A Workflow for Offline Model-Free Robotic Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: Offline reinforcement learning (RL) enables learning control policies by utilizing only prior experience, without any online interaction. This can allow robots to acquire generalizable skills from large and diverse datasets, without costly online data collection. Despite recent algorithmic advances in offline RL, applying these methods to real-world problems has proven challenging. Although offline RL methods can learn from prior data, there is no clear and well-understood process for making various design choices, from model architecture to algorithm hyperparameters, without actually evaluating the learned policies online. In this paper, our aim is to develop a practical workflow for applying offline RL analogous to the relatively well-understood workflows for supervised learning problems. To this end, we devise a set of metrics and conditions that can be tracked over the course of offline training, and can inform the practitioner about how the algorithm and model architecture should be adjusted to improve final performance. Our workflow is derived from a conceptual understanding of the behavior of conservative offline RL algorithms and cross-validation in supervised learning. We demonstrate the efficacy of this workflow in producing effective policies without any online tuning, both in several simulated robotic learning scenarios and for three tasks on two distinct real robots, focusing on learning manipulation skills with raw image observations with sparse binary rewards.

Keywords: workflow, offline RL, no online tuning

1 Introduction

Offline reinforcement learning (RL) can in principle make it possible to convert existing large datasets of robotic experience into effective policies, without the need for costly or dangerous online interaction for each training run. While offline RL algorithms have improved significantly [1, 2, 3, 4, 5], applying such methods to real-world robotic control problems presents a number of major challenges. In standard online RL, any intermediate policy found during training is executed in the environment to collect more experience, which naturally allows for an evaluation of the policy performance. This ability to evaluate intermediate policies lets practitioners use "brute-force" to evaluate the effects of various design factors, such as model capacity and expressivity, the number of training steps, and so forth, and facilitates comparatively straightforward tuning. In contrast, offline RL methods do not have access to real-world on-policy rollouts for evaluating the learned policy. Thus, in order for these methods to be truly practical for real-world applications, we not only require effective algorithms, but also an effective *workflow*: a set of protocols and metrics that can be used to reliably and consistently adjust model capacity, regularization, etc in offline RL to obtain policies with good performance, without requiring real-world rollouts for tuning.

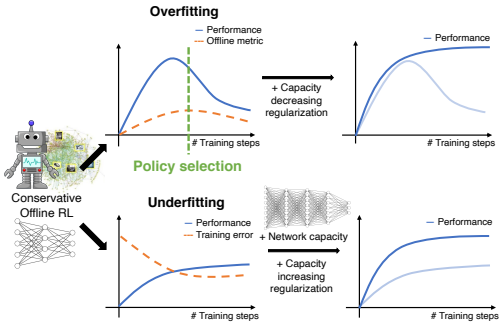


Figure 1: **Our proposed workflow** aims to detect overfitting and underfitting, and provides guidelines for addressing these issues via policy selection, regularization, and architecture design.

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

A number of prior works have studied *model selection* in offline RL by utilizing off-policy evaluation (OPE) methods [6] to estimate policy performance. These methods can be based either on model or value learning [7, 8, 9, 10] or importance sampling [6, 11, 12, 13]. However, developing reliable OPE methods is itself an open problem, and modern OPE methods themselves suffer from hyperparameter selection challenges (see Fu et al. [14] for an empirical study). Moreover, full off-policy evaluation is likely not necessary to simply tune algorithms for best performance – we do not need a precise estimate of *how* good our policy is, but rather a workflow that enables us to best improve it by adjusting various algorithm parameters.

In this paper, we devise a practically effective workflow for selecting regularizers, model architectures, and good policy checkpoints for offline RL methods in robotic learning settings. We focus on the class of conservative offline RL algorithms [15, 2] that regularize the Q-function. Our aim is not to focus on complete off-policy evaluation methods, but rather to adopt a strategy similar to the one in supervised learning. Analogously to how supervised learning practitioners can detect overfitting and underfitting by tracking training and validation losses, and then adjust hyperparameters based on these metrics, our workflow (see Figure 1 for a schematic) first defines and characterizes overfitting and underfitting, proposes a set of metrics and conditions that practitioners can track to determine if an offline RL exhibits overfitting or underfitting, and then utilizes these metrics to inform design decisions pertaining to improved neural network architectures, regularization and early stopping. This protocol is intended to act as a "user's manual" for a practitioner, enabling them to understand how a given offline RL run might fall short and then take steps to modify algorithm to improve the outcome of the next experiment, without requiring real-world evaluation rollouts.

The primary contribution of this paper is a simple yet effective workflow for robotic offline RL. We propose metrics and protocols to assist practitioners in selecting policy checkpoints, regularization parameters, and model architectures for conservative offline RL algorithms such as CQL [2]. We empirically verify our proposed workflow on simulated robotic manipulation problems as well as three real-world robotic manipulation problems on two different robots, with diverse objects, pixel observations, and sparse binary reward supervision. Our workflow improves the policy success rate to **70%** on two real-world Sawyer robot manipulation tasks which previously only failed at the task.

2 Related Work

Robotic RL with offline datasets. Learning-based methods have been applied to a number of robotics problems such as grasping objects [16, 17], in-hand object manipulation [18, 19, 20, 21], pouring fluids [22], door opening [23], and manipulating cloth [24]. While the majority of these works use standard online RL, a number of prior works have used also leveraged robotic datasets to train skills in addition to active environment rollouts. Kalashnikov et al. [16], Julian et al. [25], and Cabi et al. [26] use offline pre-training followed by a finetuning phase to improve the policy. Visual foresight [27] and related methods [28, 29, 30, 31] train a video-predictive dynamics model and use it for offline planning. Young et al. [32], Johns [33] lean skills in an offline manner, but only use imitation learning and not offline RL. Mandlekar et al. [34, 35] propose to learn hierarchical skills and combine them using offline imitation learning. Our work is complementary to these prior works in that our proposed workflow can be utilized by any robotic offline RL system.

Offline deep RL. While offline RL (also called batch RL) is a well-studied area [36, 37, 38, 15], several recent offline deep RL algorithms have been proposed. Algorithms for offline deep RL can be divided into three categories: those that constrain the policy to correct for distribution shift [39, 40, 41, 42, 43, 44], those that prevent overestimation in the value function via critic regularization [2, 45, 46] and those that train dynamics models and apply a reward penalty [47, 48]. These algorithms have been applied in robotics, for example when learning from unlabeled data [3], robotic manipulation [49, 16], goal-conditioned RL [4, 50] and multi-task RL problems [1]. Rather than developing a new offline RL method, our work develops criteria and workflow rules that simplify the application of these methods to new robotics tasks.

3 Preliminaries, Background, and Definitions

The goal in RL is to optimize the infinite horizon discounted return $R = \sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)$, where $r(s, a)$ represents the reward function evaluated at a state-action pair $(\mathbf{s}, \mathbf{a})$. We operate in the *offline* RL setting and are provided with a fixed dataset $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, r(\mathbf{s}, \mathbf{a}), \mathbf{s}')\}$, consisting of transition tuples obtained from rollouts under a behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$. Our goal is to obtain the best possible policy by only training on this fixed offline dataset $\mathcal{D}$, with no access to online rollouts.

Conservative Q-learning (CQL). In order to obtain a good policy from the offline dataset, $\mathcal{D}$, we use conservative Q-learning (CQL) [2], a state-of-the-art offline RL algorithm. We use the actor-critic formulation of CQL that trains a Q-function $Q_\theta(\mathbf{s}, \mathbf{a})$ and a separate policy $\pi_\phi(\mathbf{a}|\mathbf{s})$ that aims to maximize the expected Q-value $\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\phi}[Q_\theta(\mathbf{s}, \mathbf{a})]$ like other standard actor-critic deep RL methods [51, 52, 53]. In addition to the standard temporal difference (TD) error $\mathcal{L}_{\text{TD}}(\theta)$ (shown in blue below), CQL applies a regularizer $\mathcal{R}(\theta)$ (shown in red below) to prevent erroneous overestimation of Q-values. This aims to minimize the expected Q-value under $\mu(\mathbf{a}|\mathbf{s})$, $\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\cdot|\mathbf{s})}[Q_\theta(\mathbf{s}, \mathbf{a})]$. $\mu(\mathbf{a}|\mathbf{s})$ selectively picks out actions $\mathbf{a}$ with high Q-values $Q_\theta(\mathbf{s}, \mathbf{a})$ at a state $\mathbf{s}$, which are then minimized as shown in Equation 1 below:

$$\min_\theta \; \alpha \left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\cdot|\mathbf{s})}\left[Q_\theta(\mathbf{s}, \mathbf{a})\right] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[Q_\theta(\mathbf{s}, \mathbf{a})\right]\right) + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q_\theta(\mathbf{s}, \mathbf{a}) - \mathcal{B}^\pi \bar{Q}(\mathbf{s}, \mathbf{a})\right)^2\right], \quad (1)$$

where $\mathcal{B}^\pi \bar{Q}(\mathbf{s}, \mathbf{a})$ denotes the Bellman backup operator applied on the delayed target Q-function, $\bar{Q}$: $\mathcal{B}^\pi \bar{Q}(\mathbf{s}, \mathbf{a}) := r(\mathbf{s}, \mathbf{a}) + \gamma\mathbb{E}_{\mathbf{a}'\sim\pi(\mathbf{a}'|\mathbf{s}')}[\bar{Q}(\mathbf{s}', \mathbf{a}')]$. In practice, CQL computes $\mu(\mathbf{a}|\mathbf{s})$ using actions sampled from the policy $\pi_\phi(\mathbf{a}|\mathbf{s})$ that aims to maximize the learned Q-function. More discussion of CQL is in Appendix B. In this paper, we will utilize CQL as a base offline RL algorithm that our workflow would intend to tune. While the exact metrics and conditions in this paper are specific to CQL, a similar procedure can be used to derive a workflow for any conservative offline RL method.

Overfitting and underfitting in CQL. Conservative offline RL algorithms [2, 45] like CQL can be sensitive to design choices, including number of gradient steps for training [54, 55] and network capacity. These challenges are omnipresent throughout machine learning, even in supervised learning methods. However, supervised learning methods benefit from a simple and powerful workflow using training error and validation error (which serves as a proxy for test error) to characterize overfitting and underfitting. A practitioner can then make tuning choices based on these characterizations. To derive an analogous workflow for offline RL, we first ask: what do overfitting and underfitting actually mean for the case of conservative offline RL?

Quantity	Supervised Learning	Conservative Offline RL
Test error	Loss $\mathcal{L}$ evaluated on test data, $\mathcal{D}_{\text{test}}$	Performance of policy, $J(\pi)$
Train error	Loss $\mathcal{L}$ evaluated on train data, $\mathcal{D}_{\text{train}}$	Objective in Equations 2, 1
Overfitting	$\mathcal{L}(\mathcal{D}_{\text{train}})$ low, $\mathcal{L}(\mathcal{D}_{\text{val}})$ high, $\mathcal{D}_{\text{val}}$ is a validation set drawn i.i.d. as $\mathcal{D}_{\text{train}}$	Training objective in Equation 1 is extremely low, low value of $J(\pi)$
Underfitting	high value of train error $\mathcal{L}(\mathcal{D}_{\text{train}})$	Training objective in Equation 1 is extremely high, low value of $J(\pi)$

Table 1: Summary of train error, test error and our definitions of overfitting and underfitting in supervised learning and conservative offline RL methods. We will propose metrics to measure these phenomena in a purely offline manner and recommend how to tune the underlying method accordingly.

To define overfitting and underfitting generically for any conservative offline RL method, we consider an abstract optimization formulation for such methods [2]:

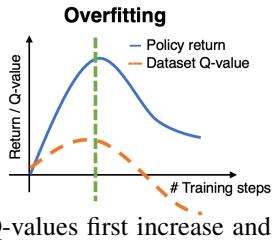
$$\pi^*(\mathbf{a}|\mathbf{s}) := \arg\max_\pi \; J_{\mathcal{D}}(\pi) - \alpha D(\pi, \pi_\beta) \qquad \text{(Conservative offline RL).} \quad (2)$$

$J_{\mathcal{D}}(\pi)$ denotes the average return of policy π in the empirical MDP induced by the transitions in the offline dataset $\mathcal{D}$, and $D(\pi, \pi_\beta)$ denotes a closeness constraint to the behavior policy, effectively applied by the offline RL method. For example, Equation 1 translates to utilizing $D_{\text{CQL}}(p, q) := \sum_{\mathbf{x}} p(\mathbf{x})(p(\mathbf{x})/q(\mathbf{x}) - 1)$ in Equation 2. The training loss in Equations 2 and 1 and the test loss is equal to the actual return $J(\pi)$ of the learned policy. Analogously to supervised learning, we can use the notion of train and test error to define overfitting and underfitting in offline RL, as shown in Table 1. However, note that the conditions summarized in Table 1 are not measurable completely offline. Precisely estimating if a run of an offline RL method overfits or underfits requires evaluating the learned policy via interaction with the real-world environment. In Section 4, our goal will be to devise offline metrics for characterizing overfitting that do not have this requirement. We will tailor our study specifically towards CQL. A similar procedure could be devised for other conservative offline RL methods, but we leave this for future work.

4 Detecting Overfitting and Underfitting in Conservative Offline RL

In standard supervised learning, we can determine if a method overfits or underfits by comparing the training loss to the same loss function evaluated on a held-out validation dataset, which serves as a "proxy" test dataset for tuning purposes without needing explicit access to the test dataset. In contrast, the return of the learned policy $J(\pi)$ in RL does not have a direct counterpart that can be computed offline. Our goal in this section is to identify feasible offline metrics and conditions that allow us to measure overfitting and underfitting in conservative offline RL, with a focus on CQL.

Detecting overfitting in CQL. Overfitting (Table 1) is defined as the scenario where the method attains a low value for the training loss (Equation 1), without actually giving rise to good policies with high $J(\pi)$. To detect overfitting for CQL, we analyze the time series of the estimated Q-values averaged over the dataset *samples* $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}') \in \mathcal{D}$ over the course of training with a large number of gradient steps. In scenarios characterized by overfitting, we should see that the this time series of the expected dataset Q-value exhibits a non-monotonic trend: the average Q-values first increase and then decrease as shown in the figure on the right. Additionally, we would see that training loss in Equation 1 eventually becomes very low. Why does this happen? Since CQL selectively penalizes the average Q-value under the distribution $\mu(\mathbf{a}|\mathbf{s})$ supported on actions with large Q-values, we would expect the Q-values on states from the dataset $\mathbf{s} \sim \mathcal{D}$ and the learned $\mathbf{a} \sim \pi(\cdot|\mathbf{s})$ to be small since the policy is trained to maximize the Q-function as well. This in turn would lead to an eventual reduction in the average Q-value on dataset actions, $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_\theta(\mathbf{s}, \mathbf{a})]$, which would be visible after sufficiently many steps of training, when values have propagated via Bellman backups in Equation 1 giving rise to the non-monotonic trend. If such a phenomenon is observed, this raises two questions:



What does a low average Q-value $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_\theta(\mathbf{s}, \mathbf{a})]$ ***imply about*** $J(\pi)$***?*** We show in Appendix A that, in principle, CQL training (Equation 1) should never learn Q-values smaller than the dataset Monte-Carlo return, and Q-values should generally increase during training. Intuitively, this is because the objective in Equation 1 aims to *maximize* the average dataset Q-value, and we formalize this intuition in Appendix A. Thus, a low Q-value on $(\mathbf{s}, \mathbf{a}) \in \mathcal{D}$ indicates that the Q-function predicts extremely small Q-values on actions sampled from $\mu(\mathbf{a}|\mathbf{s})$. Typically, this would mean that the learned policy $\pi(\mathbf{a}|\mathbf{s})$ will be close to the behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$ on states in the training dataset $\mathcal{D}$. Thus the resulting policy would have poor performance $J(\pi)$, at best matching the behavior policy and likely not improving upon it. A low average Q-value on the dataset can identify this scenario.

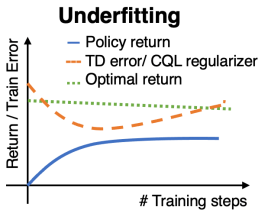
Which training checkpoint is likely to attain the best policy performance? One benefit of detecting overfitting in supervised learning is to select the best-performing checkpoint before overfitting becomes severe. Along similar lines, rather than explicitly quantifying what a "low" Q-value means, we can compare the average dataset Q-value across different checkpoints within a same run, and perform a relative comparison to guide policy selection when faced with overfitting. Since CQL aims to increase the average dataset Q-value (Equation 1), we would expect Q-values to initially increase, until learning starts to overfit and the average dataset Q-value starts decreasing. Thus, for deployment, we should select the latest checkpoint that corresponds to the peak in the dataset Q-value. A visual illustration of this idea is shown in the figure above, where the checkpoint marked by the green line is recommended to be chosen. **In summary**, **(a)** to detect overfitting we can track:

> **Metric 4.1** (Overfitting)**.** *A low average data Q-value* $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_\theta(\mathbf{s}, \mathbf{a})]$ *that decreases with more gradient steps on Equation 1 indicates that the offline RL algorithm is overfitting.*

and **(b)** further, given a run that exhibits overfitting, our principle for policy selection is given by:

> **Guideline 4.1** (Policy selection)**.** *If a run overfits (per Metric 4.1), select the checkpoint that attains the highest average dataset Q-value before overfitting for deployment.*

Detecting underfitting in CQL. Next, we turn to devising a procedure to detect underfitting. As summarized in Table 1, underfitting occurs when the RL algorithm is unable to minimize the training objective in Equation 1 effectively. Therefore, large values for the TD error, the CQL regularizer, or both imply underfitting. A large value for the CQL regularizer, $\mathcal{R}(\theta)$, indicates an overestimation of Q-values relative to their true value [2] and thus, unlike the overfitting regime, we would *not* expect the average learned Q-value to decrease with more training. Thus, one approach to predict underfitting is to track both the TD error, $\mathcal{L}_{\text{TD}}(\theta)$, and the CQL regularizer, $\mathcal{R}(\theta)$, and check if the value of even one of these quantities is large. More discussion is provided in Appendix A.



How do we determine if the TD error and the CQL regularizer are "large"? In order to determine if the error of a particular run is large, we can rerun the base CQL algorithm but with models of higher capacity. This can be achieved in many ways, two of which we will discuss in Section 5.

For each model, we record the corresponding training errors and check if the training TD error and regularizer value are reduced as capacity is increased. If increasing capacity leads to a reduction in the loss without exhibiting the overfitting signs described previously, then we are in an underfitting regime. Another approach to answer the question is to utilize the value of the TD error ($\mathcal{L}_{\text{TD}}(\theta)$) and the task horizon ($1/(1-\gamma)$) to estimate the overall error in the learned Q-values against the actual Q-value, which is equal to $\mathcal{L}_{\text{TD}}(\theta)/(1-\gamma)$ [56] (see Appendix A for details). If this overall error spans the range of allowed Q-values on the task – which could be inferred based on the structure of the reward function in the task of interest–then we can say that the algorithm is underfitting.

Metric 4.2 (Underfitting)**.** *Compute the values of the training TD error, $\mathcal{L}_{\text{TD}}(\theta)$ and CQL regularizer, $\mathcal{R}(\theta)$ for the current run and another identical run with increased model capacity. If the training errors reduce with increasing model capacity, the original run was underfitting.*

5 Addressing Overfitting and Underfitting in Conservative Offline RL

The typical workflow for supervised learning not only identifies overfitting and underfitting, but also guides the practitioner how to adjust their run so as to alleviate the phenomenon, which is expected to improve the downstream task performance during testing. For example, overfitting can be addressed by applying regularization (e.g., weight decay, dropout) and underfitting can be addressed by increasing the model capacity. Can we devise similar guidelines to address overfitting and underfitting with conservative offline RL methods like CQL? In this section, we provide a practitioner with some ways to adjust regularization and model capacity to alleviate these phenomena.

Capacity-decreasing regularization for overfitting. As we observed in Section 4, the mechanism behind extremely low Q-values on the dataset is that CQL training minimizes Q-values on actions sampled from $\mu(\mathbf{a}|\mathbf{s})$. One simple approach to prevent over-minimization of these values is to enforce that the internal representations of the learned Q-function should match a pre-specified representation for all state-action tuples. One way of doing so is to utilize regularization techniques from supervised learning such as a variational information bottleneck (VIB) [57, 58] regularizer on the learned representations, $\phi(\mathbf{s})$. Formally, let $(\mathbf{s}, \mathbf{a})$ denote a state-action pair. Instead of predicting a deterministic $\phi(\mathbf{s}) \in \mathbb{R}^d$ (Figure 9), we modify the Q-network to predict two distinct vectors, $\phi_m(\mathbf{s}) \in \mathbb{R}^d$ and $\phi_\Sigma(\mathbf{s}) \in \mathbb{R}^d$, and sample $\phi(\mathbf{s})$ randomly from a Gaussian centered at ϕ_m with covariance ϕ_Σ, i.e., $\phi(\mathbf{s}) \sim \mathcal{N}(\phi_m(\mathbf{s}), \text{diag}(\phi_\Sigma(\mathbf{s}))$. Information bottleneck then regularizes $\mathcal{N}(\phi_m(\mathbf{s}), \text{diag}(\phi_\Sigma(\mathbf{s}))$ to be close to a prior distribution, $\mathcal{N}(0, \mathbb{I})$:

$$\min_\theta \;\; \mathcal{L}_{\text{CQL}}(\theta) + \beta \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \left[\text{D}_{\text{KL}} \left(\mathcal{N}(\phi_m(\mathbf{s}), \text{diag}(\phi_\Sigma(\mathbf{s}))) \;\middle|\middle|\; \mathcal{N}(0, \mathbb{I}) \right) \right] \qquad \text{(VIB regularizer)}, \tag{3}$$

Guideline 5.1. *To address overfitting, we recommend using some form of capacity-decreasing regularization on the Q-function, such as the VIB regularizer shown in Equation 3.*

Capacity-increasing techniques for underfitting. To address underfitting, we need to increase model capacity to improve optimization of the training objective. Analogous to supervised learning, model capacity can be increased by using more expressive neural nets (e.g., ResNets [59], transformers [60]) for representing the learned policy. We use ResNets in our experiments (Figure 9). However, the RL setting presents an additional challenge with capacity: while larger models *in principle* have more capacity, recent work [61, 54, 55] has shown that utilizing larger networks to represent Q-functions does not always improve its capacity in practice, because TD-based RL methods introduce an "implicit regularization" effect that can result in aliased (i.e., similar) internal representations for different state-action inputs, even for very large neural networks that can express the true Q-function effectively. To address this issue, these works apply a "capacity-increasing" regularizer to Q-function training. For instance, we can use the DR3 regularizer [55], which penalizes the dot product of $\phi(\mathbf{s})$ and $\phi(\mathbf{s}')$ for a transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}') \in \mathcal{D}$, and hence reduces aliasing. This objective is given by:

$$\min_\theta \;\; \mathcal{L}_{\text{CQL}}(\theta) + \beta \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}} \left[\left| \phi(\mathbf{s})^\top \phi(\mathbf{s}') \right| \right] \qquad \text{(DR3 regularizer [55])}, \tag{4}$$

Guideline 5.2. *To address underfitting, we recommend using some capacity-increasing regularization on the Q-function and the policy either in conjunction or separately. Examples: **(1)** bigger policy networks (e.g., ResNets), **(2)** DR3 regularizer on the Q-network.*

6 Evaluation of Our Workflow Metrics and Protocols in Simulation

Next, we empirically validate the workflow proposed in Sections 4 and 5 on a suite of simulated robotic manipulation domains that mimic real-robot scenarios, operating directly from image observations with sparse binary rewards. We will examine how applying the workflow in Section 4 to detect overfitting or underfitting and then utilizing the strategies in Section 5 affects the performance of offline RL methods. An improved performance would indicate the efficacy of our workflow in guiding a practitioner in making successful design decisions without any online tuning.

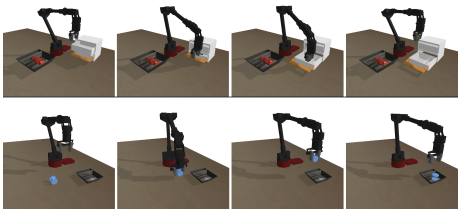


Figure 2: **Simulated domains [3].** Examples of trajectories for the *(Top)* pick and place task and *(Bottom)* grasping from drawer task.

Experimental setup. We use the environments from Singh et al. [3] to design offline RL tasks and datasets that we use for our empirical analysis. We consider two tasks: **(1)** a pick and place task and **(2)** a grasping object from a drawer task. Examples of trajectories in both of these simulated domains are shown in Figure 2 and are detailed in Appendix C. Briefly, the ***pick and place*** task consists of a 6-DoF WidowX robot in front of a tray with an object. The goal is to put the object inside the tray. A non-zero reward of +1 is provided only when the object has been placed in the box. The offline dataset for this task consists of trajectories that grasp an object with a 35% success and other trajectories that place an object with a 40% success. Our second task is a ***grasping from drawer task*** where the WidowX robot is placed in front of a drawer and multiple objects. The robot can open or close the drawer, grasp objects from inside the drawer or on the table, and place them anywhere in the scene. The goal is to close the top drawer, then open the bottom drawer and take the object out. Only if the object has been taken out, a reward of +1 is obtained. The offline dataset consists of trajectories with a 30-40% success rate for opening and closing a drawer and other trajectories with only 40% placing success. **For CQL training**, we follow prior work [3] and use a value of $\alpha = 1.0$ in Equation 1 for all our tasks. We utilize a standard convolutional architecture for the Q-function (Figure 10) and a similar architecture for the policy except that it does take in an action as input. More details regarding our empirical setup are provided in Appendix C.

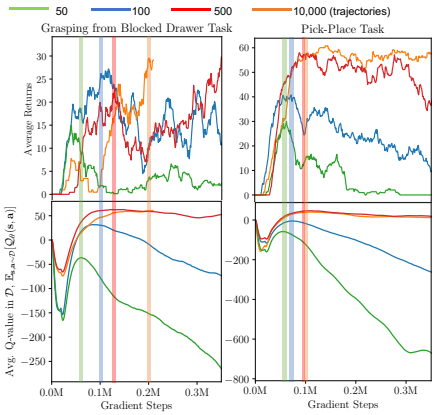


Figure 3: **Policy performance (Top) and average dataset Q-values of CQL (bottom) with varying number of trajectories.** Vertical bands indicate regions around the peak in average Q-value and observe that these regions correspond to policies with good actual performance.

Scenario #1: Variable amount of training data. Our first scenario consists of the simulated tasks discussed above with a variable number of trajectories in the training data (50, 100, 500, 10000). We run CQL and track metrics 4.1 and 4.2 in each case. Observe in Figure 3 (bottom) that with fewer trajectories, the average dataset Q-value $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q_\theta(\mathbf{s},\mathbf{a})]$ first rises, and then drops. This matches the description of overfitting in Section 4. Observe in Figure 4 (left) that, at the same time, the value of the CQL regularizer is very low, which is not consistent with what we expect of underfitting. Thus, we can conclude that these conditions exhibit overfitting, especially with 50 and 100 trajectories. The vertical dashed lines indicate the checkpoints that would be selected for evaluation per Guideline 4.1. We further visualize the performance of the chosen checkpoints against the actual return of each intermediate policy in Figure 3 (top). Note that this value is obtained by

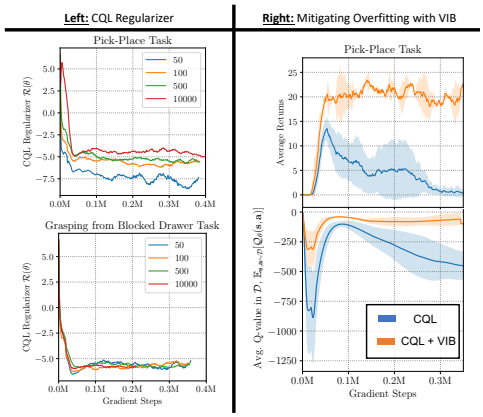


Figure 4: **Left:** CQL regularizer attains low values, especially with 50 and 100 trajectories in the pick and place task, **Right:** Using VIB mitigates overfitting, giving rise to a stable trend in Q-values and better performance which does not degrade with more training steps.

rolling out the learned policy, and would not be available in a realistic offline RL setting, but is provided only for analysis. Selecting the checkpoint based on Guideline 4.1 leads us to select a model with close to the peak performance over the training process, validating the efficacy of Guideline 4.1.

Since we detected overfitting by following our proposed workflow, we should be able to improve performance by decreasing capacity, either by using a smaller model or adding the information bottleneck regularizer. We introduce the regularizer in the setting with 100 trajectories. As shown in Figure 4 (right), applying this regularizer not only alleviates the drop in Q-values after many training steps, but allows us to pick later checkpoints in training, all of which perform equally well, and better than the base CQL algorithm on both the tasks. This validates that overfitting, as detected via our workflow, can be effectively mitigated by decreasing capacity, in this case by using VIB.

Scenario #2: Multiple training objects. Our second test scenario consists of the pick and place task, modified to include a variable number of object types (1, 5, 10, 20, 35). Handling more objects requires higher capacity, since each object has a different shape and appearance. In each case, CQL is provided with 5000 trajectories. Following our workflow from Section 4, we first compute the average dataset Q-value and the training TD error. We observe in Figure 5 that, unlike in Scenario #1, Q-values do not generally decrease when trained for many steps, suggesting that the Q-function is likely not overfitting. To check for underfitting, we visualize the training TD error and find that, with 10, 20 and 35 objects, TD error magnitudes are in the range of [1.0, 2.0], which suggests a overall Q-value error of [30.0, 60.0] since the task horizon is 30. On an absolute scale, this error magnitude is large: since the rewards are 0/1, the range of difference between actual Q-values for any two policies is at most 30, which suggests that the error magnitude in the runs in Figure 5 are high. Hence, we conclude that this scenario generally exhibits underfitting with more objects. Indeed this trend is reflected in the policy performance that we plot for analysis in Figure 5: note that the policy return decreases with an increased number of objects, and the policy performance initially increases and saturates at a suboptimal value in the settings that exhibit underfitting, consistent with Section 4.

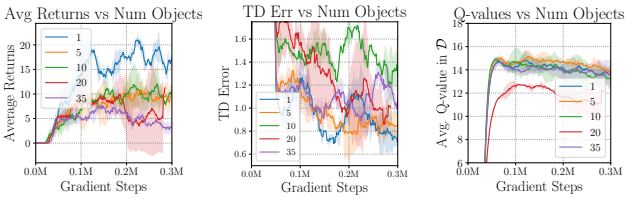
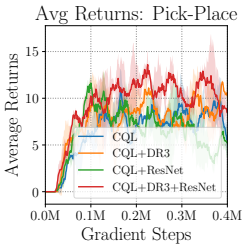


Figure 5: Performance (left), TD error (middle) and average dataset Q-values (right) for the pick and place task with a variable number of objects. Note that while the learned Q-values increase and generally stabilize, the TD error values in scenarios with more than 10 objects are large (1.0-2.0). Correspondingly, the performance generally decreases as the number of objects increases.

To address underfitting detected by our workflow in the multi-object case, we apply the proposed capacity-increasing measures to the 35-object task. We use a more expressive ResNet architecture for the policy and the DR3 regularizer for the Q-function together. Observe in the figure on the right that this combination (shown in red) improves policy performance in this setting (compared to green), which validates our workflow protocol for addressing underfitting. Metrics for this run are provided in Appendix D.1.



7 Tuning CQL for Real-World Robotic Manipulation

Having evaluated the efficacy of our proposed workflow on simulated domains, we now utilize our workflow to tune CQL for real-world robotic manipulation. We test in two setups that both require the robot to learn from sparse binary rewards and image observations. The settings differ in robot platform, task specification, and dataset size. Additional experiments are provided at the following anonymous URL: `https://sites.google.com/view/offline-rl-workflow`.

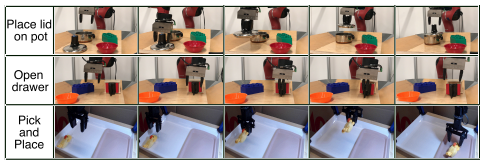


Figure 6: **Real-world tasks**. Examples of successful trajectories after tuning using our workflow from Sections 4 & 5: *(Top)* Sawyer lid on pot, *(Middle)* Sawyer drawer opening, *(Bottom)* WidowX pick and place task.

Sawyer manipulation tasks [50]. First, we train a Sawyer robot in a tabletop setting to perform two tasks: **(1)** placing the lid onto a pot and **(2)** opening a drawer. The robot must perform these tasks in the presence of visual distractor objects, as shown in Figure 6. We directly use the dataset of 100 trajectories for each task collected by Khazatsky et al.

[50] for our experiments so as to mimic the real-world use case of leveraging existing data with offline RL. We use four-dimensional actions with 3D end-effector velocity control in xyz-space and 1D gripper open/close action. More details regarding the setup are provided in Appendix C.

We run default CQL on these tasks and track the average Q-value, TD error, and CQL regularizer value. As shown in Figure 7, the average Q-value does not decrease over training, and the TD error (and CQL regularizer shown in Appendix D.2) is large. Per our discussion in Section 4, this indicates underfitting. Following our guidelines from Section 5, we utilize a more expressive ResNet policy (Figure 9), which increases the number of total convolutional layers from 3 to 9. We observe that this reduces the values of both the TD error Figure 7 and CQL regularizer (Appendix D.2) on both tasks. We then evaluate the learned policy over 12 trials conducted with different sets of distractor objects, including ones that are unseen during training. While the policy trained using base CQL is unable to successfully complete either task even once attaining a score of 0/12 on both tasks, the run that uses ResNet attains a significantly better success rate of **9/12** on the put lid on pot task and **8/12** on the drawer opening task, equal to **70.8%** success rate on average.

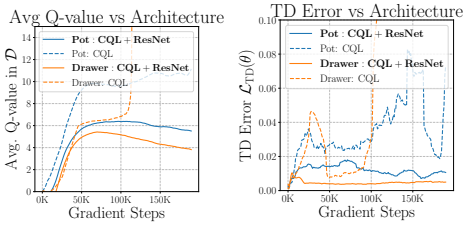


Figure 7: Average dataset Q-value and TD error on Sawyer tasks as model capacity increases. Q-values increase over training with lower capacity ruling out overfitting and increasing model capacity leads to a reduction in TD error.

WidowX pick and place task. In our second setting, we tune CQL on a pick and place task with a WidowX 250 robotic arm, shown in Figure 6. The dataset consists of 200 trajectories collected by running a noisy scripted policy (Appendix C) with 35% success. We run CQL on this task and track the average Q-values, which we find initially increase and then decrease (Figure 8), indicating overfitting. We then evaluate our policy selection scheme, which in this case suggests deploying checkpoint 50, the latest checkpoint with high Q-values. To see if this checkpoint is effective, we evaluate the performance of a few other policy checkpoints (for analysis only) and plot this performance trend in Figure 8 as a dashed line. Observe that indeed the checkpoint found by our approach attains the highest success rate (**7/9**) compared to other checkpoints, which only succeed $\leq$ 4/9 times. This indicates the efficacy of our proposed metrics in identifying overfitting and our policy selection guideline.

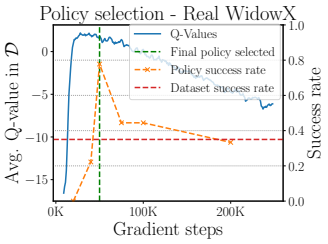


Figure 8: *Pick and place task on the real WidowX.* Q-values learned by CQL, success rates for selected policy checkpoints (out of 9 trials), and our policy selection scheme (green). Note the effectiveness of our policy selection scheme.

8 Discussion

While offline RL algorithms have improved significantly over the past years, applying these methods to real-world robotic domains is still challenging due to little guidance on selecting policies, adding regularization, or modifying model capacity. In this paper, we devise a *workflow* for conservative offline RL algorithms such as CQL, which consists of a set of metrics and conditions that can be tracked by a practitioner over the course of offline training as well as a set of recommendations to addresses the resulting overfitting and underfitting challenges. We use our workflow to tune CQL on a number of robotic manipulation problems with diverse robot types, multiple objects, and long horizons, all while learning without online interaction from image observations and sparse rewards. Both in simulation and the real world, we observe strong performance benefits.

While our proposed workflow is an initial step towards practical robotic offline RL, our work has a number of limitations. The validation of our workflow is largely empirical in nature. We believe this is unavoidable, since a workflow is a set of guidelines and recommendations, rather than a rigid algorithm. Regardless of how theoretically justified it is, in the end, its value is determined by its ability to produce good results in practice. Our validation consists of a series of case studies, and although the results (as with any empirical finding) are specific to the used domains, we believe the breadth of tasks considered, which consist of two different real-world robots and multiple simulated tasks, indicates the broad applicability of our approach. Our workflow is also specific to conservative offline RL methods, and particularly CQL. Extending our technique to devise similar workflows for other algorithms is an exciting direction for future work. Deriving theoretical guarantees regarding workflows of this type are also a very important direction for future research.

References

[1] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

[2] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[3] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *arXiv preprint arXiv:2010.14500*, 2020.

[4] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn, and S. Levine. Actionable models: Unsupervised offline reinforcement learning of robotic skills. *arXiv preprint arXiv:2104.07749*, 2021.

[5] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[6] D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[7] I. Kostrikov and O. Nachum. Statistical bootstrapping for uncertainty estimation in off-policy evaluation. *arXiv preprint arXiv:2007.13609*, 2020.

[8] C. Paduraru. *Off-policy evaluation in Markov decision processes*. PhD thesis, Ph. D. Dissertation. McGill University, 2012.

[9] T. L. Paine, C. Paduraru, A. Michi, C. Gulcehre, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

[10] O. Nachum and B. Dai. Reinforcement learning via fenchel-rockafellar duality. *arXiv preprint arXiv:2001.01866*, 2020.

[11] P. Thomas, G. Theocharous, and M. Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, pages 2380–2388, 2015.

[12] P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High-confidence off-policy evaluation. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[13] N. Jiang and L. Li. Doubly robust off-policy value evaluation for reinforcement learning. *arXiv preprint arXiv:1511.03722*, 2015.

[14] J. Fu, M. Norouzi, O. Nachum, G. Tucker, ziyu wang, A. Novikov, M. Yang, M. R. Zhang, Y. Chen, A. Kumar, C. Paduraru, S. Levine, and T. Paine. Benchmarks for deep off-policy evaluation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=kWSeGEeHvF8.

[15] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[16] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673, 2018.

[17] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. 2018.

[18] OpenAI. Learning dexterous in-hand manipulation. In *arXiv preprint arXiv:1808.00177*, 2018.

[19] H. van Hoof, T. Hermans, G. Neumann, and J. Peters. Learning robot in-hand manipulation with tactile features. 2015.

[20] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *RSS*, 2018.

[21] V. Kumar, A. Gupta, E. Todorov, and S. Levine. Learning dexterous manipulation policies from experience and imitation. *CoRR*, abs/1611.05095, 2016.

[22] C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[23] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *IROS*, 2017.

[24] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

[25] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman. Efficient adaptation for end-to-end vision-based robotic manipulation. *arXiv arXiv:2004.10190*, 2020.

[26] S. Cabi, S. G. Colmenarejo, A. Novikov, K. Konyushkova, S. Reed, R. Jeong, K. Żołna, Y. Aytar, D. Budden, M. Vecerik, et al. A framework for data-driven robotics. *arXiv preprint arXiv:1909.12200*, 2019.

[27] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[28] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[29] A. Xie, F. Ebert, S. Levine, and C. Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. *Robotics: Science and Systems (RSS)*, 2019.

[30] Y. Hristov, A. Lascarides, and S. Ramamoorthy. Interpretable latent spaces for learning from demonstration. *arXiv preprint arXiv:1807.06583*, 2018.

[31] S. Tian, S. Nair, F. Ebert, S. Dasari, B. Eysenbach, C. Finn, and S. Levine. Model-based visual planning with self-supervised functional distances. *arXiv preprint arXiv:2012.15373*, 2020.

[32] S. Young, D. Gandhi, S. Tulsiani, A. Gupta, P. Abbeel, and L. Pinto. Visual imitation made easy. *arXiv e-prints*, pages arXiv–2008, 2020.

[33] E. Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. *arXiv preprint arXiv:2105.06411*, 2021.

[34] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, L. Fei-Fei, A. Garg, and D. Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE, 2020.

[35] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations, 2020.

[36] S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

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

[38] R. Munos. Error bounds for approximate value iteration. In *AAAI Conference on Artificial intelligence (AAAI)*, pages 1006–1011. AAAI Press, 2005.

[39] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

[40] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

[41] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[42] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[43] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[44] A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[45] I. Kostrikov, J. Tompson, R. Fergus, and O. Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

[46] R. Fakoor, J. Mueller, P. Chaudhari, and A. J. Smola. Continuous doubly constrained batch reinforcement learning. *arXiv preprint arXiv:2102.09225*, 2021.

[47] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[48] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

[49] R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn. Offline reinforcement learning from images with latent space models. *Learning for Decision Making and Control (L4DC)*, 2021.

[50] A. Khazatsky, A. Nair, D. Jing, and S. Levine. What can i do here? learning new skills by imagining visual affordances. *arXiv preprint arXiv:2106.00671*, 2021.

[51] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[52] S. Fujimoto, H. Van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[53] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[54] A. Kumar, R. Agarwal, D. Ghosh, and S. Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=O9bnihsFfXU.

[55] Anonymous. Value-based deep reinforcement learning requires explicit regularization. In *Under Review*, 2021.

[56] R. Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 560–567. AAAI Press, 2003. ISBN 1577351894.

[57] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

[58] A. Achille and S. Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018.

[59] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[61] D. Ghosh and M. G. Bellemare. Representations for stable off-policy reinforcement learning. *arXiv preprint arXiv:2007.05520*, 2020.