You Only Evaluate Once – a Simple Baseline Algorithm for Offline RL

Wonjoon Goo
Department of Computer Science
University of Texas at Austin
wonjoon@cs.utexas.edu

Scott Niekum
Department of Computer Science
University of Texas at Austin
sniekum@cs.utexas.edu

Abstract

The goal of offline reinforcement learning (RL) is to find an optimal policy given prerecorded trajectories. This setup is appealing since it separates the learning process from the possibly expensive or unsafe data-gathering process. Concerning this problem, much research has been proposed that customizes existing off-policy RL algorithms, especially actor-critic algorithms in which policy evaluation and improvement are iterated. However, the convergence of such approaches is not guaranteed due to the use of complex non-linear function approximation and an intertwined optimization process. In this paper, we propose a simple baseline algorithm for offline RL that only performs the policy evaluation step once so that the algorithm does not require complex stabilization schemes. The algorithm is not likely to converge to an optimal policy, but we empirically find that when we use proper regularization, a greedy policy implied by the learned value function exhibits competitive performance in a subset of the D4RL offline RL benchmark, and it even achieves state-of-the-art results in the hopper-medium-replay, walker2d-medium-replay, and hopper-medium datasets.

1 Introduction

Reinforcement learning involves an active component during learning; an agent continuously gathers experience as it learns a policy. This is a very general learning framework that resembles the way animals learn, but the interactive component often hurts the applicability of RL since the agent interaction can be expensive or unsafe. To address this challenge, the offline reinforcement learning paradigm has been proposed and investigated, which tries to learn a policy purely from pre-generated data [15]. Considering that many recent breakthroughs in machine learning can be attributed to large-scale data, this new paradigm is very promising. However, the offline setup causes significant theoretic and algorithmic difficulties that need to be resolved to fulfill this promise.

Specifically, value-based off-policy RL algorithms suffer from the overestimation problem caused by function approximation error and bootstrapping [7, 6] in the offline RL setup, even though the algorithms are equipped with algorithmic techniques [16, 7, 9] that can stabilize learning and mitigate the so-called *Deadly Triad* [23]. This is because over-estimated values cannot be readjusted in offline RL, unlike the ordinary RL setup where incorrectly optimistic actions get executed and corrected.

Since the overestimation problem is prominent when the value function is queried for out-of-distribution inputs, many algorithms have been proposed that ensure the bootstrapping queries are within the data manifold of the given offline dataset [8, 10, 29, 21, 19, 27]. However, we argue that these algorithms are *unsafe* in the sense that the learning could diverge if the function approximator over-estimates for out-of-distribution queries. Empirically, these algorithms show reasonable stability in some domains, but, with the limited understanding of the generalization characteristics of the function approximator in use, these methods are still vulnerable because out-of-distribution queries would be made eventually in optimizing the value function and the (underlying) policy.

The Deadly Triad cannot be avoided in these actor-critic algorithms since all of the three components – off-policy learning, function approximation, and bootstrapping – are in use. However, the first critic update step is the exception since the critic is trained with a fixed behavior policy. While the first policy implied by the first value function is likely to be suboptimal, it is valuable to compute this policy as a baseline to use as a sanity-check. By comparing performance of this suboptimal policy with iterative algorithms, we can measure if there is a sufficient advantage that results from risky iterative optimization.

To this end, we propose an offline RL algorithm that evaluates the value of a behavior policy once and extracts a greedy policy under the learned value approximation. This provides an simple baseline that actor-critic methods should easily outperform, but surprisingly, we find that when our action-value function is trained with pessimistic regularization [11, 3], the learned policy achieves competitive results in several D4RL benchmarks. Our results indicate not only the effectiveness of the proposed algorithm adopting pessimistic regularization, but also implies potential vulnerabilities of the iterative optimization process of actor-critic algorithms in the offline setting.

2 Related Work

Evaluating and improving a policy with the data generated from a different policy (off-policy RL) has been widely investigated, and several papers have shown theoretical convergence properties of prediction and control algorithms in the off-policy setting [18, 4, 12]. However, the theoretical frameworks are limited to a linear function approximation while a non-linear function approximation is essential to handle large-scale MDPs. Yet, there have been efforts to build a practical algorithms with non-linear function approximators, and they have shown considerable success in various domains [24, 22, 17, 14] tackling real-world RL problems.

In theory, off-policy algorithms can be used in the offline setup without any modifications, but the algorithms often catastrophically fail when applied in the batch setting [8]. This is due to the accumulation of extrapolation error from bootstrapping and the policy improvement step (i.e. $\max$ operation) [8, 10]. In the non-batch setting, new experiences gathered via interaction can prevent this degenerate case, but it is impossible in the batch setting where interaction is prohibited. Pertaining to this problem, much research has been proposed, especially in the context of actor-critic algorithms in which policy evaluation and improvement are iterated. One class of solutions constrains the policy improvement step so that the optimized policy matches the behavior policy, in distribution [29] or in support [10]. In more recent work, a behavior policy for distribution matching is replaced with a prior policy, which is trained along with policy evaluation via weighted behavior cloning [21]. In [27], the policy improvement step is omitted while using the prior policy as a target policy for the evaluation.

The most closely related prior works are behavior cloning-based methods [20, 26, 19] which mimic a subset of good state-action pairs from the pre-generated trajectories. Since these methods approximate a value function for a behavior policy without an iterative policy improvement step, the algorithm does not diverge similar to ours. The main difference is that we train an action-value function and use the trained function at policy execution time in a non-parametric way while the prior works only evaluate a state-value function and filter state-action pairs based on advantage calculated with Monte-Carlo return [26] or TD(k) return [19].

For principled regularization of the action-value function, we adapt pessimism under uncertainty which allows a legitimate action-value function even when the dataset is not informative of the value of every policy [3]. The pessimism can be applied by directly penalizing Q values for some policies other than a behavior policy [11], or with the model-approximation [30] that leverages uncertainty prediction techniques developed for supervised learning, such as [13, 25]. While the previous methods penalize an action-value function, we propose a novel regularization method that penalizes an action value distribution.

3 Preliminaries

The common mathematical framework for reinforcement learning is a Markov Decision Process (MDP), which is defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$ defined by a set of states $s \in \mathcal{S}$, a set of actions $a \in \mathcal{A}$, a conditional transition dynamics $T(s'|s,a)$, an initial state distribution d_0, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in (0, 1]$. In this framework, the goal of reinforcement learning is to find an optimal policy $\pi(a|s)$ that maximizes an expected sum of discounted reward (return). Formally, the objective is defined as:

$$J(\pi) = \mathbb{E}_{\tau \sim p^\pi}\big[\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\big], \tag{1}$$

where τ is a sequence of states and actions $(s_0, a_0, \ldots, s_H, a_H)$ of length H, and p^π is a trajectory distribution of a policy, which can be represented as:

$$p^\pi(\tau) = d_0(s_0) \prod_{t=0}^{H} \pi(a_t|s_t) T(s_{t+1}|s_t, a_t). \tag{2}$$

One way to find an optimal policy is to estimate an action-value function Q^β, which represents the expected return over possible trajectories following a policy β starting from a given state and action: $Q^\beta(s_t, a_t) = \mathbb{E}_{\tau \sim p^\beta}\big[\sum_{t'=t}^{H} \gamma^{t'-t} r(s_{t'}, a_{t'})\big]$. Q^β function implies a policy $\hat{\beta}(a|s) = \delta\big(a = \arg\max_a Q(s,a)\big)$, which is better than or equal to its original evaluation target policy β. Therefore, when we perform policy evaluation (Q estimation) and policy updates iteratively, we can find the optimal policy π^* and the optimal Q function Q^{π^*}. Policy evaluation can be done with a Monte-Carlo method, but bootstrapping is commonly used, which utilizes a recursive equation that must be satisfied at convergence:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim T(s'|s,a), a' \sim \pi(a'|s')} Q^\pi(s', a'). \tag{3}$$

When an MDP is discrete and Q can be represented by a tabular representation (i.e. when $|\mathcal{S}| \times |\mathcal{A}|$ is small), it is known that policy evaluation converges to a correct solution in the limit of the number of transition tuples [23]. However, when an MDP has a large state or action space, Q has to be represented with a function approximator, such as a deep neural network. In addition, when the action space is continuous, directly extracting a better policy from Q becomes infeasible due to the $\max$ operator. These restrictions are addressed in actor-critic algorithms [16, 7, 9] which explicitly alternate policy evaluation and policy improvement with a batch of (online) transition samples D and a parameterized value function $\hat{Q}_\theta$ and a policy π_ϕ:

$$\theta_{k+1} \leftarrow \arg\min_\theta \mathbb{E}_{s,a,s' \sim D}\left[d\big(\hat{Q}_\theta(s,a), r(s,a) + \gamma \mathbb{E}_{a' \sim \pi_{\phi_k}(a'|s')} \hat{Q}_\theta(s', a')\big)\right] \text{ (policy evaluation)}, \tag{4}$$

$$\phi_{k+1} \leftarrow \arg\max_\phi \mathbb{E}_{s \sim D, a \sim \pi_\phi(a|s)}[\hat{Q}_{\theta_{k+1}}(s,a)] \text{ (policy improvement)}, \tag{5}$$

where k is an update step, and d is a distance measure such as squared l_2 distance.

4 You Only Evaluate Once

Behind actor-critic based offline RL algorithms, there is a common presumption that the iterative process is essential in achieving better performance than that of behavior policy, even though it could sacrifice the reliability of the algorithm due to the destructive over-estimation problem. This is because we want to build a policy that behaves differently to the data-generating policy by making counterfactual queries (policy improvement) and answering (policy evaluation) them iteratively [15]. However, it has not been established that a simpler and safer baseline cannot work well—a policy that selects the best action with regard to the action-value function of the *behavior policy*. Without rigorously examining this hypothesis, the true worth of iterative algorithm and counterfactual queries cannot be fully comprehended. We try to answer this question by proposing a stable offline algorithm that only behaves optimally with regard to the action-value function Q^β.

The accuracy of the trained $\hat{Q}^\beta$ is essential to fully maximize the potential of the baseline. To achieve this, we propose a method that combines two recently proposed value-learning methods: distributional reinforcement learning [1, 8, 28] and pessimism in the face of uncertainty [3, 11]. Specifically, we propose an algorithm that trains the action-value distribution of the behavior policy in a supervised way; first, a state-value distribution $\hat{Y}^\beta_\psi$ and action-value distribution $\hat{Z}^\beta_\psi$ is trained without regularization, and then we train a regularized $\hat{Z}^\beta_\theta$ by generating a fixed target value distribution with $\hat{Y}^\beta_\psi$ and $\hat{Z}^\beta_\psi$. Then, we convert $\hat{Z}^\beta_\theta$ back to $\hat{Q}^\beta$ by taking an expectation of $\hat{Z}^\beta_\theta$. We also use the distortion measure in taking expectation to embrace the notion of risk-sensitivity [28].

We hypothesize that distributional representation is beneficial in selecting the best action because the representation can encode the inherently diverse policies of the given data-distribution by not aggregating the diverse statistics into a single statistics; while Q^β treats the underlying policies of a dataset as a single mixture policy, Z^β considers the stochasticity of the behavior policy by modeling a distribution instead of a single scalar. We use implicit quantile network (IQN) [28] to parameterize a value distribution. In IQN, a distribution is represented in the form of the inverse cumulative distribution function:

$$\hat{Y}_\psi(s;\tau) = \psi_{f_y}\big(\psi_{E_y}(s) \odot \psi_{T_y}(\tau)\big), \tag{6}$$

$$\hat{Z}_\psi(s,a;\tau) = \psi_{f_z}\big(\psi_{E_z}(s,a) \odot \psi_{T_z}(\tau)\big), \tag{7}$$

$$\hat{Z}_\theta(s,a;\tau) = \theta_f\big(\theta_E(s,a) \odot \theta_T(\tau)\big), \tag{8}$$

where $\odot$ is element-wise vector product, τ is a probability value $\tau \in [0,1]$, and each of $\psi_{f,E,T}$ and $\theta_{f,E,T}$ represents a parameterized function that maps corresponding input to $1, 64, 64$-dimensional output. For $\theta_{f,E}$, we use fully-connected layers, and for θ_T, we use cosine-based embedding following [28]:

$$\theta_T(\tau)_j = \text{ReLU}\big(\sum_{i=0}^{63} \cos(\pi i \tau) w_{ij} + b_j\big), \tag{9}$$

where w_{ij} and b_j are parameters for θ_T. Now, the state-value distribution $\hat{Y}_\psi$ and unregularized action-value distribution $\hat{Z}_\psi$ can be trained by minimizing the 1-Wasserstein metric between the bootstrapped target distribution and $\hat{Y}_\psi$ or $\hat{Z}_\psi$ for all transition tuples in D iteratively:

$$\psi^{k+1} \leftarrow \arg\min_\psi \mathbb{E}_{s,a,r,s',a' \sim D}\Bigg[\int_0^1 |r + \gamma \hat{Y}_{\psi^k}(s';\omega) - \hat{Y}_\psi(s;\omega)| d\omega + \int_0^1 |r + \gamma \hat{Z}_{\psi^k}(s',a';\omega) - \hat{Z}_\psi(s,a;\omega)| d\omega\Bigg]. \tag{10}$$

Instead of directly optimizing the objective, we minimize the IQN loss [28] iteratively with a stochastic gradient descent algorithm:

$$\delta^{\tau,\tau'} = \Big(r + \gamma \hat{Y}_\psi(s';\tau') - \hat{Y}_\psi(s;\tau)\Big) + \Big(r + \gamma \hat{Z}_\psi(s',a';\tau') - \hat{Z}_\psi(s,a;\tau)\Big) \tag{11}$$

$$\mathcal{L} = \frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho^\kappa_{\tau_i}\big(\delta^{\tau_i,\tau'_j}\big) \tag{12}$$

where N and N' denote the number of iid samples $\tau_i, \tau'_j \sim U([0,1])$ used to estimate the loss, and $\rho^\kappa_\tau(\delta)$ is the Huber quantile regression loss, $\rho^\kappa_\tau(\delta) = |\tau - \mathbb{I}\{\delta < 0\}| \frac{\mathcal{L}_\kappa(\delta)}{\kappa}$ with the Huber loss $\mathcal{L}_\kappa$ having threshold κ [8, 28].

With the approximated state-value distribution $\hat{Y}_\psi$ and action-value distribution $\hat{Z}_\psi$ of the behavior policy, we train an action-value distribution $\hat{Z}_\theta$ with a proper regularization term $\mathcal{R}$ in a supervised way:

$$\theta = \arg\min_\theta \mathbb{E}_{s,a,r,s',a' \sim D}\Bigg[\int_0^1 |r + \gamma \hat{Z}_\psi(s',a';\omega) - \hat{Z}_\theta(s,a;\omega)|\Bigg] + \mathcal{R}(\psi,\theta). \tag{13}$$

We use the same loss formulation shown in Eq. 12 to approximate the expectation. For the regularization term $\mathcal{R}$, we implement the principle of pessimism in the face of uncertainty [3, 11] leveraging the pretrained, unregularized value distributions.

First, we propose a model-free regularization method that utilizes $\hat{Z}_\psi$ to bound the pessimism without a sensitive hyperparameter. Since $\hat{Z}_\psi$ represents the possible values we get when the behavior policy is executed, we can pessimistically expect that the unseen action $\tilde{a}$ would consist of the lower part of the distribution. This idea can be formally implemented by distorting the distribution with a risk-averse distortion measure $\xi : [0, 1] \to [0, 1]$ which maps τ to a smaller value: $\hat{Z}'_\psi(s, a; \tau) = \hat{Z}_\psi(s, a; \xi(\tau))$ [28]. We call this type of pessimism *data-driven* pessimism since the level of pessimism is determined by the inherent stochasticity measured by trained $\hat{Z}_\psi$. Also, the strength of pessimism is applied differently for each given state and action, and we hypothesize this allows a more sensible target action-value distribution that is not overly regularizing. In contrast, CQL implements the pessimism via unbounded objective, so the level of pessimism has to be controlled delicately by an extra hyperparameter. For the implementation, we use a conditional value-at-risk (CVaR) distortion measure CVaR$(\eta, \tau) = \eta\tau$, to generate a bounded pessimistic target distribution for $\hat{Z}_\theta(s, \tilde{a})$:

$$\mathcal{R} = \mathbb{E}_{s\sim D, \tilde{a}\sim D}\left[\int_0^1 |\hat{Z}_\psi(s, \tilde{a}; \eta\omega) - \hat{Z}_\theta(s, \tilde{a}; \omega)| d\omega\right]. \tag{14}$$

On top of that, we introduce an additional regularization method that handles the degenerate case of bounded data-driven pessimism. While data-driven pessimism can provide a sensible pessimistic target distribution $\hat{Z}_\psi(s, \tilde{a})$, when a dataset has a lack of action diversity for a certain state s, the target pessimistic distribution can be not pessimistic enough since the trained $\hat{Z}_\psi(s, \cdot)$ behaves more like a state value function $V(s)$ ignoring action input. It could make the proposed regularization ineffective. Fortunately, the lack of diversity can be easily captured by observing the entropy or the variance of the distribution $\hat{Y}(s)$, so this problem can be addressed straightforwardly. In our implementation, instead of estimating a variance, we use the gap between the maximum and the minimum value of the distribution, which is more efficient to compute. With the hyperparameter α that states the minimum gap requirements between the pessimistic distribution and the ordinary distribution, we define k as following and use it to ensure the minimum level of pessimism:

$$k = \max\big(\alpha - \hat{Y}_\psi(s; 1-\epsilon) + \hat{Y}_\psi(s; \epsilon), 0\big) \tag{15}$$

where ϵ is a small positive number as 1e-2. We call this feature *gap requirement*. The final regularization term is formally defined as follow:

$$\mathcal{R} = \mathbb{E}_{s\sim D, \tilde{a}\sim D}\left[\int_0^1 |\hat{Z}_\psi(s, \tilde{a}; \eta\omega) - k - \hat{Z}_\theta(s, \tilde{a}; \omega)| d\omega\right]. \tag{16}$$

With the regularized action-value distribution $\hat{Z}_\theta$ and inferred $\hat{Q}$ from it, we build a greedy non-parametric policy which selects the best action among actions exist in a dataset D:

$$\pi(b|s) = \delta\big(b = \arg\max_{a\sim D} \hat{Q}(s, a)\big).. \tag{17}$$

The motivation behind using non-parametric policy instead of a parameterized policy is that we can (1) avoid the unwanted generalization error derived from policy parameterization, (2) reduce extra hyperparameters related to parameterization and training and (3) expect more accurate value estimation since $\hat{Z}_\theta$ is directly regularized with the same action distribution which is used for a policy.

5 Experiments

In our experiments, we aim to study the following question: how much performance gain will the policy iteration provide? To answer this question, we compare the performance of YOEO, which evaluates the policy only once, against several baselines and prior works based on policy iterations; behavior cloning (BC), soft actor-critic (SAC) [9] without interaction, two policy constrained methods, namely bootstrapping error accumulation reduction (BEAR) [10] and behavior regularized actor-critic (BRAC) [29], and conservative Q-learning (CQL) [11]. We use 2 fully-connected layers for each

Table 1: Performance of YOEO and prior methods on a subset of D4RL benchmarks. Each number represents the performance relative to a random policy as 0 and an expert policy as 100. All the numbers except ours are borrowed from [5], [11]. The numbers are averaged over 3 different random seeds.

Type	**Env.**	**BC**	**SAC-offline**	**BEAR**	**BRAC**	**AWR**	**BCQ**	**CQL ($\mathcal{H}$)**	**YOEO (Ens. 5)**	**YOEO (Ens. 9)**
Medium-Replay	Hopper	11.8	3.5	33.7	0.6	28.4	33.1	48.6	83.4	**94.8**
	Walker2D	11.3	1.9	19.2	0.9	15.5	15	26.7	**39.3**	**41.5**
Medium	Hopper	29	0.8	52.1	31.1	35.9	54.5	58	**63.4**	**76.4**
	Walker2D	6.6	0.9	59.1	**81.1**	17.4	53.1	**79.2**	70.4	68.6
Medium-Expert	Hopper	**111.9**	1.6	96.3	0.8	27.1	**110.9**	**111**	94	**106.4**
	Walker2D	6.4	-0.1	40.1	81.6	53.8	57.5	**98.7**	88.9	92.7

trainable component except the cosine embedding function, and we ensemble five $\hat{Z}_\theta$s to construct the nonparametric policy. We calculate $\hat{Q}$ by taking a mean over 20 random samples generated by a distorted $\hat{Z}'_\theta$, which is distorted with CVaR($\eta = 0.2$). In addition, we reduce the search space for the nonparametric policy by first finding the 100 nearest states in raw-state space and querying the actions of those states. We use an approximated nearest neighbor algorithm called annoy [2] for the implementation. The detailed hyperparameters used for the experiments are shown in Appendix, and code is also available[1].

The comparison is made on a subset of datasets in the D4RL offline RL benchmark [5]. We use Mujoco locomotion and Adroit hand-manipulation tasks. For Mujoco locomotion tasks, we use three environments (`hopper`, `walker2d`, and `halfcheetah`) with three different settings (`medium-replay`, `medium`, and `medium-expert`); `medium-replay` consists of replay buffer gathered while running SAC until a policy reaches the environment-specific performance threshold; `medium` consists of 1 million transition tuples generated by executing a medium policy; `medium-expert` consists of 2 million transition tuples each half of which generated with medium and expert policy. For Adroit hand-manipulation tasks, we use four environments (`pen`, `door`, `relocate`, and `hammer`) in two settings (`human` and `cloned`). We train YOEO for 1 million stochastic gradient steps, then report the average normalized performance score over 100 trajectories. The results are displayed in Table 1 and Table A.2.

YOEO achieves state-of-the-art performance in hopper-medium-replay, walker2d-medium-replay, and hopper-medium datasets, and the proposed method consistently shows competitive performance across all configurations, surpassing most of the previous offline RL algorithms without iterative policy evaluation. The results indicate that the previous offline RL algorithms fail to fully exploit the potential benefit of iterative evaluation and update.

5.1 Ablation Experiments & Analysis

We conduct ablation studies to disentangle the contribution of each component in YOEO. There are three major components: (1) action-restricted optimization, (2) ensemble, and (3) distributional value estimation and adaptive pessimism.

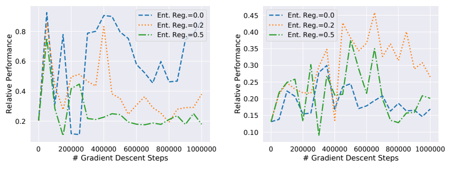
5.1.1 Action-restricted Optimization

We build a non-parametric greedy policy based on the pessimistically regularized action-value distribution $\hat{Z}^\beta_\theta$; the best action is selected among the actions exist in the dataset. There are two other common ways generating a policy given an action-value function (or distribution): (1) filtering-based behavior cloning [21, 27] which selects a good state-action pairs based on an advantage, and (2) actor-critic methods [10, 11] which optimizes policy parameters with regard to action-value function. We test these two baseline algorithms with the same regularized action-value distributions $\hat{Z}^\beta_\theta$ to show possible drawbacks of the common approaches. For a fair comparison, we perform hyperparameter search, and report the best for each domain; for filtering-based behavior cloning

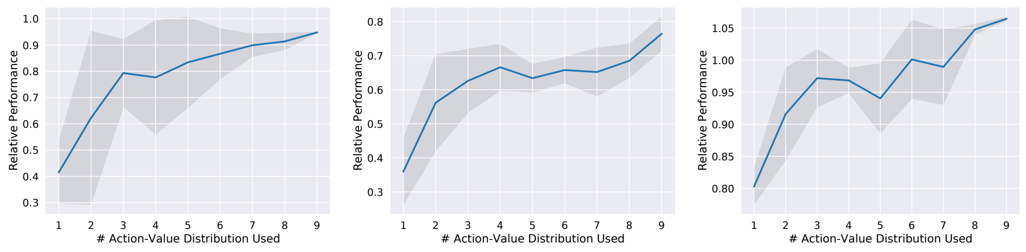
[1]`https://github.com/hiwonjoon/YOEO_public.git`

Table 2: Normalized performance of different policy implementations with the same action-value distribution. Average score over 100 trajectories is reported. On the right, we plot the normalized performance of the Z with SGD method over training iterations. The performance is measured for every 50,000 gradient descent steps, and results of different hyperparameters are shown with different colors.

		BC	**BC w/ Filter.**	**Z w/ SGD**	**YOEO**
Medium-Replay	Hopper	11.8	36.9	42.9	**83.4**
	Walker2D	11.3	21.5	30.8	**39.3**
Medium	Hopper	29.0	56.2	5.6	**63.4**
	Walker2D	6.6	28.9	**70.2**	**70.4**
Medium-Expert	Hopper	**111.9**	**111.6**	98.2	94.0
	Walker2D	6.4	6.0	34.4	**88.9**



(a) Hopper-med.-rep. (b) Walker2d-med.-rep.



(a) Hopper-medium-replay (b) Hopper-medium (c) Hopper-medium-expert

Figure 1: Normalized performance with the different number of Z models used for a policy. A shaded uncertainty band represents standard deviation of 3 different runs.

method, we test different filtering threshold, and for gradient optimization method, we test different entropy regularization coefficient. The details about implementation and other hyperparameters are provided in Appendix.

The results are shown in Table 2. Overall, YOEO outperforms the other two baselines for most of the environments. Also, it is noteworthy that even when the baselines show competitive performance, there was no common hyperparameter that consistently worked for all domains as well as YOEO; for instance, while policy gradient method with no entropy regularization showed the best result in hopper-medium-replay domain, but the same hyperparameter was the worst for walker2d-medium-replay domain. Furthermore, the performance fluctuated by the number of gradient descent steps while the loss decreased consistently over time. These results implies that tuning such an algorithm can be difficult and laborious.

5.1.2 Ensemble

We test the effect of the ensemble in YOEO by changing the number of trained $\hat{Z}^\beta$ used in the greedy policy. Each $\hat{Z}^\beta$ is trained independently with the same data with different initial parameter initialization. To build a nonparametric policy with multiple $\hat{Z}^\beta$s, we simply take the average over $\hat{Q}$s derived from each $\hat{Z}_\theta$. The results are included in Figure 1 and Figure A.1. We observe a clear positive correlation between the performance and the number of trained $\hat{Z}^\beta$ across all datasets.

5.1.3 Distributional Value Estimation and Adaptive Pessimism

We investigate the advantage and its effect of learning value distribution (Y and Z) instead of aggregated point estimate (Q). We use value distributions in three different ways: (1) data-driven pessimism, which controls a pessimism adaptively for each state and unseen action by generating a pessimistic target distribution with risk-averse distortion measure, (2) gap-requirement regularization, which adjusts the strength of pessimism based on the diversity of policies for each state using Y, and (3) risk-sensitive nonparametric policy, which becomes viable due to action-value distribution.

Table 3: Results on various ablations experiments related to distributional value estimation. The normalized performance are displayed.

		YOEO	**YOEO (Z@0.5)**	**YOEO w/ $k=0$**	**YOEO w/ $\eta=1, k=c$**	$Q-c$	$\hat{Z}_\psi$
Distributional		✓	✓	✓	✓	✗	✓
Data-Driven Pessimism		✓	✓	✓	✗	✗	✗
Gap Requirement		✓	✓	✗	✗	✗	✗
Risk-sensitive Policy		✓	✗	✓	✓	✗	✗
Medium-Replay	Hopper	83.4	74.5	**93.2**	82.8	25.5	22.7
	Walker2D	**39.3**	36.0	31.1	27.8	11.2	13.9
Medium	Hopper	63.4	**78.3**	65.8	58.0	31.9	5.7
	Walker2D	**70.4**	62.0	65.6	60.2	10.3	38.1
Medium-Expert	Hopper	94	95.2	**105.1**	38.7	56.4	27.3
	Walker2D	**88.9**	65.1	**86.9**	29.8	4.7	17.5

We test 5 different ablations: (1) YOEO without risk-sensitive policy distortion (denoted as **Z@0.5**), (2) YOEO without gap requirement (**YOEO w/ $k = 0$**), (3) YOEO without data-driven pessimism (**YOEO w/ $\eta = 1, k = c$**) in which $\hat{Z}_\theta$ is regularized with a some constant pessimistic penalty, (4) Q with constant pessimism (**$Q - c$**) in which Q is trained with constant pessimistic penalty, and (5) YOEO without any pessimism ($\hat{Z}_\psi$). For (3) and (4) that involve extra hyperparameter c, we perform hyperparameter search and report the best performance for each domain.

The result is shown in Table 3. We were able to confirm the effectiveness of the proposed pessimitic regularization; very large performance drop is observed when we do not regularize the action-value distribution. In addition, while the significance of each component varies for each dataset, we were able to confirm the importance of each component; while different versions of YOEO with data-driven pessimism show a competitive performance, the ablated algorithms without the feature show large degradation in performance; the risk-sensitive policy provides the large performance gain in -medium-replay datasets, which consist of diverse suboptimal trajectories.

6 Discussion

We investigate a simple baseline algorithm for offline RL that only evaluates the value function of the behavior policy. Since the proposed algorithm does not involve a policy optimization step and value re-evaluation based on the updated policy, the algorithm can be stable, but the resulting policy is more likely to be suboptimal. This makes the algorithm an appropriate baseline for actor-critic algorithms that ought to outperform this baseline if there is indeed value in iterative optimization in the offline setting. In the experiments, however, the proposed baseline shows competitive results on the several D4RL benchmarks, surpassing the state-of-the-art results in some tasks. This implies the usefulness of conservativeness under uncertainty, which can prevent incorrect generalization behavior of a complex function approximator occurring due to lack of data, and second, the potential flaws of iterative optimization in actor-critic algorithms in the offline setting. Therefore, it is essential to build a theoretical framework that sheds light on iterative optimization and generalization of offline actor-critic methods that use deep neural networks. Additionally, it would be interesting future work to develop a new practical regularization method leveraging the empirical generalization characteristics of deep neural networks rather than penalizing all the unseen state-action pairs indiscriminately. This will allow for better and safer policy optimization that fully utilizes the power of deep neural networks.

References

[1] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, page 449–458, 2017.

[2] Erik Bernhardsson. Approximate nearest neighbors oh yeah (annoy). `https://github.com/spotify/annoy`, 2020.

[3] Jacob Buckman, Carles Gelada, and Marc G Bellemare. The importance of pessimism in fixed-dataset policy optimization. *arXiv preprint arXiv:2009.06799*, 2020.

[4] Amir-massoud Farahmand, Csaba Szepesvári, and Rémi Munos. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, pages 568–576, 2010.

[5] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[6] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, 2019.

[7] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[8] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

[9] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[10] 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 32*, pages 11784–11794, 2019.

[11] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[12] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[13] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems*, page 6405–6416, 2017.

[14] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[15] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[16] 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 *International Conference on Learning Representations*, 2016.

[17] 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.

[18] Rémi Munos. Error bounds for approximate value iteration. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1006. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[19] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[20] Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *International conference on Machine learning*, pages 745–750, 2007.

[21] Noah Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2020.

[22] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2018.

[24] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[25] Joost van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In *International Conference on Machine Learning*, 2020.

[26] Qing Wang, Jiechao Xiong, Lei Han, Han Liu, Tong Zhang, et al. Exponentially weighted imitation learning for batched historical data. In *Advances in Neural Information Processing Systems*, pages 6288–6297, 2018.

[27] Ziyu Wang, Alexander Novikov, Konrad Żołna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

[28] Will, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, pages 1104–1113, 2018.

[29] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[30] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

A Experimental Details

For filtered behavior cloning, we calculate the advantage by comparing $\hat{Q}$ derived from $\hat{Z}_\theta$ without distortion to $\hat{Y}_\psi$ or $\hat{V}$ derived by the state-value distribution. We use two criteria for filtering: constant filtering and relative filtering; in constant filtering, we calculate the advantage $A(s,a) = \hat{Q}(s,a) - \hat{V}$ then take the state and action pairs which is larger than a threshold value Th_c. In relative filtering, we calculate the relative advantage $A(s,a) = P_{Y(s)}(V \leq \hat{Q}(s,a)) \in [0,1]$ and select pairs whose relative advantage is larger than some threshold Th_r. For Th_c, we tested $[0, 10, 30, 100]$, and for Th_r, we tested $[0.5, 0.65, 0.8]$. For the policy baseline that optimizes a parameterized policy via gradient descent, we implement a stochastic policy which models an independent normal distribution for each action dimension. We also add an entropy regularization term besides its maximization target $\hat{Q}$. With action-value distribution and ensemble into account, the loss function is formally defined as follow:

$$L_\pi = -\mathbb{E}_{s\sim D, a\sim\pi(a|s)}\big[\min_{i=1,\ldots,5} Z_i(s,a;0.1)\big] + \alpha H(\pi). \tag{18}$$

Following the implementation of CQL, we initialize the policy by optimizing a policy with a behavior cloning loss ($L_{\text{bc}} = -\mathbb{E}_{s,a\sim D}[\log \pi(a|s)]$) for 200,000 gradient descent steps. Then, we train 1 million time steps with the loss function shown above. We tested $\alpha = [0.0, 0.2, 0.5]$.

We provide the default hyperparameter used for the experiments in A.1. We use the provided hyperparameters unless mentioned otherwise for the ablation purpose.

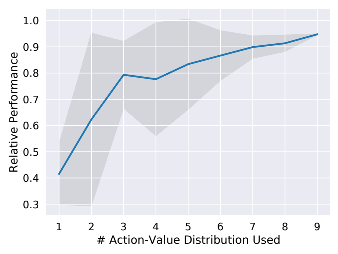
Table A.1: Hyperparameters used in the experiments

	Y_ψ	Z_ψ	Z_θ
# Training Iterations		1 million time steps	
Learning Rate	1.00E-04	1.00E-04	1.00E-03
ψ_E, θ_E	$\lvert S\rvert \rightarrow 256+\text{ReLU} \rightarrow 256+\text{ReLU} \rightarrow \lvert F\rvert$	$\lvert S\rvert+\lvert A\rvert \rightarrow 256+\text{ReLU} \rightarrow 256+\text{ReLU} \rightarrow \lvert F\rvert$	
ψ_f, θ_f		$\lvert F\rvert \rightarrow 256+\text{ReLU} \rightarrow 256+\text{ReLU} \rightarrow 1$	
Feature Dims ($\lvert F\rvert$)		64	
N, N'		16	
κ		1	
Discount rate (γ)		0.999	
Gap Requirement (α)		10	
Distortion for Pessimism (η)		0.1	
Distortion for Policy		0.2	
# Samples for $\hat{Q}$		20	
Num Ensembles		5	

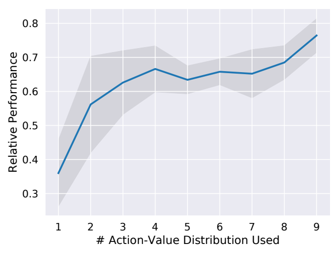
B Additional Experimental Results

Table A.2: Performance of YOEO and prior methods on a subset of D4RL benchmarks. Each number represents the performance relative to a random policy as 0 and an expert policy as 100. All the numbers except ours are borrowed from [5], [11]. The numbers are averaged over 3 different random seeds.

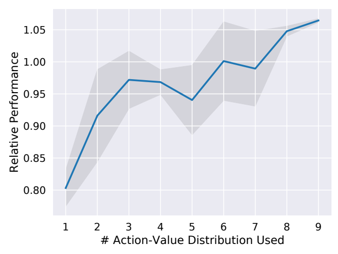
Type	Env.	BC	SAC-offline	BEAR	BRAC	AWR	BCQ	CQL ($\mathcal{H}$)	YOEO (Ens. 5)	YOEO (Ens. 9)
Random	HalfCheetah	2.1	30.5	25.1	31.2	2.5	2.2	**35.4**	7	7.2
	Hopper	9.8	11.3	11.4	12.2	10.2	10.6	**10.8**	9.8	9.8
	Walker2D	1.6	4.1	7.3	1.9	1.5	4.9	7	**8.3**	**8.1**
Medium-Replay	HalfCheetah	38.4	-2.4	38.6	**47.7**	40.3	38.2	**46.2**	33.1	34.4
	Hopper	11.8	3.5	33.7	0.6	28.4	33.1	48.6	83.4	**94.8**
	Walker2D	11.3	1.9	19.2	0.9	15.5	15	26.7	**39.3**	**41.5**
Medium	HalfCheetah	36.1	-4.3	41.7	**46.3**	37.4	40.7	**44.4**	35.3	35.6
	Hopper	29	0.8	52.1	31.1	35.9	54.5	58	**63.4**	**76.4**
	Walker2D	6.6	0.9	59.1	**81.1**	17.4	53.1	**79.2**	70.4	68.6
Medium-Expert	HalfCheetah	35.8	1.8	53.4	44.2	52.7	**64.7**	**62.4**	15	16.6
	Hopper	**111.9**	1.6	96.3	0.8	27.1	**110.9**	**111**	94	**106.4**
	Walker2D	6.4	-0.1	40.1	81.6	53.8	57.5	**98.7**	88.9	92.7
human	pen	34.4	6.3	-1	8.1	12.3	**68.9**	37.5	29.8	34.7
	door	0.5	3.9	-0.3	-0.3	0.4	0	**9.9**	0.7	0.4
	relocate	0	0	-0.3	-0.3	0	-0.1	**0.2**	0	0
	hammer	1.5	0.5	0.3	0.3	1.2	0.5	**4.4**	1	0.2
cloned	pen	56.9	23.5	26.5	1.6	28	44	39.2	**61.3**	**61.2**
	door	-0.1	0	-0.1	-0.1	0	0	0.4	**1.1**	**1.1**
	relocate	-0.1	-0.2	-0.3	-0.3	-0.2	-0.3	-0.1	-0.2	-0.2
	hammer	0.8	0.2	0.3	0.3	0.4	0.4	**2.1**	0.4	0.4



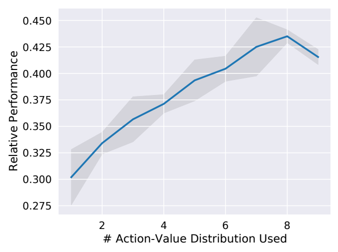
(a) Hopper-medium-replay



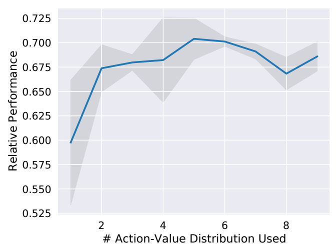
(b) Hopper-medium



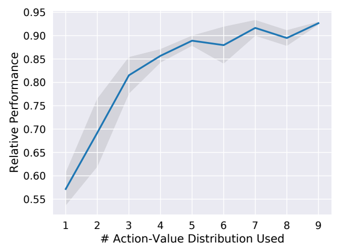
(c) Hopper-medium-expert



(d) Walker2d-medium-replay



(e) Walker2d-medium



(f) Walker2d-medium-expert

Figure A.1: Normalized performance with the different number of Z models used for a policy. A shaded uncertainty band represents standard deviation of 3 different runs.