Accelerating Online Reinforcement Learning with Offline Datasets

Ashvin Nair anair17@berkeley.edu
Department of EECS, UC Berkeley

Murtaza Dalal mdalal@berkeley.edu
Department of EECS, UC Berkeley

Abhishek Gupta abhigupta@eecs.berkeley.edu
Department of EECS, UC Berkeley

Sergey Levine svlevine@eecs.berkeley.edu
Department of EECS, UC Berkeley

Abstract

Reinforcement learning provides an appealing formalism for learning control policies from experience. However, the classic active formulation of reinforcement learning necessitates a lengthy active exploration process for each behavior, making it difficult to apply in real-world settings. If we can instead allow reinforcement learning to effectively use previously collected data to aid the online learning process, where the data could be expert demonstrations or more generally any prior experience, we could make reinforcement learning a substantially more practical tool. While a number of recent methods have sought to learn offline from previously collected data, it remains exceptionally difficult to train a policy with offline data and improve it further with online reinforcement learning. In this paper we systematically analyze why this problem is so challenging, and propose a novel algorithm that combines sample-efficient dynamic programming with maximum likelihood policy updates, providing a simple and effective framework that is able to leverage large amounts of offline data and then quickly perform online fine-tuning of reinforcement learning policies. We show that our method enables rapid learning of skills with a combination of prior demonstration data and online experience across a suite of difficult dexterous manipulation and benchmark tasks.

1. Introduction

Learning models that generalize effectively to complex open-world settings, from image recognition [10] to natural language processing [3], relies on large, high-capacity models and large, diverse, and representative datasets. Leveraging this recipe for reinforcement learning (RL) has the potential to yield powerful policies for real-world control applications such as robotics. However, while deep RL algorithms enable the use of large models, the use of large datasets for real-world RL is conceptually challenging. Most RL algorithms collect new data online every time a new policy is learned, which limits the size and diversity of the datasets for RL. In the same way that powerful models in computer vision and NLP are often pre-trained on large, general-purpose datasets and then fine-tuned on task-specific data, RL policies that generalize effectively to open-world settings will need to be able to incorporate large amounts of prior data effectively into the learning process, while still collecting additional data online for the task at hand.

For data-driven reinforcement learning, offline datasets consist of trajectories of states, actions and associated rewards. This data can potentially come from demonstrations for the desired task [21; 2], suboptimal policies [6], demonstrations for related tasks [27], or even just random exploration in the environment. Depending on the quality of the data that is provided, useful knowledge can be extracted about the dynamics of the world, about the task being solved, or both. Effective data-driven methods for deep reinforcement learning should be able to use this data to pre-train offline while improving with online fine-tuning. Since this prior data can come from a variety of sources, we require an algorithm that does not utilize different types of data in any privileged way. For example, prior methods that incorporate demonstrations into RL directly aim to mimic these demonstrations [15], which is desirable when the demonstrations are known to be optimal, but can cause undesirable bias when the prior data is not optimal. While prior methods for fully offline RL provide a mechanism for utilizing offline data [5; 11], as we will show in our experiments, such methods generally are not effective for fine-tuning with online data as they are often too conservative. In effect, prior methods require us to choose: Do we assume prior data is optimal or not? Do we use only offline data, or only online data? To make it feasible to learn policies for open-world settings, we need algorithms that contain all of the aforementioned qualities.

In this work, we study how to build RL algorithms that are effective for pre-training from a variety of off-policy datasets, but also well suited to continuous improvement with online data collection. We systematically analyze the challenges with using standard off-policy RL algorithms [8; 11; 1] for this problem, and introduce a simple actor critic algorithm that elegantly bridges data-driven pre-training from offline data and improvement with online data collection. Our method combines the best of supervised learning and actor-critic algorithms. Dynamic programming leverages off-policy data and enables sample-efficient learning. A simple supervised actor update implicitly enforces a constraint that mitigates the effects of out-of-distribution actions when learning from offline data [5; 11], while avoiding overly conservative updates. We evaluate our algorithm on a wide variety of robotic control and benchmark tasks across three simulated domains: dexterous manipulation, tabletop manipulation, and MuJoCo control tasks. We see that our algorithm, advantage weighted actor critic (AWAC), is able to quickly learn successful policies on difficult tasks with high action dimension and binary sparse rewards, significantly better than prior methods for off-policy and offline reinforcement learning. Moreover, we see that AWAC can utilize different types of prior data: demonstrations, suboptimal data, and random exploration data.

2. Preliminaries

We consider the standard reinforcement learning notation, with states $\mathbf{s}$, actions $\mathbf{a}$, policy $\pi(\mathbf{a}|\mathbf{s})$, rewards $r(\mathbf{s}, \mathbf{a})$, and dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. The discounted return is defined as $R_t = \sum_{i=t}^{T} \gamma^i r(\mathbf{s}_i, \mathbf{a}_i)$, for a discount factor γ and horizon T which may be infinite. The objective of an RL agent is to maximize the expected discounted return $J(\pi) = \mathbb{E}_{p_\pi(\tau)}[R_0]$ under the distribution induced by the policy. The optimal policy can be learned by direct optimization of this objective using the policy gradient, estimating $\nabla J(\pi)$ [25], but this is often ineffective due to high variance of the estimator. Many algorithms attempt to reduce this variance by making use of the value function $V^\pi(\mathbf{s}) = \mathbb{E}_{p_\pi(\tau)}[R_t|\mathbf{s}]$, action-value function $Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{p_\pi(\tau)}[R_t|\mathbf{s}, \mathbf{a}]$, or advantage $A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s})$. The action-value function for a policy can be written recursively via the Bellman equation: $Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')] = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}', \mathbf{a}')]]$. Instead of estimating policy gradients directly, actor-critic algorithms maximize returns by alternating between two phases [9]: policy evaluation and policy improvement. During the policy evaluation phase, the critic $Q^\pi(\mathbf{s}, \mathbf{a})$ is estimated for the current policy π. This can be accomplished by repeatedly applying the Bellman operator $\mathcal{B}$, corresponding to the right-hand side of Equation **??**, $\mathcal{B}^\pi Q(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}', \mathbf{a}')]]$. By iterating according to $Q^{k+1} = \mathcal{B}^\pi Q^k$, Q^k converges to Q^π [23]. With function approximation, we cannot apply the Bellman operator exactly, and instead minimize the Bellman error with respect to Q-function parameters ϕ_k:

$$\phi_k = \arg\min_\phi \ \mathbb{E}_\mathcal{D}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2], y = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}',\mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]. \tag{1}$$

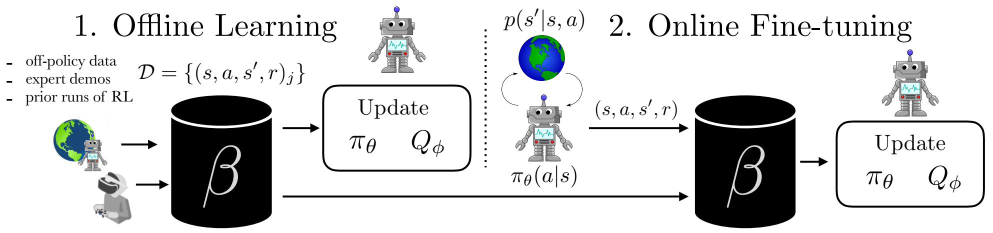


Figure 1: We study learning policies by offline learning on a prior dataset $\mathcal{D}$ and then fine-tuning with online interaction. The prior data could be obtained via prior runs of RL, expert demonstrations, or any other source of transitions. Our method, advantage weighted actor critic (AWAC) is able to learn effectively from offline data and fine-tune in order to reach expert-level performance after collecting a limited amount of interaction data. Videos are available at awacrl.github.io.

During policy improvement, the actor π is typically updated based on the current estimate of Q^π. A commonly used technique [13; 4; 8] is to update the actor via likelihood ratio or pathwise derivatives: $\pi_{\theta_k}(\mathbf{a}|\mathbf{s}) = \arg\max_\theta \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\mathbb{E}_{\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_{\phi_k}(\mathbf{s},\mathbf{a})]]$. Actor-critic algorithms are widely used in deep RL [14; 13; 8; 4]. With a Q-function estimator, they can in principle utilize off-policy data when used with a replay buffer for storing prior transition tuples, which we will denote β.

3. Challenges in Offline RL with Online Fine-tuning

In this section, we aim to better understand the unique challenges that exist when pre-training using offline data, followed by fine-tuning with online data collection. We first describe the problem, and then analyze what makes this problem difficult for prior methods.

Problem definition. We assume that a static dataset of transitions, $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$, is provided to the algorithm at the beginning of training. This dataset can be sampled from an arbitrary policy or mixture of policies, and may even be collected by a human expert. This definition is general and encompasses many scenarios, such as learning from demonstrations, learning from random data, learning from prior RL experiments, or even learning from multi-task data. Given the dataset $\mathcal{D}$, our goal is to leverage $\mathcal{D}$ for pre-training and use some online interaction to learn the optimal policy $\pi^*(\mathbf{a}|\mathbf{s})$, with as few interactions with the environment as possible (depicted in Fig 1). This setting is representative of many real-world RL settings, where prior data is available and the aim is to learn new skills efficiently. We first study existing algorithms empirically in this setting on the HalfCheetah-v2 Gym environment. The prior dataset consists of 15 demonstrations from an expert policy and 100 suboptimal trajectories sampled from a behavioral clone of these demonstrations.

3.1) Data efficiency. One of the simplest ways to utilize prior data such as demonstrations for RL is to pre-train a policy with imitation learning, and fine-tune with on-policy RL [7; 19]. This has two drawbacks: (1) prior data may not be optimal; (2) on-policy fine-tuning is data inefficient as it does not reuse the prior data in the RL stage. In our setting, data efficiency is vital. To this end, we require algorithms that are able to reuse arbitrary off-policy data during online RL for data-efficient fine-tuning. We find that algorithms that use on-policy fine-tuning [19; 7], or Monte-Carlo return estimation [16; 17] are generally much less efficient than off-policy actor-critic algorithm, which iterate between improving π and estimating Q^π via Bellman backups. This can be seen from the results in Figure 2 plot 1, where on-policy methods like DAPG [19] and Monte-Carlo return methods like AWR [16] are an order of magnitude slower than off-policy actor-critic methods. Actor-critic methods, shown in Figure 2 plot 2, can in principle use off-policy data. However, as we will discuss next, naïvely applying these algorithms to our problem does not perform well due to different challenges.

3.2) Bootstrap Error in Offline Learning with Actor-Critic Methods. When standard off-policy actor-critic methods are applied to this problem setting, they perform poorly, as shown in the second plot in Figure 2: despite having a prior dataset in the replay buffer, these algorithms

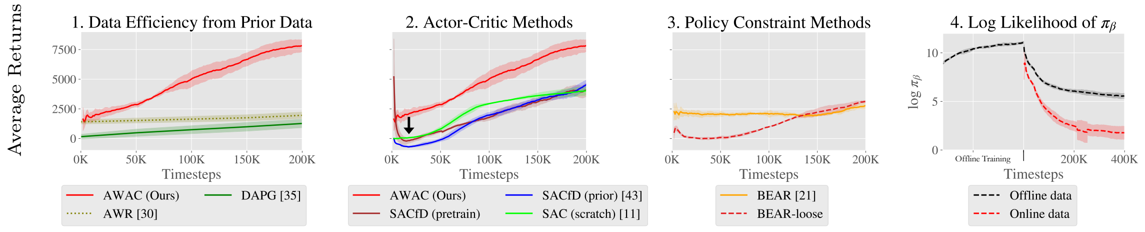


Figure 2: Analysis of prior methods on HalfCheetah-v2 using offline RL with online fine-tuning. (1) DAPG [19] and AWR [16] learn relatively slowly, even with access to prior data. We present our method, AWAC, as an example of how off-policy RL methods can learn much faster. (2) Soft actor-critic (SAC) with offline training (performed before timestep 0) and fine-tuning. We see a "dip" in the initial performance, even if the policy is pretrained with behavioral cloning. (3) Offline RL method BEAR [11] on offline training and fine-tuning, including a "loose" variant of BEAR with a weakened constraint. Standard offline RL methods fine-tune slowly, while the "loose" BEAR variant experiences a similar dip as SAC. (4) We show that the fit of the behavior models $\hat{\pi}_\beta$ used by these offline methods degrades as new data is added to the buffer during fine-tuning, potentially explaining their poor fine-tuning performance.

do not benefit significantly from offline training. We evaluate soft actor critic [8], a state-of-the-art actor-critic algorithm for continuous control. Note that "SAC (scratch)," which does not receive the prior data, performs similarly to "SACfD (prior)," which does have access to the prior data, indicating that the off-policy RL algorithm is not actually able to make use of the off-policy data for pre-training. Moreover, even if the SAC is policy is pre-trained by behavior cloning, labeled "SACfD (pretrain)", we still observe an initial decrease in performance.

This challenge can be attributed to off-policy bootstrapping error accumulation, as observed in several prior works [23; 11; 26; 12; 5]. In actor-critic algorithms, the target value $Q(\mathbf{s}', \mathbf{a}')$, with $\mathbf{a}' \sim \pi$, is used to update $Q(\mathbf{s}, \mathbf{a})$. When $\mathbf{a}'$ is outside of the data distribution, $Q(\mathbf{s}', \mathbf{a}')$ will be inaccurate, leading to accumulation of error on static datasets. Prior offline RL algorithms [5; 11; 26] propose to address this issue by explicitly adding constraints on the policy improvement update. Here, π_θ is the actor being updated and $\pi_\beta(a|s)$ represents the (potentially unknown) distribution from which all of the data seen so far (both offline data and online data) was generated. In the case of a replay buffer, π_β corresponds to a mixture distribution over all past policies. Typically, π_β is not known, especially for offline data, and must be estimated from the data itself. Many offline RL algorithms [11; 5; 22] explicitly fit a parametric model to samples for the distribution π_β via maximum likelihood estimation, where samples from π_β are obtained simply by sampling uniformly from the data seen thus far: $\hat{\pi}_\beta = \max_{\hat{\pi}_\beta} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi_\beta}[\log \hat{\pi}_\beta(\mathbf{a}|\mathbf{s})]$. After estimating $\hat{\pi}_\beta$, prior methods use it in various ways, including penalties on the policy update [11; 26] or architecture choices for sampling actions for policy training [5; 22]. As we will see next, the requirement for accurate estimation of $\hat{\pi}_\beta$ makes these methods difficult to use with online fine-tuning.

3.3) Excessively Conservative Online Learning. While offline RL algorithms with constraints [11; 5; 26] perform well offline, they struggle to improve with fine-tuning, as shown in the third plot in Figure 2. We see that the purely offline RL performance (at "0K" in Fig. 2) is much better than the standard off-policy methods shown in Section 3.2. However, with additional iterations of online fine-tuning, the performance increases very slowly (as seen from the slope of the BEAR curve in Fig 2). This can be attributed to challenges in fitting an accurate behavior model as data is collected online during finetuning. In the offline setting, behavior models must only be trained once via maximum likelihood, but in the online setting, the behavior model must be updated online to track incoming data. Training density models online (in the "streaming" setting) is a challenging research problem [20], made more difficult by a potentially complex multi-modal behavior distribution induced by the mixture of online and offline data. To understand this, we plot the log likelihood of learned behavior models on the dataset during online and offline training for the HalfCheetah task. As we can see in the plot, the accuracy of the behavior models ($\log \pi_\beta$ on the y axis) reduces during online fine-tuning, indicating that it is not fitting the new data well during online training. When the behavior models are inaccurate or unable to model new data well, constrained optimization becomes too conservative, resulting in limited improvement with fine-tuning. This analysis suggests that, in order to address our problem setting, we require an off-policy RL algorithm that constrains the policy to prevent offline instability and error accumulation, but is not so conservative that it prevents online fine-tuning due to imperfect behavior modeling. Our proposed algorithm accomplishes this by employing an *implicit* constraint, which does not require any explicit modeling of the behavior policy.

4. Advantage Weighted Actor Critic: A Simple Algorithm for Fine-tuning from Offline Datasets

In this section, we will describe the advantage weighted actor-critic (AWAC) algorithm, which trains an off-policy critic and an actor with an *implicit* policy constraint. We will show AWAC mitigates the challenges outlined in Section 3. AWAC follows the standard paradigm for actor-critic algorithms as described in Section 2, with a policy evaluation step to learn Q^π and a policy improvement step to update π. AWAC uses off-policy temporal-difference learning to estimate Q^π in the policy evaluation step, and a unique policy improvement update that is able to obtain the benefits of offline RL algorithms at training from prior datasets, while avoiding overly conservative behavior.

Policy improvement for AWAC proceeds by learning a policy that maximizes the value of the critic learned in the policy evaluation step via TD bootstrapping. If done naively, this can lead to the issues described in Section 3.3, but we can avoid the challenges of bootstrap error accumulation by restricting the policy distribution to stay close to the data observed thus far during the actor update, while maximizing the value of the critic. At iteration k, AWAC therefore optimizes the policy to maximize the estimated Q-function $Q^{\pi_k}(\mathbf{s},\mathbf{a})$ at every state, while constraining it to stay close to the actions observed in the data, similar to prior offline RL methods, though this constraint will be enforced differently. Note from the definition of the advantage in Section 2 that optimizing $Q^{\pi_k}(\mathbf{s},\mathbf{a})$ is equivalent to optimizing $A^{\pi_k}(\mathbf{s},\mathbf{a})$. We can therefore write this optimization as:

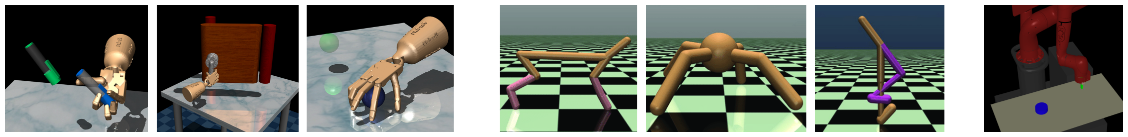
$$\pi_{k+1} = \arg\max_{\pi\in\Pi} \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] \text{ s.t. } D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s})) \leq \epsilon. \tag{2}$$

As we saw in Section 3.2, enforcing the constraint by incorporating an explicit learned behavior model [11; 5; 26; 22] leads to poor fine-tuning performance. Instead, we will enforce the constraint *implicitly*, without explicitly learning a behavior model. We first derive the solution to the constrained optimization in Equation 2 to obtain a non-parametric closed form for the actor. This solution is then projected onto the parametric policy class *without* any explicit behavior model. The analytic solution to Equation 2 can be obtained by enforcing the KKT conditions [17; 18; 16]. The Lagrangian is $\mathcal{L}(\pi,\lambda) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] + \lambda(\epsilon - D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s})))$, and the closed form solution to this problem is $\pi^*(\mathbf{a}|\mathbf{s}) = \frac{1}{Z(\mathbf{s})}\pi_\beta(\mathbf{a}|\mathbf{s})\exp\left(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s},\mathbf{a})\right)$, where $Z(s) = \int_{\mathbf{a}} \pi_\beta(\mathbf{a}|\mathbf{s})\exp(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s},\mathbf{a}))d\mathbf{a}$ is the normalizing partition function. When using function approximators, such as deep neural networks as we do in our implementation, we need to project the non-parametric solution into our policy space. For a policy π_θ with parameters θ, this can be done by minimizing the KL divergence of π_θ from the optimal non-parametric solution π^* under the data distribution $\rho_{\pi_\beta}(\mathbf{s})$:

$$\arg\min_\theta \mathop{\mathbb{E}}_{\rho_{\pi_\beta}(\mathbf{s})}\left[D_{\mathrm{KL}}(\pi^*(\cdot|\mathbf{s})||\pi_\theta(\cdot|\mathbf{s}))\right] = \arg\min_\theta \mathop{\mathbb{E}}_{\rho_{\pi_\beta}(\mathbf{s})}\left[\mathop{\mathbb{E}}_{\pi^*(\cdot|\mathbf{s})}[-\log\pi_\theta(\cdot|\mathbf{s})]\right] \tag{3}$$

Note that the parametric policy could be projected with either direction of KL divergence. Choosing the reverse KL results in explicit penalty methods [26] that rely on evaluating π_β. Instead, by using forward KL, we can compute the policy update by sampling from β:

$$\theta_{k+1} = \arg\max_\theta \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\beta}\left[\log\pi_\theta(\mathbf{a}|\mathbf{s})\frac{1}{Z(\mathbf{s})}\exp\left(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s},\mathbf{a})\right)\right]. \tag{4}$$



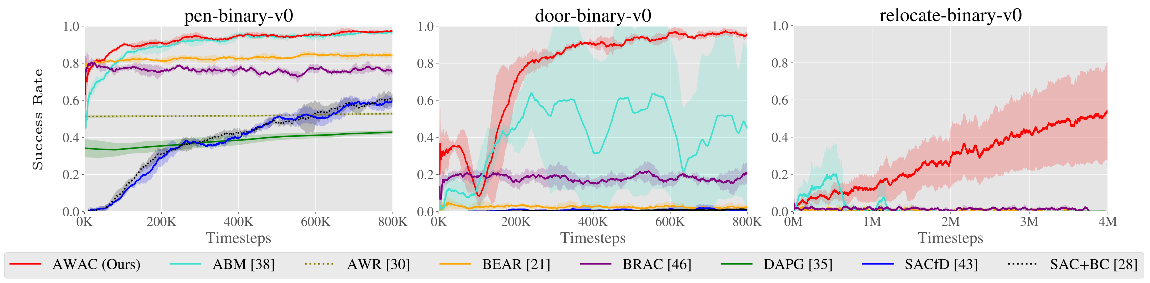


Figure 3: Comparative evaluation on the dexterous manipulation tasks. These tasks are difficult due to their high action dimensionality and reward sparsity. We see that AWAC is able to learn these tasks with little online data collection required (100K samples $\approx$ 16 minutes of equivalent real-world interaction time). Meanwhile, most prior methods are not able to solve the harder two tasks: door opening and object relocation.

This actor update amounts to weighted maximum likelihood (i.e., supervised learning), where the targets are obtained by re-weighting the state-action pairs observed in the current dataset by the predicted advantages from the learned critic, *without* explicitly learning any parametric behavior model, simply sampling (s, a) from the replay buffer β. See Appendix A.4 for a more detailed derivation and Appendix A.5 for specific implementation details.

5. Experimental Evaluation

In our experiments, we first compare our method against prior methods in the offline training and fine-tuning setting. We show that we can learn difficult, high-dimensional, sparse reward dexterous manipulation problems from human demonstrations and off-policy data. Further experiments fine-tuning from suboptimal data and on simpler MuJoCo environments can be found in the appendix.

6.1) Comparative Evaluation on Dexterous Manipulation Tasks. We aim to study tasks representative of the difficulties of real-world robot learning, where offline learning and online fine-tuning are most relevant. One such setting is the suite of dexterous manipulation tasks proposed by Rajeswaran et al. [19]. These environments exhibit many challenges: high dimensional action spaces, complex manipulation physics with many intermittent contacts, and randomized hand and object positions. The reward functions in these environments are binary 0-1 rewards for task completion. Rajeswaran et al. [19] provide 25 human demonstrations for each task, which are not fully optimal but do solve the task. Since this dataset is very small, we generated another 500 trajectories of interaction data by sampling from a behavioral cloned policy.

First, we compare our method on the dexterous manipulation tasks described earlier against prior methods for off-policy learning, offline learning, and bootstrapping from demonstrations. Specific implementation details are discussed in Appendix A.6. The results are shown in Fig. 3. Our method is able to leverage the prior data to quickly attain good performance, and the efficient off-policy actor-critic component of our approach fine-tunes much more quickly than other methods. While the baseline comparisons and ablations are able to make some amount of progress on the pen task, alternative off-policy RL and offline RL algorithms are largely unable to solve the door and relocate task in the time-frame considered. We find that the design decisions to use off-policy critic estimation allow AWAC to significantly outperform AWR [16] while the implicit behavior modeling allows AWAC to significantly outperform ABM [22], although ABM does make some progress. Rajeswaran et al. [19] show that DAPG can eventually solve these tasks but much slower.

References

[1] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a Posteriori Policy Optimisation. In *International Conference on Learning Representations (ICLR)*, pages 1–19, 2018.

[2] C. G. Atkeson and S. Schaal. Robot Learning From Demonstration. In *International Conference on Machine Learning (ICML)*, 1997. URL `http://www.cc.gatech.edu/fac/fChris.`

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Association for Compuational Linguistics (ACL)*, oct 2019. URL `http://arxiv.org/abs/1810.04805`.

[4] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. *International Conference on Machine Learning (ICML)*, 2018.

[5] S. Fujimoto, D. Meger, and D. Precup. Off-Policy Deep Reinforcement Learning without Exploration. In *International Conference on Machine Learning (ICML)*, dec 2019. URL `http://arxiv.org/abs/1812.02900`.

[6] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell. Reinforcement learning from imperfect demonstrations. *CoRR*, abs/1802.05313, 2018. URL `http://arxiv.org/abs/1802.05313`.

[7] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning. In *Conference on Robot Learning (CoRL)*, oct 2019. URL `http://arxiv.org/abs/1910.11956`.

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*, 2018. URL `https://arxiv.org/pdf/1801.01290.pdf`.

[9] V. R. Konda and J. N. Tsitsiklis. Actor-Critic Algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2000.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[11] A. Kumar, J. Fu, G. Tucker, and S. Levine. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In *Neural Information Processing Systems (NeurIPS)*, jun 2019. URL `http://arxiv.org/abs/1906.00949`.

[12] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. Technical report, 2020.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016. ISBN 0-7803-3213-X. doi: 10.1613/jair.301. URL `https://arxiv.org/pdf/1509.02971.pdf`.

[14] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, K. Kavukcuoglu, K. Com, and G. Deepmind. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 2016. URL `https://arxiv.org/pdf/1602.01783.pdf`.

[15] A. Nair, B. Mcgrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming Exploration in Reinforcement Learning with Demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. URL `https://arxiv.org/pdf/1709.10089.pdf`.

[16] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. sep 2019. URL `http://arxiv.org/abs/1910.00177`.

[17] J. Peters and S. Schaal. Reinforcement Learning by Reward-weighted Regression for Operational Space Control. In *International Conference on Machine Learning*, 2007. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.6266{&}rep=rep1{&}type=pdf`.

[18] J. Peters, K. Mülling, and Y. Altün. Relative Entropy Policy Search. In *AAAI Conference on Artificial Intelligence*, pages 1607–1612, 2010. URL `https://pdfs.semanticscholar.org/ff47/526838ce85d77a50197a0c5f6ee5095156aa.pdfhttp://www-clmc.usc.edu/publications/P/Peters{_}POTTNCOAIPGAT{_}2010.pdf`.

[19] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Robotics: Science and Systems*, 2018. URL `https://arxiv.org/pdf/1709.10087.pdf`.

[20] J. Ramapuram, M. Gregorova, and A. Kalousis. Lifelong Generative Modeling. *Neurocomputing*, may 2017. URL `http://arxiv.org/abs/1705.09847`.

[21] S. Schaal. Learning from demonstration. In *Advances in Neural Information Processing Systems (NeurIPS)*, number 9, pages 1040–1046, 1997. ISBN 1558604863. doi: 10.1016/j.robot.2004.03.001. URL `http://www.cc.gatech.edulfachttp://wwwiaim.ira.uka.de/users/rogalla/WebOrdnerMaterial/ml-robotlearning.pdfhttp://www.cc.gatech.edulfac/Stefan.Schaal`.

[22] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, N. Heess, and M. Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning, 2020.

[23] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* 1998. URL `http://incompleteideas.net/sutton/book/bookdraft2016sep.pdfhttps://webdocs.cs.ualberta.ca/{~}sutton/book/bookdraft2016sep.pdf`.

[24] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *CoRR*, abs/1707.0, 2017. URL `https://arxiv.org/pdf/1707.08817.pdf`.

[25] R. J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, pages 229–256, 1992.

[26] Y. Wu, G. Tucker, and O. Nachum. Behavior Regularized Offline Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, nov 2020. URL `http://arxiv.org/abs/1911.11361`.

[27] A. Zhou, E. Jang, D. Kappler, A. Herzog, M. Khansari, P. Wohlhart, Y. Bai, M. Kalakrishnan, S. Levine, and C. Finn. Watch, try, learn: Meta-learning from demonstrations and reward. *CoRR*, abs/1906.03352, 2019. URL `http://arxiv.org/abs/1906.03352`.

Appendix A. Appendix

A.1 Fine-Tuning from Random Policy Data

An advantage of using off-policy RL for reinforcement learning is that we can also incorporate suboptimal data, rather than only demonstrations. In this experiment, we evaluate on a simulated tabletop pushing environment with a Sawyer robot (shown in Fig 3), described further in Appendix A.3. To study the potential to learn from suboptimal data, we use an off-policy dataset of 500 trajectories generated by a random process. The task is to push an object to a target location in a 40cm x 20cm goal space.

The results are shown in Figure 4. We see that while many methods begin at the same initial performance, AWAC learns the fastest online and is actually able to make use of the offline dataset effectively as opposed to some methods which are completely unable to learn.

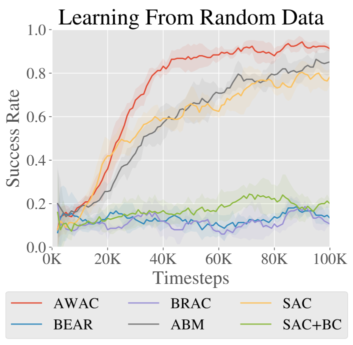


Figure 4: Comparison of fine-tuning from an initial dataset of suboptimal data on a Sawyer robot pushing task.

A.2 Analysis on MuJoCo Benchmarks from Prior Data

Since the dexterous manipulation environments are challenging to solve, we provide a comparative evaluation on MuJoCo benchmark tasks for analysis. On these simpler problems, many prior methods are able to learn, but it allows us to understand more precisely which design decisioins are crucial. For each task, we collect 15 demonstration trajectories using a pre-trained expert on each task, and 100 trajectories of off-policy data by rolling out a behavioral cloned policy trained on the demonstrations. The same data is made available to all methods. The results are presented in Figure 5. AWAC is consistently the best-performing method, but several other methods show reasonable performance. We summarize the results according to the challenges in Section 3.

Data efficiency. The two methods that do not estimate Q^π are DAPG [1] and AWR [16]. Across all three tasks, we see that these methods are somewhat worse offline than the best performing offline methods, and exhibit steady but slow improvement.

Bootstrap error in offline off-policy learning. For SAC [8], across all three tasks, we see that the offline performance at epoch 0 is generally poor. Due to the data in the replay buffer, SAC with prior data does learn faster than from scratch, but AWAC is faster to solve the tasks in general. SAC with additional data in the replay buffer is similar to the approach proposed by Večerík et al. [24]. SAC+BC reproduces Nair et al. [15] but uses SAC instead of DDPG [13] as the underlying RL algorithm. We find that these algorithms exhibit a characteristic dip at the start of learning.

Conservative online learning. Finally, we consider conservative offline algorithms: ABM [22], BEAR [11], and BRAC [26]. We found that BRAC performs similarly to SAC for working hyperpa-

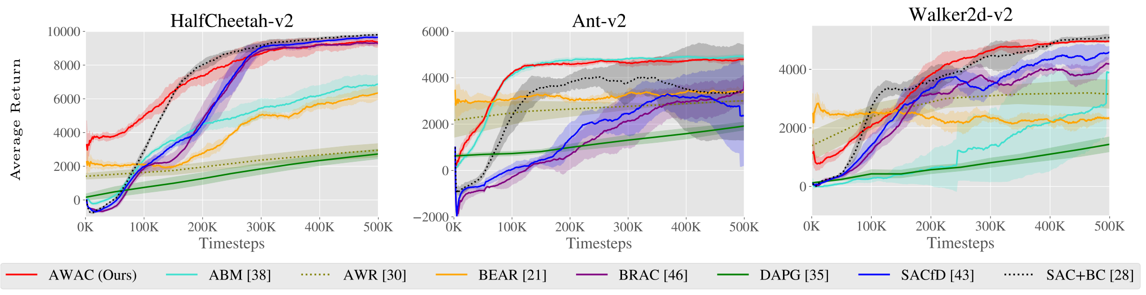


Figure 5: Comparison of our method and prior methods on standard MuJoCo benchmark tasks.

rameters. BEAR trains well offline - on Ant and Walker2d, BEAR significantly outperforms prior methods before online experience. However, online improvement is slow for BEAR and the final performance across all three tasks is much lower than AWAC. The closest in performance to our method is ABM, which is comparable on Ant-v2, but much slower on other domains.

A.3 Environment-Specific Details

We evaluate our method on three domains: dexterous manipulation environments, Sawyer manipulation environments, and MuJoCo benchmark environments. In the following sections we describe specific details.

A.3.1 Dexterous Manipulation Environments

These environments are modified from those proposed by by Rajeswaran et al. [19], and available in this repository.

pen-binary-v0. The task is to spin a pen into a given orientation. The action dimension is 24 and the observation dimension is 45. Let the position and orientation of the pen be denoted by x_p and x_o respectively, and the desired position and orientation be denoted by d_p and d_o respectively. The reward function is $r = \mathbb{1}_{|x_p - d_p| \leq 0.075} \mathbb{1}_{|x_o \cdot d_o| \leq 0.95}$ - 1. In Rajeswaran et al. [19], the episode was terminated when the pen fell out of the hand; we did not include this early termination condition.

door-binary-v0. The task is to open a door, which requires first twisting a latch. The action dimension is 28 and the observation dimension is 39. Let d denote the angle of the door. The reward function is $r = \mathbb{1}_{d > 1.4}$ - 1.

relocate-binary-v0. The task is to relocate an object to a goal location. The action dimension is 30 and the observation dimension is 39. Let x_p denote the object position and d_p denote the desired position. The reward is $r = \mathbb{1}_{|x_p - d_p| \leq 0.1}$ - 1.

A.3.2 Sawyer Manipulation Environment

SawyerPush-v0. This environment is included in the Multiworld library. The task is to push a puck to a goal position in a 40cm x 20cm, and the reward function is the negative distance between the puck and goal position. When using this environment, we use hindsight experience replay for goal-conditioned reinforcement learning. The random dataset for prior data was collected by rolling out an Ornstein-Uhlenbeck process with $\theta = 0.15$ and $\sigma = 0.3$.

A.4 Algorithm Derivation Details

The full optimization problem we solve, given the previous off-policy advantage estimate A^{π_k} and buffer distribution π_β is:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] \tag{5}$$

$$\text{s.t. } D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s}) || \pi_\beta(\cdot|\mathbf{s})) \leq \epsilon \tag{6}$$

$$\int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) d\mathbf{a} = 1. \tag{7}$$

Our derivation follows Peters et al. [18] and Peng et al. [16]. The analytic solution for the constrained optimization problem above can be obtained by enforcing the KKT conditions. The Lagrangian is:

$$\mathcal{L}(\pi, \lambda, \alpha) = \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] + \lambda(\epsilon - D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s}) || \pi_\beta(\cdot|\mathbf{s}))) + \alpha(1 - \int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) d\mathbf{a}). \tag{8}$$

Differentiating with respect to π gives:

$$\frac{\partial \mathcal{L}}{\partial \pi} = A^{\pi_k}(\mathbf{s}, \mathbf{a}) - \lambda \log \pi_\beta(\mathbf{a}|\mathbf{s}) + \lambda \log \pi(\mathbf{a}|\mathbf{s}) + \lambda - \alpha. \tag{9}$$

Setting $\frac{\partial \mathcal{L}}{\partial \pi}$ to zero and solving for π gives the closed form solution to this problem:

$$\pi^*(\mathbf{a}|\mathbf{s}) = \frac{1}{Z(\mathbf{s})} \pi_\beta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right), \tag{10}$$

Next, we project the solution into the space of parametric policies. For a policy π_θ with parameters θ, this can be done by minimizing the KL divergence of π_θ from the optimal non-parametric solution π^* under the data distribution $\rho_{\pi_\beta}(\mathbf{s})$:

$$\arg\min_\theta \mathop{\mathbb{E}}_{\rho_{\pi_\beta}(\mathbf{s})} \left[D_{\mathrm{KL}}(\pi^*(\cdot|\mathbf{s}) || \pi_\theta(\cdot|\mathbf{s})) \right] = \arg\min_\theta \mathop{\mathbb{E}}_{\rho_{\pi_\beta}(\mathbf{s})} \left[\mathop{\mathbb{E}}_{\pi^*(\cdot|\mathbf{s})} [-\log \pi_\theta(\cdot|\mathbf{s})] \right] \tag{11}$$

Note that in the projection step, the parametric policy could be projected with either direction of KL divergence. However, choosing the reverse KL direction has a key advantage: it allows us to optimize θ as a maximum likelihood problem with an expectation over data $s, a \sim \beta$, rather than sampling actions from the policy that may be out of distribution for the Q function. In our experiments we show that this decision is vital for stable off-policy learning.

Furthermore, assume discrete policies with a minimum probably density of $\pi_\theta \geq \alpha_\theta$. Then the upper bound:

$$\begin{aligned} D_{\mathrm{KL}}(\pi^* || \pi_\theta) &\leq \frac{2}{\alpha_\theta} D_{\mathrm{TV}}(\pi^*, \pi_\theta)^2 \quad (12) \\ &\leq \frac{1}{\alpha_\theta} D_{\mathrm{KL}}(\pi_\theta || \pi^*) \quad (13) \end{aligned}$$

holds by the Pinsker's inequality, where D_{TV} denotes the total variation distance between distributions. Thus minimizing the reverse KL also bounds the forward KL. Note that we can control the minimum α if desired by applying Laplace smoothing to the policy.

A.5 Implementation Details

We implement the algorithm building on top of twin delayed deep deterministic policy gradient (TD3) from [4]. The base hyperparameters are given in table 1.

The policy update is replaced with:

$$\theta_{k+1} = \arg\max_\theta \mathop{\mathbb{E}}_{\mathbf{s}, \mathbf{a} \sim \beta} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \frac{1}{Z(\mathbf{s})} \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right) \right]. \tag{14}$$

We found that explicitly computing $Z(s) = \int_{\mathbf{a}} \pi_\beta(\mathbf{a}|\mathbf{s}) \exp(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})) d\mathbf{a}$ results in worse performance, so we ignore the effect of $Z(s)$ and empirically find that this results in strong performance both offline and online.

The Lagrange multiplier λ is a hyperparameter. In this work we use $\lambda = 0.3$ for the manipulation environments and $\lambda = 1.0$ for the MuJoCo benchmark environments. One could adaptively learn λ with a dual gradient descent procedure, but this would require access to π_β.

As rewards for the dextrous manipulation environments are non-positive, we clamp the Q value for these experiments to be at most zero. We find this stabilizes training slightly.

Hyper-parameter	**Value**
Training Batches Per Timestep	1
Exploration Noise	None (stochastic policy)
RL Batch Size	1024
Discount Factor	0.99
Reward Scaling	1
Replay Buffer Size	1000000
Number of pretraining steps	25000
Policy Hidden Sizes	$[256, 256, 256, 256]$
Policy Hidden Activation	ReLU
Policy Weight Decay	10^{-4}
Policy Learning Rate	3×10^{-4}
Q Hidden Sizes	$[256, 256, 256, 256]$
Q Hidden Activation	ReLU
Q Weight Decay	0
Q Learning Rate	3×10^{-4}
Target Network τ	5×10^{-3}

Table 1: Hyper-parameters used for RL experiments.

Name	$\hat{Q}$	Policy Objective	$\hat{\pi}_\beta$?	Constraint
SAC	Q^π	$D_{\text{KL}}(\pi_\theta \vert\vert \bar{Q})$	No	None
SAC + BC	Q^π	Mixed	No	None
BCQ	Q^π	$D_{\text{KL}}(\pi_\theta \vert\vert \bar{Q})$	Yes	Support (ℓ^∞)
BEAR	Q^π	$D_{\text{KL}}(\pi_\theta \vert\vert \bar{Q})$	Yes	Support (MMD)
AWR	Q^β	$D_{\text{KL}}(\bar{Q} \vert\vert \pi_\theta)$	No	Implicit
MPO	Q^π	$D_{\text{KL}}(\bar{Q} \vert\vert \pi_\theta)$	Yes*	Prior
ABM-MPO	Q^π	$D_{\text{KL}}(\bar{Q} \vert\vert \pi_\theta)$	Yes	Learned Prior
DAPG	-	$J(\pi_\theta)$	No	None
BRAC	Q^π	$D_{\text{KL}}(\pi_\theta \vert\vert \bar{Q})$	Yes	Explicit KL penalty
AWAC (Ours)	Q^π	$D_{\text{KL}}(\bar{Q} \vert\vert \pi_\theta)$	No	Implicit

Figure 6: Comparison of prior algorithms that can incorporate prior datasets. See section A.6 for specific implementation details.

A.6 Baseline Implementation Details

We used public implementations of prior methods (DAPG, AWR) when available. We implemented the remaining algorithms in our framework, which also allows us to understand the effects of changing individual components of the method. In the section, we describe the implementation details. The full overview of algorithms is given in Figure 6.

Behavior Cloning (BC). This method learns a policy with supervised learning on demonstration data.

Soft Actor Critic (SAC). Using the soft actor critic algorithm from [8], we follow the exact same procedure as our method in order to incorporate prior data, initializing the policy with behavior cloning on demonstrations and adding all prior data to the replay buffer.

Behavior Regularized Actor Critic (BRAC). We implement BRAC as described in [26] by adding policy regularization $\log(\pi_\beta(a|s))$ where π_β is a behavior policy trained with supervised learning on the replay buffer. We add all prior data to the replay buffer before online training.

Advantage Weighted Regression (AWR). Using the advantage weighted regression algorithm from [16], we add all prior data to the replay buffer before online training. We use the implementation provided by Peng et al. [16], with the key difference from our method being that AWR uses TD(λ) on the replay buffer for policy evaluation.

Maximum a Posteriori Policy Optimization (MPO). We evaluate the MPO algorithm presented by Abdolmaleki et al. [1]. Due to a public implementation being unavailable, we modify our algorithm to be as close to MPO as possible. In particular, we change the policy update in Advantage Weighted Actor Critic to be:

$$\theta_i \longleftarrow \arg\max_{\theta_i} \; \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi(a|s)} \left[\log \pi_{\theta_i}(a|s) \exp(\frac{1}{\beta} Q^{\pi_\beta}(s,a))\right]. \tag{15}$$

Note that in MPO, actions for the update are sampled from the policy and the Q-function is used instead of advantage for weights. We failed to see offline or online improvement with this implementation in most environments, so we omit this comparison in favor of ABM.

Advantage-Weighted Behavior Model (ABM). We evaluate ABM, the method developed in Siegel et al. [22]. As with MPO, we modify our method to implement ABM, as there is no public implementation of the method. ABM first trains an advantage model $\pi_{\theta_{\text{abm}}}(a|s)$:

$$\theta_{\text{abm}} = \arg\max_{\theta_i} \;\; \mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{t=1}^{|\tau|} \log \pi_{\theta_{\text{abm}}}(a_t|s_t) f(R(\tau_{t:N}) - \hat{V}(s))\right]. \tag{16}$$

where f is an increasing non-negative function, chosen to be $f = 1_+$. In place of an advantage computed by empirical returns $R(\tau_{t:N}) - \hat{V}(s)$ we use the advantage estimate computed per transition by the Q value $Q(s,a) - V(s)$. This is favorable for running ABM online, as computing $R(\tau_{t:N}) - \hat{V}(s)$ is similar to AWR, which shows slow online improvement. We then use the policy update:

$$\theta_i \longleftarrow \arg\max_{\theta_i} \;\; \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_{\text{abm}}(a|s)}\left[\log \pi_{\theta_i}(a|s) \exp\left(\frac{1}{\lambda}(Q^{\pi_i}(s,a) - V^{\pi_i}(s))\right)\right]. \tag{17}$$

Additionally, for this method, actions for the update are sampled from a behavior policy trained to match the replay buffer and the value function is computed as $V^\pi(s) = Q^\pi(s,a)$ s.t. $a \sim \pi$.

Demonstration Augmented Policy Gradient (DAPG). We directly utilize the code provided in [19] to compare against our method. Since DAPG is an on-policy method, we only provide the demonstration data to the DAPG code to bootstrap the initial policy from.

Bootstrapping Error Accumulation Reduction (BEAR). We utilize the implementation of BEAR provided in rlkit. We provide the demonstration and off-policy data to the method together. Since the original method only involved training offline, we modify the algorithm to include an online training phase. In general we found that the MMD constraint in the method was too conservative. As a result, in order to obtain the results displayed in our paper, we swept the MMD threshold value and chose the one with the best final performance after offline training with offline fine-tuning.