
\[ q(s, a) = r + \gamma \max_{a_i} \lambda \min_{j=1,...,K} Q_j(s', a_i) + (1 - \lambda) \max_{j=1,...,K} Q_j(s', a_i) \]
where \( s' \) is the next state of current state \( s \) after taking action \( a \), \( \theta'_j \) is the parameters for target network, \( \gamma \) is the discount factor in Bellman Equation, \( \lambda \) is the weighting factor.

Following BEAR, we define the generative modeling update process as a constrained optimization problem, which tries to improve the generator and constrains the policy within a threshold \( \epsilon \):
\[ G_\omega = \max_{G_\omega} \mathbb{E}_{(s,a,r,s') \sim D}[\mathbb{E}_{\hat{a} \sim G_\omega(s)} \min_{i=1,...,K} Q_i(s, \hat{a}) - \alpha(L_{G_\omega(s)} - \epsilon)] \]
where \( \alpha \) is the tradeoff factor that balances constraining the policy to the dataset and optimizing the policy value, and can
be automatically tuned via Lagrangian dual gradient descent for continuous control and is fixed as a constant for discrete control.

This forms our proposed offline RL method, which consists of three main parameterized components: a generative model \( G_w(s) \), Q-ensemble \( \{ Q_i \}_{i=1}^{K} \), target Q-networks \( \{ Q_i^\tau \}_{i=1}^{K} \) and target generative model \( G_{w'}. \) In the following section, we demonstrate AQL results in mitigating the generative modeling error and a strong performance in the offline RL setting.

6 Experiment

We compare AQL to behavior cloning, off-policy methods and prior offline RL methods on a range of dataset compositions generated by (1) completely random behavior policy, (2) partially trained, medium scoring policy, and (3) an optimal policy. Following [Fujimoto et al., 2019; Kumar et al., 2019; Wu et al., 2019], we evaluate performance on four Mujoco [Todorov et al., 2012] continuous control environments in OpenAI Gym [Brockman et al., 2016]: HalfCheetah-v2, Hopper-v2, and Walker2d-v2. We evaluate offline RL algorithms by training on these fixed datasets provided by open-access benchmarking dataset D4RL [Fu et al., 2020] and evaluating the learned policies on the real environments. The statistics for these datasets can be found in [Fu et al., 2020].

Baseline. We compare with off-policy RL methods - Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015] and Soft Actor-Critic (SAC) [Haarnoja et al., 2018], Behavior Cloning (BC) [Ross and Bagnell, 2010], and state-of-the-art off-policy RL methods, including BCQ [Fujimoto et al., 2019], BEAR [Kumar et al., 2019], Behavior Regularized Actor Critic with policy (BRAC-p) or value (BRAC-v) regularization [Wu et al., 2019], and Conservative Q-Learning (CQL(H)) [Kumar et al., 2020]. We use the open-source codes provided in corresponding methods and keep the same parameters in our proposed method for a fair comparison.

Experimental settings. To keep the same parameter settings as BCQ and BEAR, we set \( K = 2 \) (number of candidate Q-functions), \( \lambda = 0.75 \) (minimum weighting factor), \( \epsilon = 0.05 \) (policy constraint threshold), and \( H = 1000 \) (total training steps). We report the average evaluation return over three seeds of the learned algorithm’s policy, in the form of a learning curve as a function of the number of gradient steps taken by the algorithm. The samples collected during the evaluation process are only used for testing and not used for training.

6.1 Results

Effectiveness of mitigating generative modeling error. In previous sections, we argued in favor of using the residual generative modeling to decrease the generative modeling error. Revisiting the argument, in this section, we investigate the empirical results on the error between true \( \pi \) at state \( s \) from \( \mathcal{D} \) and the generated action \( \hat{a} \) from \( G_w(s) \). In BCQ, it uses a vanilla conditional VAE as \( G_w(s) \); BEAR use a simple feed-forward neural network. Our proposed method combines these two models with a residual network.

Figure 1 shows the comparison of the error, with the mean and standard deviation from the last 500 training batches of each method. We have the following observations:

- Both AQL and BEAR have a lower error in generative modeling than BCQ, and both AQL and BEAR keep the errors below 0.05 in most cases. This is because they use the dual gradient descent to keep the target policy constrained below the threshold \( \epsilon \) that is set as 0.05, while BCQ does not have this constraint.

- Our proposed method AQL has a consistent lower error than BEAR. This is because AQL uses residual learning to mitigate the generative modeling error, which matches our analysis, as suggested by Theorem 2.

Effectiveness of boosting offline RL. As analyzed in previous sections, we can prevent an increase in the state-action value function error by learning a generative model \( G_w(s) \) with smaller \( \epsilon_g \), and thus learn a better offline RL policy. Here, we investigate the effectiveness of the proposed method on boosting the performance of offline RL.

Table 1 shows the comparison of our proposed method over behavior cloning, off-policy methods, and state-of-the-art offline RL methods. We have the following observations:

- Off-policy methods (DDPG and SAC) training in purely offline setting yields a bad performance in all cases. This is due to the incorrect estimation of the value of OOD actions, which matches the existing literature.

- Offline RL methods can outperform BC under datasets generated by the random and medium policy. This is because BC simply mimicking the policy behavior from the dataset without the guidance of state-action value. Since the dataset \( \mathcal{D} \) is generated by a non-optimal policy, the policy learned by BC could generate non-optimal actions. This non-optimality could accumulate as the policy rolls out in the environments.

- AQL performs similarly or better than the best prior methods in most scenarios. We noticed that BRAC and CQL(H) yields better performance in random and medium data than in expert data. This is because under expert data, the variance of the state-action distribution in the dataset might be small, and simply mimicking the behavior policy could yield satisfactory performance (like BC). While BRAC and CQL(H) does not have any specific design to reduce the generative error, AQL has a better generative model to mimicking the distribution of \( \mathcal{D} \) and thus consistently performs as one of the best methods across most settings. As analyzed in previous sections, we can lower the error of state-action value estimation, thus boost the offline RL performance.

We also noticed that although the overall performance of AQL is slightly better than CQL(H), since CQL(H) has additional constraints that AQL does not consider. We added the constraints of CQL(H) into AQL and found out that AQL can further improve on CQL(H) in all cases, which means the effectiveness of boosting offline RL methods by reducing the generative error.

6.2 Case Study: Honor of Kings 1v1

An open research problem for existing offline RL methods is the lack of evaluations on complex control tasks [Levine et al., 2020; Wu et al., 2019] with large action spaces. Therefore, in this paper, we further implement our experiment into
Compared with BCQ version of our method with DQN, for learning the distribution of $a \sim D(s)$ and $\hat{a} \sim G_{\omega}(s)$ from generative model. Note that the y-axis of HALF CHEETAH is in log scale. The lower, the better. All the results are reported over last 500 training batches. With residual network, our proposed method can largely reduce the error.

<table>
<thead>
<tr>
<th>Method</th>
<th>BC</th>
<th>SAC</th>
<th>DDPG</th>
<th>BCQ</th>
<th>BEAR</th>
<th>BRAC-p</th>
<th>BRAC-v</th>
<th>CQL($H$)</th>
<th>AQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOPPER-random</td>
<td>-17.9</td>
<td>3502.0</td>
<td>209.22</td>
<td>-1.3</td>
<td>2831.4</td>
<td>2713.6</td>
<td>3590.1</td>
<td>3902.4</td>
<td>3053.1</td>
</tr>
<tr>
<td>HOPPER-random</td>
<td>299.4</td>
<td>347.7</td>
<td>62.13</td>
<td>323.9</td>
<td>349.9</td>
<td>337.5</td>
<td>376.3</td>
<td>340.7</td>
<td>379.2</td>
</tr>
<tr>
<td>WALKER2d2d-random</td>
<td>73.0</td>
<td>192.0</td>
<td>39.1</td>
<td>228.0</td>
<td>336.3</td>
<td>-7.2</td>
<td>87.4</td>
<td>346.6</td>
<td>350.3</td>
</tr>
<tr>
<td>HALF CHEETAH-medium</td>
<td>4196.4</td>
<td>-808.6</td>
<td>-754.85</td>
<td>4342.67</td>
<td>4159.08</td>
<td>5158.8</td>
<td>5473.8</td>
<td>5236.8</td>
<td>4397.2</td>
</tr>
<tr>
<td>WALKER2d2d-medium</td>
<td>304.8</td>
<td>44.2</td>
<td>4.63</td>
<td>244.1</td>
<td>2717.0</td>
<td>3559.9</td>
<td>3725.8</td>
<td>3487.1</td>
<td>1763.5</td>
</tr>
<tr>
<td>HOPPER-medium</td>
<td>923.5</td>
<td>5.7</td>
<td>10.19</td>
<td>1752.4</td>
<td>1674.5</td>
<td>1044.0</td>
<td>990.4</td>
<td>1694.0</td>
<td>1768.1</td>
</tr>
<tr>
<td>HALF CHEETAH-expert</td>
<td>12984.5</td>
<td>-230.6</td>
<td>-649.1</td>
<td>10539.1</td>
<td>13130.1</td>
<td>461.13</td>
<td>-133.5</td>
<td>12189.9</td>
<td>12303.4</td>
</tr>
<tr>
<td>HOPPER-expert</td>
<td>3525.4</td>
<td>22.6</td>
<td>48.2</td>
<td>3410.5</td>
<td>3567.4</td>
<td>213.5</td>
<td>119.7</td>
<td>3522.6</td>
<td>3976.4</td>
</tr>
<tr>
<td>WALKER2d2d-expert</td>
<td>3143.9</td>
<td>-13.8</td>
<td>9.8</td>
<td>3259.2</td>
<td>3025.7</td>
<td>-9.2</td>
<td>0.0</td>
<td>143.3</td>
<td>3253.5</td>
</tr>
</tbody>
</table>

Table 1: Performance of AQL and prior methods on gym domains from D4RL, on the unnormalized return metric, averaged over three random seeds, with top-3 emphasized. While BCQ, BEAR, BRAC and CQL(\$H\$) perform unstably across different scenarios, AQL consistently performs similarly or better than the best prior methods.

a multiplayer online battle arena (MOBA) game, the 1v1 version of Honor of Kings (the most popular and widely studied MOBA game at present [Ye et al., 2020a; Ye et al., 2020c; Chen et al., 2020]). Compared with traditional games like Go or Atari, Honor of Kings 1v1 version has larger state and action space and more complex control strategies. A detailed description of the game can be found in [Ye et al., 2020c].

**Baselines.** Since the action space is discrete, we compare our method with DQN [Mnih et al., 2013] and the discrete version of BCQ [Fujimoto et al., 2019] instead of BEAR. Compared with DQN, BCQ only adds one generative model to learn the distribution of $D$. For AQL, we add a residual network upon BCQ as the discrete version of our proposed method. We use the same network parameters and training parameters for all the baseline methods for a fair comparison.

**Results.** In our experiment, we evaluate the ability of the agents learned by DQN, BCQ and our method, against the internal behavior-tree-based AI in the game, as is shown in Figure 2. We have the following observations:

- Offline RL methods (BCQ and our proposed method) largely reduce the convergence time of classical off-policy method DQN. This means the generative modeling of the dataset $D$ helps the learning of value function.
- Our proposed method further outperforms BCQ. Compared with BCQ, the discrete version of our method uses the residual network in the generative modeling process, which mitigates the error of modeling the distribution of $D$, and boosts the performance of learned value function.

### 7 Conclusion

The goal of this work is to study the impact of generative modeling error in offline reinforcement learning (RL). We theoretically and empirically analyze how error propagates from generative modeling to the value function approximation in Bellman backup of off-policy RL. We propose AQL (action-conditioned Q-learning), a residual generative model to reduce policy approximation error. Our experiments suggest that AQL can help to boost the performance of offline RL methods. Our case study on complex tasks further verifies that offline RL methods can efficiently learn with faster convergence when integrated in the online process with offline learning.
References


