

IMPLICIT UNDER-PARAMETERIZATION INHIBITS DATA-EFFICIENT DEEP REINFORCEMENT LEARNING

Aviral Kumar^{*1,2}

¹UC Berkeley

Rishabh Agarwal^{* 2,3}

²Google Research

Dibya Ghosh¹

Sergey Levine^{1,2}

³MILA, Université de Montréal

ABSTRACT

We identify an implicit under-parameterization phenomenon in value-based deep RL methods that use bootstrapping: when value functions, approximated using deep neural networks, are trained with gradient descent using iterated regression onto target values generated by previous instances of the value network, more gradient updates decrease the expressivity of the current value network. We characterize this loss of expressivity in terms of a drop in the rank of the learned value network features, and show that this corresponds to a drop in performance. We demonstrate this phenomenon on widely studies domains, including Atari and Gym benchmarks, in both offline and online RL settings. We formally analyze this phenomenon and show that it results from a pathological interaction between bootstrapping and gradient-based optimization. We further show that mitigating implicit under-parameterization by controlling rank collapse improves performance.

1 INTRODUCTION

Many effective deep reinforcement learning (RL) algorithms estimate value functions using bootstrapping, that is, by sequentially fitting value functions to target value estimates generated from the value function learned in the previous iteration. Despite high-profile achievements (Silver et al., 2017), these algorithms are highly unreliable (Henderson et al., 2017) due to poorly understood optimization issues. Although a number of hypotheses have been proposed to explain these issues, resulting in several effective fixes (Achiam et al., 2019; Bengio et al., 2020; Fu et al., 2019; Igl et al., 2020; Liu et al., 2018; Kumar et al., 2020a; Du et al., 2019), a complete understanding remains elusive.

We identify an “implicit under-parameterization” phenomenon (Figure 1) that emerges when value networks are trained using gradient descent combined with bootstrapping. This phenomenon manifests as an excessive aliasing of features learned by the value network across states, which is exacerbated with more gradient updates. While the supervised deep learning literature suggests that some feature aliasing is desirable for generalization (*e.g.*, Gunasekar et al., 2017; Arora et al., 2019), implicit under-parameterization exhibits more pronounced aliasing than in supervised learning. This over-aliasing causes an otherwise expressive value network to *implicitly* behave as an *under-parameterized* network, often resulting in poor performance.

Implicit under-parameterization becomes aggravated when the rate of data re-use is increased, restricting the sample efficiency of deep RL methods. In online RL, increasing the number of gradient steps between data collection steps for *data-efficient* RL (Fu et al., 2019; Fedus et al., 2020b) causes the problem to emerge more frequently. In the extreme case when no additional data is collected, referred to as *offline* RL (Lange et al., 2012; Agarwal et al., 2020; Levine et al., 2020), implicit under-parameterization manifests consistently, limiting the viability of offline methods.

We demonstrate the existence of implicit under-parameterization in common value-based deep RL methods, including Q-learning (Mnih et al., 2015; Hessel et al., 2018) and actor-critic algorithms (Haarnoja et al., 2018), as well as neural fitted-Q iteration (Riedmiller, 2005; Ernst et al., 2005). To isolate the issue, we study the effective rank of the features in the penultimate layer of the value network (Section 3). We observe that after an initial learning period, the rank of the features drops steeply. As the rank decreases, the features are less suitable for fitting subsequent target values, resulting in a sharp decrease in performance (Section 3.3).

^{*}Equal contribution. Correspondence to Aviral Kumar (aviralk@berkeley.edu) and Rishabh Agarwal (rishabhagarwal@google.com).

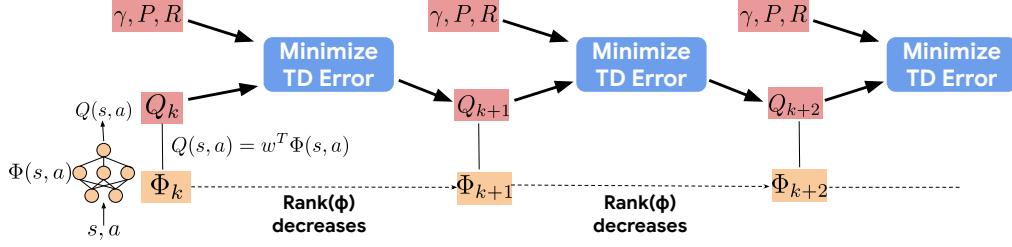


Figure 1: **Implicit under-parameterization.** Schematic diagram depicting the emergence of an *effective rank* collapse in deep Q-learning. Minimizing TD errors using gradient descent with deep neural network Q-function leads to a collapse in the effective rank of the learned features Φ , which is exacerbated with further training.

To better understand the emergence of implicit under-parameterization, we formally study the dynamics of Q-learning under two distinct models of neural net behavior (Section 4): kernel regression (Jacot et al., 2018; Mobahi et al., 2020) and deep linear networks (Arora et al., 2018). We corroborate the existence of this phenomenon in both models, and show that implicit under-parameterization stems from a pathological interaction between bootstrapping and the implicit regularization of gradient descent. Since value networks are trained to regress towards targets generated by a previous version of the same model, this leads to a sequence of value networks of potentially decreasing expressivity, which can result in degenerate behavior and a drop in performance.

The main contribution of this work is the identification of implicit under-parameterization in deep RL methods that use bootstrapping. Empirically, we demonstrate a collapse in the rank of the learned features during training, and show that it typically corresponds to a drop in performance in the Atari (Bellemare et al., 2013) and continuous control Gym (Brockman et al., 2016) benchmarks in both the offline and data-efficient online RL settings. We additionally analyze the causes of this phenomenon theoretically. We then show that mitigating this phenomenon via a simple penalty on the singular values of the learned features improves performance of value-based RL methods in the offline setting on Atari.

2 PRELIMINARIES

The goal in RL is to maximize long-term discounted reward in a Markov decision process (MDP), defined as a tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$ (Puterman, 1994), with state space \mathcal{S} , action space \mathcal{A} , a reward function $R(\mathbf{s}, \mathbf{a})$, transition dynamics $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ and a discount factor $\gamma \in [0, 1]$. The Q-function $Q^\pi(\mathbf{s}, \mathbf{a})$ for a policy $\pi(\mathbf{a}|\mathbf{s})$, is the expected long-term discounted reward obtained by executing action \mathbf{a} at state \mathbf{s} and following $\pi(\mathbf{a}|\mathbf{s})$ thereafter, $Q^\pi(\mathbf{s}, \mathbf{a}) := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}_t, \mathbf{a}_t)]$. $Q^\pi(\mathbf{s}, \mathbf{a})$ is the fixed point of the Bellman operator \mathcal{T}^π , $\forall \mathbf{s}, \mathbf{a}: \mathcal{T}^\pi Q(\mathbf{s}, \mathbf{a}) := R(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim P(\cdot|\mathbf{s}, \mathbf{a}), \mathbf{a}' \sim \pi(\cdot|\mathbf{s}')} [Q(\mathbf{s}', \mathbf{a}')]$, which can be written in vector form as: $\mathbf{Q}^\pi = \mathbf{R} + \gamma P^\pi \mathbf{Q}^\pi$. The optimal Q-function, $Q^*(\mathbf{s}, \mathbf{a})$, is the fixed point of the Bellman optimality operator $\mathcal{T}: \mathcal{T}Q(\mathbf{s}, \mathbf{a}) := R(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim P(\cdot|\mathbf{s}, \mathbf{a})} [\max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')]$.

Practical Q-learning methods (e.g., Mnih et al., 2015; Hessel et al., 2018; Haarnoja et al., 2018) convert the Bellman equation into an bootstrapping-based objective for training a Q-network, Q_θ , via gradient descent. This objective, known as mean-squared temporal difference (TD) error, is given by: $L(\theta) = \sum_{\mathbf{s}, \mathbf{a}} (R(\mathbf{s}, \mathbf{a}) + \gamma \bar{Q}_\theta(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a}))^2$, where \bar{Q}_θ is a delayed copy of the Q-function, typically referred to as the *target network*. These methods train Q-networks via gradient descent and slowly update the target network via Polyak averaging on its parameters. We refer the output of the penultimate layer of the deep Q-network as the learned *feature matrix* Φ , such that $Q(\mathbf{s}, \mathbf{a}) = \mathbf{w}^T \Phi(\mathbf{s}, \mathbf{a})$, where $\mathbf{w} \in \mathbb{R}^d$ and $\Phi \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times d}$.

For simplicity of analysis, we abstract deep Q-learning methods into a generic fitted Q-iteration (**FQI**) framework (Ernst et al., 2005). We refer to FQI with neural nets as neural FQI (Riedmiller, 2005). In the k -th *fitting* iteration, FQI trains the Q-function, Q_k , to match the target values, $\mathbf{y}_k = \mathbf{R} + \gamma P^\pi Q_{k-1}$ generated using previous Q-function, Q_{k-1} (Algorithm 1). Practical methods can be instantiated as variants of FQI, with different target update styles, different optimizers, etc.

Algorithm 1 Fitted Q-Iteration (FQI)

```

1: Initialize Q-network  $Q_\theta$ , buffer  $\mu$ .
2: for fitting iteration  $k$  in  $\{1, \dots, N\}$  do
3:   Compute  $Q_\theta(\mathbf{s}, \mathbf{a})$  and target values
       $y_k(\mathbf{s}, \mathbf{a}) = r + \gamma \max_{\mathbf{a}'} Q_{k-1}(\mathbf{s}', \mathbf{a}')$ 
      on  $\{(\mathbf{s}, \mathbf{a})\} \sim \mu$  for training
4:   Minimize TD error for  $Q_\theta$  via  $t = 1, \dots, T$  gradient descent updates,
       $\min_{\theta} (Q_\theta(\mathbf{s}, \mathbf{a}) - y_k)^2$ 
5: end for

```

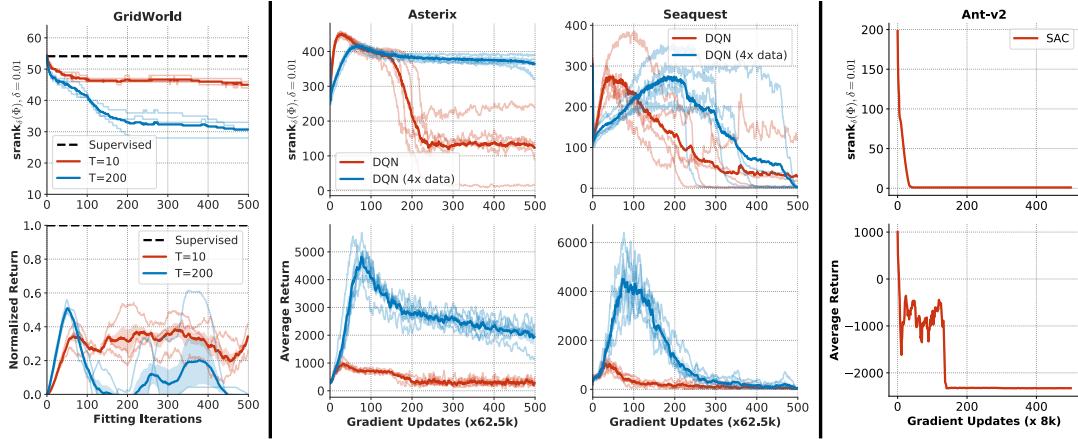


Figure 2: **Data-efficient offline RL.** $\text{srank}_\delta(\Phi)$ and performance of neural FQI on gridworld, DQN on Atari and SAC on Gym environments in the offline RL setting. Note that low rank (top row) generally corresponds to worse policy performance (bottom row). Rank collapse is worse with more gradient steps per fitting iteration ($T = 10$ vs. 200 on gridworld). Even when a larger, high coverage dataset is used, marked as DQN (4x data), rank collapse occurs (for Asterix also see Figure A.2 for a complete figure with a larger number of gradient updates).

3 IMPLICIT UNDER-PARAMETERIZATION IN DEEP Q-LEARNING

In this section, we empirically demonstrate the existence of implicit under-parameterization in deep RL methods that use bootstrapping. We characterize implicit under-parameterization in terms of the *effective rank* (Yang et al., 2019) of the features learned by a Q-network. The effective rank of the feature matrix Φ , for a threshold δ (we choose $\delta = 0.01$), denoted as $\text{srank}_\delta(\Phi)$, is given by $\text{srank}_\delta(\Phi) = \min \left\{ k : \frac{\sum_{i=1}^k \sigma_i(\Phi)}{\sum_{i=1}^d \sigma_i(\Phi)} \geq 1 - \delta \right\}$, where $\{\sigma_i(\Phi)\}$ are the singular values of Φ in decreasing order, i.e., $\sigma_1 \geq \dots \geq \sigma_d \geq 0$. Intuitively, this quantity represents the number of “effective” unique components of the feature matrix Φ that form the basis for linearly approximating the Q-values. When the network maps different states to orthogonal feature vectors, such as when Φ is an identity matrix, then $\text{srank}_\delta(\Phi)$ has high values close to d . When the network “aliases” state-action pairs by mapping them to a smaller subspace, Φ has only a few active singular directions, and $\text{srank}_\delta(\Phi)$ takes on a small value. Based on this insight we can define implicit under-parameterization as:

Definition 1. Implicit under-parameterization refers to a reduction in the effective rank of the features, $\text{srank}_\delta(\Phi)$, that occurs implicitly as a by-product of learning deep neural network Q-functions.

While rank decrease also occurs in supervised learning, it is usually beneficial for obtaining generalizable solutions (Gunasekar et al., 2017; Arora et al., 2019). However, we will show that in deep Q-learning, an interaction between bootstrapping and gradient descent can lead to more aggressive rank reduction (or rank collapse), which can hurt performance.

Experimental setup. To study implicit under-parameterization empirically, we compute $\text{srank}_\delta(\Phi)$ on a minibatch of state-action pairs sampled i.i.d. from the training data (*i.e.*, the dataset in the offline setting, and the replay buffer in the online setting). We investigate offline and online RL settings on benchmarks including Atari games (Bellemare et al., 2013) and Gym environments (Brockman et al., 2016). We also utilize gridworlds described by Fu et al. (2019) to compare the learned Q-function against the oracle solution computed using tabular value iteration. We evaluate DQN (Mnih et al., 2015) on gridworld and Atari and SAC (Haarnoja et al., 2018) on Gym domains.

3.1 DATA-EFFICIENT OFFLINE RL

In offline RL, our goal is to learn effective policies by performing Q-learning on a fixed dataset of transitions. We investigate the presence of rank collapse when deep Q-learning is used with broad state coverage offline datasets from Agarwal et al. (2020). In the top row of Figure 2, we show that after an initial learning period, $\text{srank}_\delta(\Phi)$ decreases in all domains (Atari, Gym and the gridworld). The final value of $\text{srank}_\delta(\Phi)$ is often quite small – *e.g.*, in Atari, only 20–100 singular components are active for 512-dimensional features, implying significant underutilization of network capacity. Since under-parameterization is *implicitly* induced by the learning process, even high-capacity value networks behave as low-capacity networks as more training is performed with a bootstrapped objective (*e.g.*, mean squared TD error).

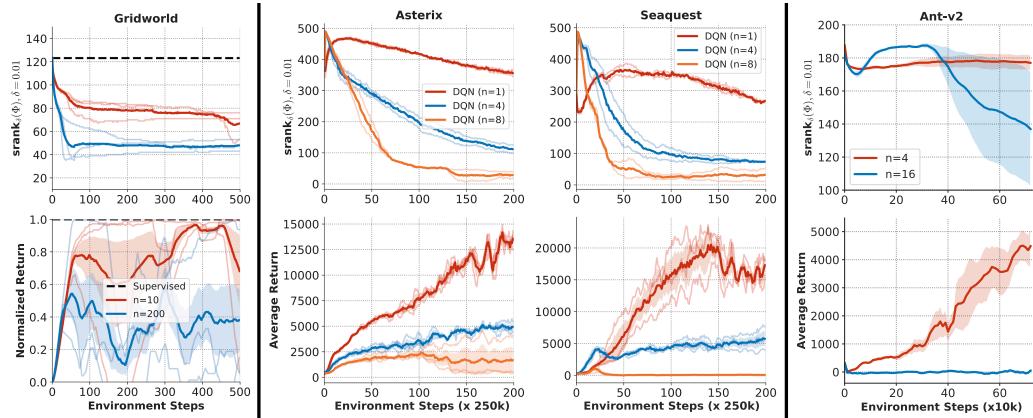


Figure 3: **Data-efficient online RL.** $\text{srank}_\delta(\Phi)$ and performance of neural FQI on gridworld, DQN on Atari and SAC on Gym domains in the online RL setting, with varying numbers of gradient steps per environment step (n). Rank collapse happens earlier with more gradient steps, and the corresponding performance is poor.

On the gridworld environment, regressing to Q^* using supervised regression results in a much higher $\text{srank}_\delta(\Phi)$ (black dashed line in Figure 2(left)) than when using neural FQI. On Atari, even when a 4x larger offline dataset with much broader coverage is used (blue line in Figure 2), rank collapse still persists, indicating that implicit under-parameterization is not due to limited offline dataset size. Figure 2 (2nd row) illustrates that policy performance generally deteriorates as $\text{srank}(\Phi)$ drops, and eventually collapses simultaneously with the rank collapse. While we do not claim that implicit under-parameterization is the only issue in deep Q-learning, the results in Figure 2 show that the emergence of this under-parameterization is *strongly* associated with poor performance.

It is well-known that distributional shift between the offline dataset and the learned policy is a major reason for instability of standard RL algorithms in the offline regime (Fujimoto et al., 2019; Kumar et al., 2019). To control for this potential confounding factor, we also study CQL (Kumar et al., 2020b), a recently-proposed offline RL method designed to handle distribution mismatch by training the Q-function with an additional regularizer that encourages low Q-values at unseen actions. We find a similar degradation in effective rank and performance for CQL (Figure A.3), implying that implicit under-parameterization still appears when methods that correct for distribution shift are employed. The reason is that such methods still operate in the regime where multiple gradient steps are performed on a given unit of experience and so implicit under-parameterization can appear. We provide evidence for implicit under-parameterization in more Atari and Gym environments in Appendix A.1.

3.2 DATA-EFFICIENT ONLINE RL

Deep Q-learning methods typically use very few gradient updates (n) per unit amount of environment interaction (*e.g.*, DQN performs 1 update for every unit environment interaction on Atari (= 4 environment steps) and thus $n = 1$, SAC performs 1 update per unit environment interaction (= 1 environment step). Improving the sample efficiency of these methods requires increasing n to utilize the replay data more effectively. However, we find that using larger values of n results in higher levels of rank collapse as well as performance degradation.

In the top row of Figure 3, we show that larger values of n lead to a more aggressive drop in $\text{srank}_\delta(\Phi)$ (red vs. blue/orange lines), and that rank continues to decrease with more training. Furthermore, the bottom row illustrates that larger values of n result in worse performance, corroborating Fu et al. (2019); Fedus et al. (2020b).

As in the offline setting, directly regressing to Q^* via supervised learning does not cause rank collapse (black line in Figure 3), unlike when using bootstrapping.

Finally, we evaluate data-efficient Rainbow (DER) (van Hasselt et al., 2019) that modifies the Rainbow algorithm (Hessel et al., 2018) to obtain improved performance with limited environment interaction.

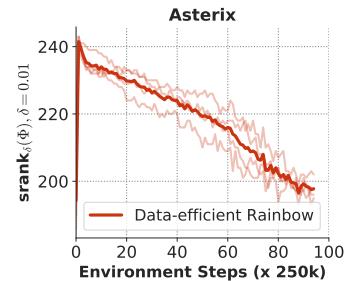


Figure 4: **Data-efficient Rainbow.** $\text{srank}_\delta(\Phi)$ drop for the data efficient rainbow agent (van Hasselt et al., 2019) which uses 4 gradient updates per unit environment interaction ($n = 4$) with multi-step returns.

DER utilizes 4 gradient updates per environment step ($n = 4$) with multi-step returns (20-steps) and a smaller Q-network. We observe that DER also exhibits rank collapse (Figure 4), similarly to the DQN algorithm with more updates. These results indicate that DER also suffers from implicit under-parameterization, and using multi-step returns does not alleviate the issue. When training DER for longer, up to 25M steps, we also observe that it performs suboptimally as compared to a standard Rainbow algorithm ($n = 1$), thus giving rise to similar trends in performance as the DQN variants in Figure 3. We present similar results demonstrating implicit under-parameterization on more Atari and Gym environments in the online data-efficient setting in Appendix A.2.

3.3 UNDERSTANDING IMPLICIT UNDER-PARAMETERIZATION AND ITS IMPLICATIONS

Finally, we aim to understand the primary sources of error behind implicit under-parameterization and and how the emergence of this phenomenon affects the performance of RL algorithms in the data-efficient setting via controlled empirical experiments that we discuss next.

How does implicit under-parameterization degrade performance? Having established the presence of rank collapse in data-efficient RL, we now discuss how it can adversely affect performance. As the effective rank of the network features Φ decreases, so does the network’s ability to fit the subsequent target values, and eventually results in inability to fit Q^* . In the gridworld domain, we measure this loss of expressivity by measuring the error in fitting oracle-computed Q^* values via a linear transformation of Φ . When rank collapse occurs, the error in fitting Q^* steadily increases during training, and the consequent network is not able to predict Q^* at all by the end of training (Figure 5) – this entails a drop in performance. In Atari domains, we do not have access to Q^* , and so we instead measure TD error, that is, the error in fitting the target value estimates, $\mathbf{R} + \gamma P^\pi \mathbf{Q}_k$. In SEAQUEST, as rank decreases, the TD error increases (Figure 6) and the value function is unable to fit the target values, culminating in a performance plateau (Figure 3). This observation is consistent across other environments; we present further supporting evidence in Appendix A.4.

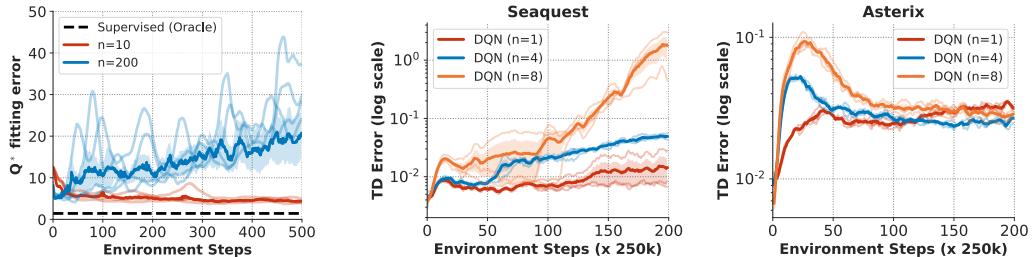


Figure 5: **Q^* Fitting Error.** Fitting error for Q^* prediction for $n = 10$ vs $n = 200$ steps in Figure 3 (left). Observe that rank collapse inhibits fitting Q^* as the fitting error rises over training while rank collapses.

Figure 6: **TD error** (in log-scale) for varying values of n for SEAQUEST and ASTERIX shown in Figure 3 (middle). The value of TD error is larger for larger values of n (see orange vs. red in both the games), indicating that larger values of n that exhibit lower values of the effective feature matrix rank also attain larger TD error in both cases.

Does bootstrapping cause implicit under-parameterization? We perform a number of controlled experiments in the gridworld and Atari environments to isolate the connection between rank collapse and bootstrapping. We first remove confounding with issues of poor network initialization (Fedus et al., 2020a) and non-stationarity (Igl et al., 2020) by showing that rank collapse occurs even when the Q-network is re-initialized from scratch at the start of each fitting iteration (Figure 7). To show that the problem is not isolated to the control setting, we show evidence of rank collapse in the policy evaluation setting as well. We trained a value network using fitted Q-evaluation for a fixed policy π (*i.e.*, using the Bellman operator \mathcal{T}^π instead of \mathcal{T}), and found that rank drop still occurs (FQE in Figure 8). Finally, we show that by removing bootstrapped updates and instead regressing directly to Monte-Carlo (MC) estimates of the value, the effective rank *does not* collapse (MC Returns in Figure 8). These results, along with similar findings on other Atari environments (Appendix A.3), our analysis indicates that bootstrapping is at the core of implicit under-parameterization.

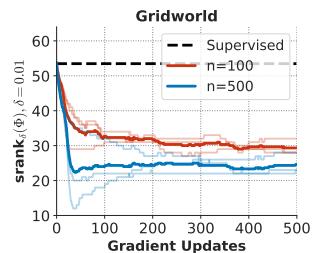


Figure 7: **Periodic reinitialization of Q-networks** still exhibits implicit under-parameterization.

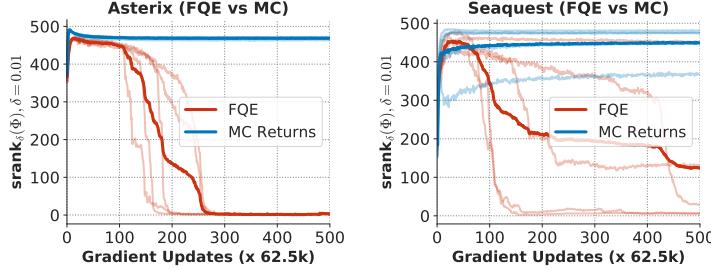


Figure 8: **Bootstrapping vs. Monte-Carlo evaluation.** Trend of $\text{srank}_\delta(\Phi)$ for policy evaluation based on bootstrapped updates (FQE) vs Monte-Carlo returns (no bootstrapping). Note that rank-collapse still persists with reinitialization and FQE, but goes away in the absence of bootstrapping.

4 THEORETICAL ANALYSIS OF IMPLICIT UNDER-PARAMETERIZATION

In this section, we formally analyze implicit under-parameterization and prove that training neural networks with bootstrapping reduces the effective rank of the Q-network, corroborating the empirical observations in the previous section. We focus on policy evaluation (Figure 8 and Figure A.10), where we aim to learn a Q-function that satisfies $\mathbf{Q} = \mathbf{R} + \gamma P^\pi \mathbf{Q}$ for a fixed π , for ease of analysis. We also presume a fixed dataset of transitions, \mathcal{D} , to learn the Q-function.

4.1 ANALYSIS VIA KERNEL REGRESSION

We first study bootstrapping with neural networks through a mathematical abstraction that treats the Q-network as a kernel machine, following the neural tangent kernel (NTK) formalism (Jacot et al., 2018). Building on prior analysis of self-distillation (Mobahi et al., 2020), we assume that each iteration of bootstrapping, the Q-function optimizes the squared TD error to target labels \mathbf{y}_k with a kernel regularizer. Intuitively, this regularizer captures the inductive bias from gradient descent, resembling the regularization imposed by gradient descent under NTK (Mobahi et al., 2020). The error is computed on $(\mathbf{s}_i, \mathbf{a}_i) \in \mathcal{D}$ whereas the regularization imposed by a universal kernel u with a coefficient of $c \geq 0$ is applied to the Q-values at *all* state-action pairs as shown in Equation 1.

$$\mathbf{Q}_{k+1} \leftarrow \arg \min_{\mathbf{Q} \in \mathcal{Q}} \sum_{\mathbf{s}_i, \mathbf{a}_i \in \mathcal{D}} (Q(\mathbf{s}_i, \mathbf{a}_i) - y_k(\mathbf{s}_i, \mathbf{a}_i))^2 + c \sum_{(\mathbf{s}, \mathbf{a})} \sum_{(\mathbf{s}', \mathbf{a}')} u((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}')) Q(\mathbf{s}, \mathbf{a}) Q(\mathbf{s}', \mathbf{a}'). \quad (1)$$

The solution to Equation 1 can be expressed as $Q_{k+1}(\mathbf{s}, \mathbf{a}) = \mathbf{g}_{(\mathbf{s}, \mathbf{a})}^T (c\mathbf{I} + \mathbf{G})^{-1} \mathbf{y}_k$, where \mathbf{G} is the Gram matrix for a special positive-definite kernel (Duffy, 2015) and $\mathbf{g}_{(\mathbf{s}, \mathbf{a})}$ denotes the row of \mathbf{G} corresponding to the input (\mathbf{s}, \mathbf{a}) (Mobahi et al., 2020, Proposition 1). A detailed proof is in Appendix C. When combined with the fitted Q-iteration recursion, setting labels $\mathbf{y}_k = \mathbf{R} + \gamma P^\pi \mathbf{Q}_{k-1}$, we recover a recurrence that relates subsequent value function iterates

$$\mathbf{Q}_{k+1} = \mathbf{G}(c\mathbf{I} + \mathbf{G})^{-1} \mathbf{y}_k = \underbrace{\mathbf{G}(c\mathbf{I} + \mathbf{G})^{-1}}_{\mathbf{A}} [\mathbf{R} + \gamma P^\pi \mathbf{Q}_k] \quad (2)$$

$$= \mathbf{A} \left(\sum_{i=1}^k \gamma^{k-i} (P^\pi \mathbf{A})^{k-i-1} \right) \mathbf{R} := \mathbf{A} \mathbf{M}_k \mathbf{R}. \quad (3)$$

Equation 3 follows from unrolling the recurrence and setting the algorithm-agnostic initial Q-value, \mathbf{Q}_0 , to be $\mathbf{0}$. We now show that the sparsity of singular values of the matrix \mathbf{M}_k generally increases over fitting iterations, implying that the effective rank of \mathbf{M}_k diminishes with more iterations.

Theorem 4.1. *Let \mathbf{S} be a shorthand for $\mathbf{S} = \gamma P^\pi \mathbf{A}$ and assume \mathbf{S} is a normal matrix. Then there exists an infinite, strictly increasing sequence of fitting iterations, $(k_l)_{l=1}^\infty$ starting from $k_1 = 0$, such that, for any two singular-values $\sigma_i(\mathbf{S})$ and $\sigma_j(\mathbf{S})$ of \mathbf{S} with $\sigma_i(\mathbf{S}) < \sigma_j(\mathbf{S})$,*

$$\forall l \in \mathbb{N} \text{ and } l' \geq l, \frac{\sigma_i(\mathbf{M}_{k_{l'}})}{\sigma_j(\mathbf{M}_{k_{l'}})} < \frac{\sigma_i(\mathbf{M}_{k_l})}{\sigma_j(\mathbf{M}_{k_l})} \leq \frac{\sigma_i(\mathbf{S})}{\sigma_j(\mathbf{S})}. \quad (4)$$

Hence, the effective rank of \mathbf{M}_k satisfies: $\text{srank}_\delta(\mathbf{M}_{k_{l'}}) \leq \text{srank}_\delta(\mathbf{M}_{k_l})$. Moreover, if \mathbf{S} is positive semi-definite, then $(k_l)_{l=1}^\infty = \mathbb{N}$, i.e., effective rank continuously decreases in each fitting iteration.

We provide a proof of the theorem above as well as present a stronger variant that shows a gradual decrease in the effective rank for fitting iterations outside this infinite sequence in Appendix C. As k increases along the sequence of iterations given by $k = (k_l)_{l=1}^\infty$, the effective rank of the matrix \mathbf{M}_k drops, leading to gradually decreasing expressivity of this matrix as k increases. Since \mathbf{M}_k linearly maps rewards to the Q-function (Equation 3), drop in expressivity results of \mathbf{M}_k in the inability to model the actual \mathbf{Q}^π .

4.2 ANALYSIS WITH DEEP LINEAR NETWORKS UNDER GRADIENT DESCENT

While Section 4.1 demonstrates rank collapse will occur in a kernel-regression model of Q-learning, it does not illustrate *when* the rank collapse occurs. To better specify the point in training that rank collapse emerges, we present a complementary derivation motivated by implicit regularization of gradient descent over deep linear neural networks (Arora et al., 2019). Our analysis will show that rank collapse can emerge as the generated target values begin to approach the previous value estimate, in particular, when in the vicinity of the optimal Q-function.

Additional notation and assumptions. We represent the Q-function as a deep linear network with at ≥ 3 layers, such that $Q(\mathbf{s}, \mathbf{a}) = \mathbf{W}_N \mathbf{W}_\phi[\mathbf{s}; \mathbf{a}]$, where $N \geq 3$, $\mathbf{W}_N \in \mathbb{R}^{1 \times d_{N-1}}$ and $\mathbf{W}_\phi = \mathbf{W}_{N-1} \mathbf{W}_{N-2} \cdots \mathbf{W}_1$ with $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$ for $i = 1, \dots, N-1$. \mathbf{W}_ϕ maps an input $[\mathbf{s}; \mathbf{a}]$ to corresponding penultimate layer’ features $\Phi(\mathbf{s}, \mathbf{a})$. Let $\mathbf{W}_j(k, t)$ denotes the weight matrix \mathbf{W}_j at the t -th step of gradient descent during the k -th fitting iteration (Algorithm 1). We define $\mathbf{W}_{k,t} = \mathbf{W}_N(k, t) \mathbf{W}_\phi(k, t)$ and $L_{N,k+1}(\mathbf{W}_{k,t})$ as the TD error objective in the k -th fitting iteration. We study $\text{srank}_\delta(\mathbf{W}_\phi(k, t))$ since the rank of features $\Phi = \mathbf{W}_\phi(k, t)[\mathcal{S}, \mathcal{A}]$ is equal to rank of $\mathbf{W}_\phi(k, t)$ provided the state-action inputs have high rank.

We assume that the evolution of the network weights is governed by a continuous-time differential equation (Arora et al., 2018) *within* each fitting iteration k . Similar to Arora et al. (2018; 2019), we assume that all except the last-layer weights share singular values (a.k.a. “balancedness”). In this model, we can characterize the evolution of the singular values of the feature matrix $\mathbf{W}_\phi(k, t)$, using techniques analogous to Arora et al. (2019):

Proposition 4.1. *The singular values of the feature matrix $\mathbf{W}_\phi(k, t)$ evolve according to:*

$$\dot{\sigma}_r(k, t) = -N \cdot (\sigma_r^2(k, t))^{1 - \frac{1}{N-1}} \cdot \left\langle \mathbf{W}_N(k, t)^T \frac{dL_{N,k+1}(\mathbf{W}_{K,t})}{d\mathbf{W}}, \mathbf{u}_r(k, t) \mathbf{v}_r(k, t)^T \right\rangle, \quad (5)$$

for $r = 1, \dots, \min_{i=1}^{N-1} d_i$, where $\mathbf{u}_r(k, t)$ and $\mathbf{v}_r(k, t)$ denote the left and right singular vectors of the feature matrix, $\mathbf{W}_\phi(k, t)$, respectively.

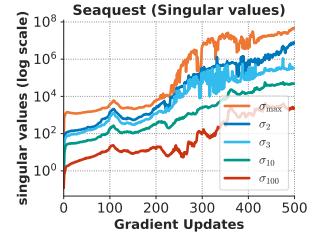
Solving the differential equation (5) tells us that larger singular values will evolve at a exponentially faster rate than smaller singular values (as we also formally show in Appendix D.1) and the difference in their magnitudes disproportionately increase with increasing t . This behavior also occurs empirically, illustrated in the figure on the right (also see Figure D.1), where larger singular values are orders of magnitude larger than smaller singular values. Hence, the effective rank of the features, $\text{srank}_\delta(\mathbf{W}_\phi(k, t))$, will decrease with more gradient steps *within* a fitting iteration k .

Explaining implicit under-parameterization across fitting iterations.

Since gradient descent within a fitting iteration has an implicit bias towards solutions with low effective rank, we can express the solution obtained at the end of a fitting iteration k as (with $\lambda_k > 0$)

$$\min_{\mathbf{W}_\phi, \mathbf{W}_N} \|\mathbf{W}_N \mathbf{W}_\phi[\mathbf{s}; \mathbf{a}] - \mathbf{y}_k(\mathbf{s}, \mathbf{a})\|^2 + \lambda_k \text{srank}_\delta(\mathbf{W}_\phi). \quad (6)$$

This provides an abstraction to analyze the evolution of $\text{srank}_\delta(\mathbf{W}_\phi)$ across fitting iterations. Before considering the full bootstrapping setting, we first analyze the self-training setting, when the target values are equal to the previous Q-values, $y_k(\mathbf{s}, \mathbf{a}) = Q_{k-1}(\mathbf{s}, \mathbf{a}) = \mathbf{W}_N(k-1, T) \mathbf{W}_\phi(k-1, T)[\mathbf{s}; \mathbf{a}]$. Since just copying over weights from iteration $k-1$ to k attains zero TD error without increasing effective rank, the optimal solution $\mathbf{W}_\phi(k, T)$ for Equation 6 must attain a smaller value of $\text{srank}_\delta(\mathbf{W}_\phi(k, t))$. This tradeoff between Bellman error and $\text{srank}_\delta(\mathbf{W}_\phi)$ is also observed



Evolution of singular values of \mathbf{W}_ϕ on SEAQUEST

in practice (*c.f.* Figure 3 (middle) and Figure 5). Applying this argument inductively implies that $\text{srank}_\delta(\mathbf{W}_\phi(k, t))$ decreases with increasing k , right from the start (*i.e.*, $k = 0$) with self-training.

Case with bootstrapped updates in RL. In the RL setting, the target values are now given by $y_k(s, a) = r(s, a) + \gamma P^\pi Q_{k-1}(s, a)$. Unlike the self-training setting, $y_k(s, a)$ is not directly expressible as a function of the previous $\mathbf{W}_\phi(k, T)$ due to additional reward and dynamics transformations, and the rank of the solution to Equation 6 may increase initially. However, as the value function gets closer to the Bellman fixed point, the learning dynamics begins to resemble the self-training regime, since the target values approach the previous value iterate $\mathbf{y}_k \approx \mathbf{Q}_{k-1}$. As learning approaches this regime, the rank decrease phenomenon discussed above begins to occur and under-parameterization emerges. This insight is captured by our next result, which shows rank decrease when target values are sufficiently close to the previous value estimate (*e.g.*, near convergence).

Theorem 4.2. *Suppose target values $\mathbf{y}_k = \mathbf{R} + \gamma P^\pi \mathbf{Q}_{k-1}$ are close to the previous value estimate \mathbf{Q}_{k-1} , i.e. $\forall s, a$, $y_k(s, a) = Q_{k-1}(s, a) + \varepsilon(s, a)$. Then, there is a constant ϵ_0 such that whenever $\|\varepsilon\| < \epsilon_0$, the solution to Equation 6, \mathbf{Q}_k , has lower effective rank than \mathbf{Q}_{k-1} :*

$$\text{srank}_\delta(\mathbf{W}_\phi(k, T)) \leq \text{srank}_\delta(\mathbf{W}_\phi(k-1, T)) - \|\mathbf{Q}_k - \mathbf{y}_k\|/\lambda_k \quad (7)$$

The constant ϵ_0 depends on the magnitude of \mathbf{W}_N and \mathbf{W}_Φ as well as the gap between singular values of \mathbf{W}_ϕ ; we provide the complete form in Appendix D.2. One important consequence of this theorem is that rank decrease occurs when the value function is near (but not at) the fixed point, since gradient descent will preferentially choose solutions with decreased effective rank, which can also increase TD error. This is consistent with the empirical evidence in Figure 2, where rank collapse happens right after a peak in performance is achieved.

5 MITIGATING UNDER-PARAMETRIZATION IMPROVES DEEP Q-LEARNING

We now show that mitigating implicit under-parameterization by preventing rank collapse can improve performance. We place special emphasis on the offline RL setting in this section, since it is particularly vulnerable to the adverse effects of rank collapse.

We devise a penalty (or a regularizer) $\mathcal{L}_p(\Phi)$ that encourages higher effective rank of the learned features, $\text{srank}_\delta(\Phi)$, to prevent rank collapse. The effective rank function $\text{srank}_\delta(\Phi)$ is non-differentiable, so we choose a simple surrogate that can be optimized over deep networks. Since effective rank is maximized when the magnitude of the singular values is roughly balanced, one way to increase effective rank is to minimize the largest singular value of Φ , $\sigma_{\max}(\Phi)$, while simultaneously maximizing the smallest singular value, $\sigma_{\min}(\Phi)$. We construct a simple penalty $\mathcal{L}_p(\Phi)$ derived from this intuition, formally given by:

$$\mathcal{L}_p(\Phi) = \sigma_{\max}^2(\Phi) - \sigma_{\min}^2(\Phi). \quad (8)$$

$\mathcal{L}_p(\Phi)$ can be computed by invoking the singular value decomposition subroutines in standard automatic differentiation frameworks. We estimate the singular values over the feature matrix computed over a minibatch, and add the resulting value of \mathcal{L}_p as a penalty to the TD error objective, with a tradeoff factor $\alpha = 0.001$.

Does $\mathcal{L}_p(\Phi)$ address rank collapse? We first verify whether controlling the minimum and maximum singular values using $\mathcal{L}_p(\Phi)$ actually prevents rank collapse. When using this penalty on the gridworld problem (Figure 9a), the effective rank does not collapse, instead gradually decreasing at the onset and then plateauing, akin to the evolution of effective rank in supervised learning. In Figure 9b, we plot the evolution of the effective rank values on two Atari games in the offline setting, SEAQUEST

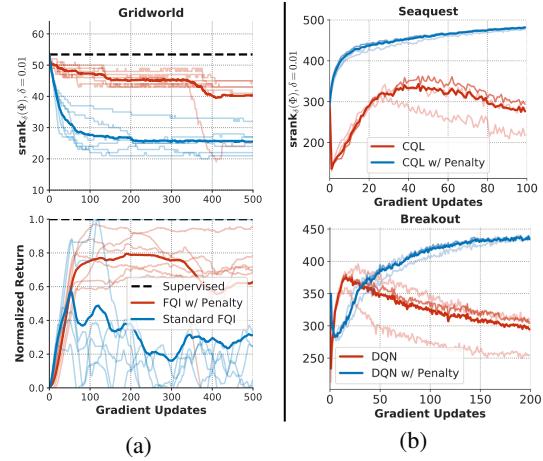


Figure 9: (a): $\text{srank}_\delta(\Phi)$ (top) and performance (bottom) of FQI on gridworld in the offline setting with 200 gradient updates per fitting iteration. Note the reduction in rank collapse and higher performance with the regularizer $\mathcal{L}_p(\Phi)$. (b): $\mathcal{L}_p(\Phi)$ mitigates the rank collapse in DQN and CQL in the offline RL setting on Atari.

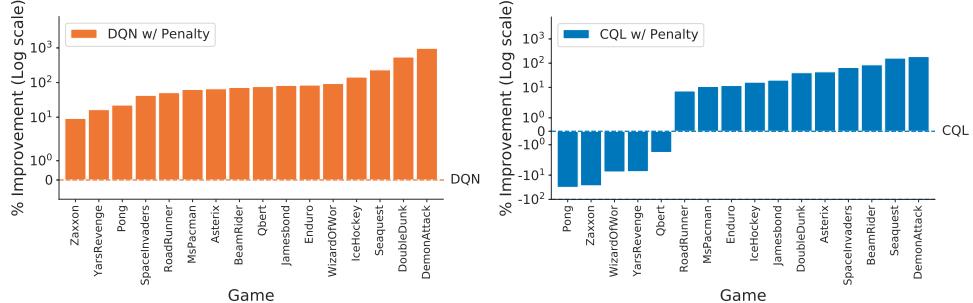


Figure 10: DQN and CQL with $\mathcal{L}_p(\Phi)$ penalty vs. their standard counterparts in the 5% offline setting on Atari from Section 3. \mathcal{L}_p improves DQN on 16/16 and CQL on 11/16 games.

and BREAKOUT (all games in Appendix A.5), and observe that the addition of the penalty, \mathcal{L}_p , also generally leads to increasing ranks.

Does mitigating rank collapse improve performance? We now evaluate the performance of the penalty using DQN (Mnih et al., 2015) and CQL (Kumar et al., 2020b) on Atari dataset from Agarwal et al. (2020) (5% replay data), used in Section 3. Figure 10 summarizes the relative improvement from using the penalty for 16 Atari games. Adding the penalty to DQN improves performance on all 16/16 games with a median improvement of **74.5%**; adding it to CQL, a state-of-the-art offline algorithm, improves performance on 11/16 games with median improvement of **14.1%**. Prior work has discussed that standard Q-learning methods designed for the online setting, such as DQN, are generally ineffective with small offline datasets (Kumar et al., 2020b; Agarwal et al., 2020). Our results show that mitigating rank collapse makes even such simple methods substantially more effective in this setting, suggesting that rank collapse and the resulting implicit under-parameterization may be an crucial piece of the puzzle in explaining the challenges of offline RL.

We also evaluated the regularizer $\mathcal{L}_p(\Phi)$ in the data-efficient online RL setting, with results in Appendix A.6. This variant achieved median improvement of 20.6% performance with Rainbow (Hessel et al., 2018), however performed poorly with DQN, where it reduced median performance by 11.5%. Thus, while our proposed penalty is effective in many cases spanning both offline and data-efficient online settings, it does not solve the problem fully, and a more sophisticated solution may better prevent the issues with implicit under-parameterization. Nevertheless, our results suggest that mitigation of implicit under-parameterization can improve performance of data-efficient RL.

6 RELATED WORK

Prior work has extensively studied the learning dynamics of Q-learning with tabular and linear function approximation, to study error propagation (Munos, 2003; Farahmand et al., 2010; Chen & Jiang, 2019) and to prevent divergence (De Farias, 2002; Maei et al., 2009; Sutton et al., 2009; Dai et al., 2018), as opposed to deep Q-learning analyzed in this work. Q-learning has been shown to have favorable optimization properties with certain classes of features (Ghosh & Bellemare, 2020), but our work shows that the features learned by a neural net when minimizing TD error do not enjoy such guarantees, and instead suffer from rank collapse. Recent theoretical analyses of deep Q-learning have shown convergence under restrictive assumptions (Yang et al., 2020; Cai et al., 2019; Zhang et al., 2020; Xu & Gu, 2019), but Theorem 4.2 shows that implicit under-parameterization appears when the estimates of the value function approach the optimum, potentially preventing convergence. Several works including Chen & Jiang (2019); Du et al. (2019; 2020); Xie & Jiang (2020) attempted to address complexity of fitted Q-iteration with restricted function classes (and not neural nets), however our work is distinct in that it does not aim to provide complexity guarantees and is focused more on understanding the learning dynamics of deep Q-learning. Xu et al. (2005; 2007) present variants of LSTD (Boyan, 1999; Lagoudakis & Parr, 2003), which model the Q-function as a kernel-machine but do not take into account the regularization from gradient descent, as done in Equation 1, which is essential for implicit under-parameterization.

Igl et al. (2020); Fedus et al. (2020a) argue that non-stationarity arising from distribution shift hinders generalization and recommend periodic network re-initialization. Under-parameterization is not caused by this distribution shift, and we find that network re-initialization does little to prevent rank collapse (Figure 7). Luo et al. (2020) proposes a regularization similar to ours, but in a different setting, finding

that more expressive features increases performance of on-policy RL methods. Finally, Yang et al. (2019) study the effective rank of the Q^* -values when expressed as a $|\mathcal{S}| \times |\mathcal{A}|$ matrix in online RL and find that low ranks for this Q^* -matrix are preferable. We analyze a fundamentally different object: the learned features (and illustrate that a rank-collapse of features can hurt), not the Q^* -matrix, whose rank is upper-bounded by the number of actions (*e.g.*, 24 for Atari).

7 DISCUSSION

We identified an implicit under-parameterization phenomenon in deep RL algorithms that use bootstrapping, where gradient-based optimization of a bootstrapped objective can lead to a reduction in the expressive power of the value network. This effect manifests as a collapse of the rank of the features learned by the value network, causing aliasing across states and often leading to poor performance. Our analysis reveals that this phenomenon is caused by the implicit regularization due to gradient descent on bootstrapped objectives. We observed that mitigating this problem by means of a simple regularization scheme improves performance of deep Q-learning methods.

While our proposed regularization provides some improvement, devising better mitigation strategies for implicit under-parameterization remains an exciting direction for future work. Our method explicitly attempts to prevent rank collapse, but relies on the emergence of useful features solely through the bootstrapped signal. An alternative path may be to develop new auxiliary losses (*e.g.*, Jaderberg et al., 2016) that learn useful features while passively preventing underparameterization. More broadly, understanding the effects of neural nets and associated factors such as initialization, choice of optimizer, *etc.* on the learning dynamics of deep RL algorithms, using tools from deep learning theory, is likely to be key towards developing robust and data-efficient deep RL algorithms.

ACKNOWLEDGEMENTS

We thank Lihong Li, Aaron Courville, Aurick Zhou, Abhishek Gupta, George Tucker, Ofir Nachum, Wesley Chung, Emmanuel Bengio, Zafarali Ahmed, and Jacob Buckman for feedback on an earlier version of this paper. We thank Hossein Mobahi for insightful discussions about self-distillation and Hanie Sedghi for insightful discussions about implicit regularization and generalization in deep networks. We additionally thank Michael Janner, Aaron Courville, Dale Schuurmans and Marc Bellemare for helpful discussions. AK was partly funded by the DARPA Assured Autonomy program, and DG was supported by a NSF graduate fellowship and compute support from Amazon.

REFERENCES

- Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *ArXiv*, abs/1903.08894, 2019.
- Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.
- Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. *arXiv preprint arXiv:1802.06509*, 2018.
- Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, pp. 7413–7424, 2019.
- Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Int. Res.*, 47(1):253–279, May 2013. ISSN 1076-9757.
- Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 449–458. JMLR.org, 2017.
- Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. *arXiv preprint arXiv:2003.06350*, 2020.
- Justin A Boyan. Least-squares temporal difference learning. In *ICML*, pp. 49–56. Citeseer, 1999.

- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
- Qi Cai, Zhuoran Yang, Jason D Lee, and Zhaoran Wang. Neural temporal-difference and q-learning provably converge to global optima. *arXiv preprint arXiv:1905.10027*, 2019.
- Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.
- Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. *arXiv preprint arXiv:1905.00360*, 2019.
- Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. Sbeed: Convergent reinforcement learning with nonlinear function approximation. In *International Conference on Machine Learning*, pp. 1133–1142, 2018.
- Daniela Pucci De Farias. *The linear programming approach to approximate dynamic programming: Theory and application*. PhD thesis, 2002.
- Simon Du, Yuping Luo, Ruosong Wang, and Hanrui Zhang. Provably efficient q -learning with function approximation via distribution shift error checking oracle. In *NeurIPS*, 06 2019.
- Simon S Du, Jason D Lee, Gaurav Mahajan, and Ruosong Wang. Agnostic q -learning with function approximation in deterministic systems: Tight bounds on approximation error and sample complexity. *arXiv preprint arXiv:2002.07125*, 2020.
- Dean G Duffy. *Green's functions with applications*. CRC Press, 2015.
- Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.
- Amir-massoud Farahmand, Csaba Szepesvári, and Rémi Munos. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.
- William Fedus, Dibya Ghosh, John D Martin, Marc G Bellemare, Yoshua Bengio, and Hugo Larochelle. On catastrophic interference in atari 2600 games. *arXiv preprint arXiv:2002.12499*, 2020a.
- William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. *ICML*, 2020b.
- Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q -learning algorithms. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019.
- Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.
- Dibya Ghosh and Marc G Bellemare. Representations for stable off-policy reinforcement learning. *arXiv preprint arXiv:2007.05520*, 2020.
- Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. RI unplugged: Benchmarks for offline reinforcement learning. 2020.
- Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pp. 6151–6159, 2017.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL <http://arxiv.org/abs/1801.01290>.
- Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Edmund Hlawka, Rudolf Taschner, and Johannes Schoißengeier. *The Dirichlet Approximation Theorem*, pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. ISBN 978-3-642-75306-0. doi: 10.1007/978-3-642-75306-0_1. URL https://doi.org/10.1007/978-3-642-75306-0_1.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*. 2018.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Robert Johnson. Approximate irrational numbers by rational numbers, 2016. URL <https://math.stackexchange.com/questions/1829743/>.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. 2019. URL <http://arxiv.org/abs/1906.00949>.

Aviral Kumar, Abhishek Gupta, and Sergey Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *arXiv preprint arXiv:2003.07305*, 2020a.

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

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

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pp. 45–73. Springer, 2012.

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

Vincent Liu, Raksha Kumaraswamy, Lei Le, and Martha White. The utility of sparse representations for control in reinforcement learning. *CoRR*, abs/1811.06626, 2018. URL <http://arxiv.org/abs/1811.06626>.

Xufang Luo, Qi Meng, Di He, Wei Chen, and Yunhong Wang. I4r: Promoting deep reinforcement learning by the indicator for expressive representations. In Christian Bessiere (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 2669–2675. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/370. URL <https://doi.org/10.24963/ijcai.2020/370>. Main track.

Hamid R. Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, 2009.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharrshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015. ISSN 0028-0836.

Hossein Mobahi, Mehrdad Farajtabar, and Peter L Bartlett. Self-distillation amplifies regularization in hilbert space. *arXiv preprint arXiv:2002.05715*, 2020.

- Rémi Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 560–567. AAAI Press, 2003. ISBN 1577351894.
- Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.
- Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pp. 317–328. Springer, 2005.
- David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
- Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning (ICML)*, 2009.
- James Townsend. Differentiating the singular value decomposition. Technical report, Technical Report 2016, <https://j-towns.github.io/papers/svd-derivative...>, 2016.
- Hado P van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, pp. 14322–14333, 2019.
- Tengyang Xie and Nan Jiang. Batch value-function approximation with only realizability. *arXiv preprint arXiv:2008.04990*, 2020.
- Pan Xu and Quanquan Gu. A finite-time analysis of q-learning with neural network function approximation. *arXiv preprint arXiv:1912.04511*, 2019.
- Xin Xu, Tao Xie, Dewen Hu, and Xicheng Lu. Kernel least-squares temporal difference learning. *International Journal of Information Technology*, 11(9):54–63, 2005.
- Xin Xu, Dewen Hu, and Xicheng Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.
- Yuzhe Yang, Guo Zhang, Zhi Xu, and Dina Katabi. Harnessing structures for value-based planning and reinforcement learning. *arXiv preprint arXiv:1909.12255*, 2019.
- Zhuoran Yang, Yuchen Xie, and Zhaoran Wang. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pp. 486–489. PMLR, 2020.
- Yufeng Zhang, Qi Cai, Zhuoran Yang, Yongxin Chen, and Zhaoran Wang. Can temporal-difference and q-learning learn representation? a mean-field theory. *arXiv preprint arXiv:2006.04761*, 2020.

Appendices

A ADDITIONAL EVIDENCE FOR IMPLICIT UNDER-PARAMETERIZATION

In this section, we present additional evidence that demonstrates the existence of the implicit under-parameterization phenomenon from Section 3. In all cases, we plot the values of $\text{srank}_\delta(\Phi)$ computed on a batch size of 2048 i.i.d. sampled transitions from the dataset.

A.1 OFFLINE RL

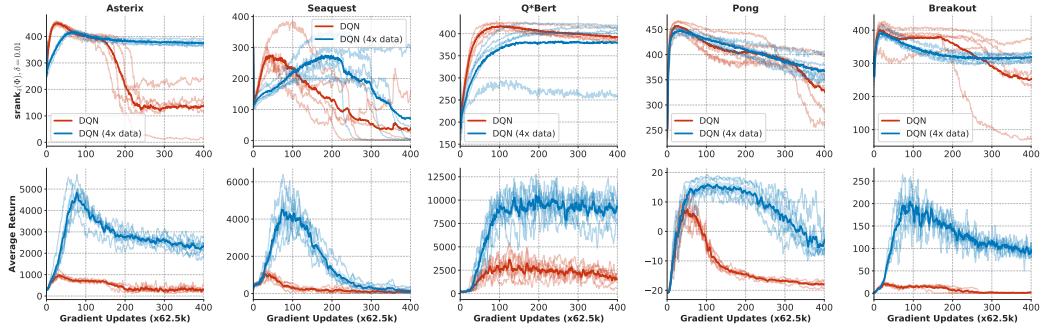


Figure A.1: **Offline DQN on Atari.** $\text{srank}_\delta(\Phi)$ and performance of DQN on five Atari games in the offline RL setting using the 5% DQN Replay dataset (Agarwal et al., 2020) (marked as **DQN**) and a larger 20% DQN Replay dataset (Agarwal et al., 2020) (marked as **DQN (4x data)**). Note that low srank (top row) generally corresponds to worse policy performance (bottom row). Average across 5 runs is showed for each game with individual runs.

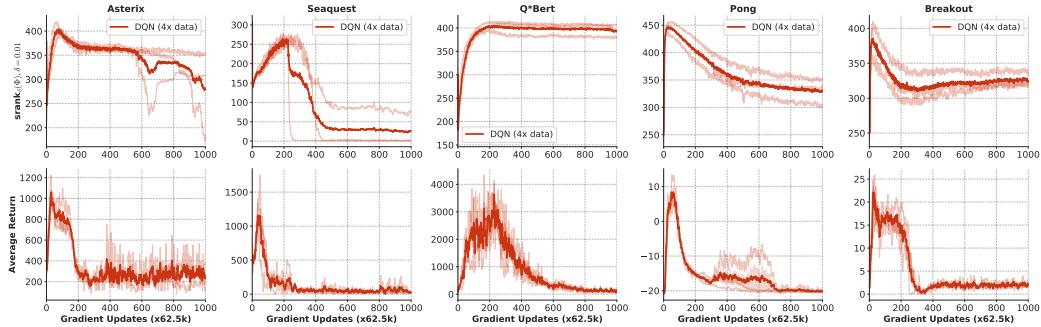


Figure A.2: **Offline DQN on Atari.** $\text{srank}_\delta(\Phi)$ and performance of DQN on five Atari games in the offline RL setting using the 20% DQN Replay dataset (Agarwal et al., 2020) (marked as **DQN**) trained for 1000 iterations. Note that low srank (top row) generally corresponds to worse policy performance (bottom row). Average across 5 runs is showed for each game with individual runs.

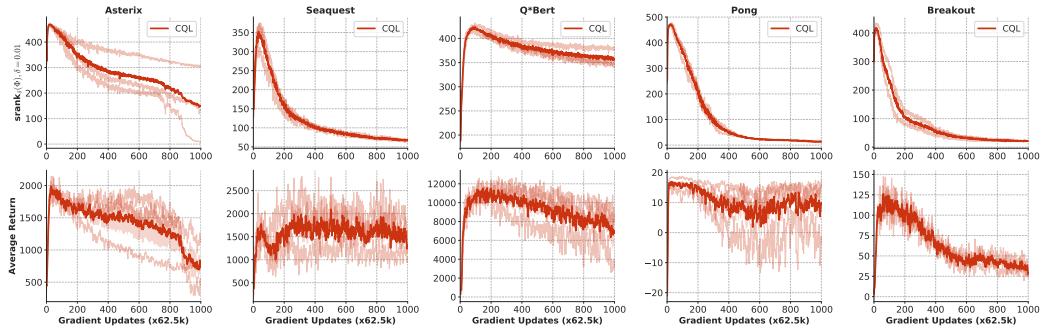


Figure A.3: **Offline CQL on Atari.** $\text{srank}_\delta(\Phi)$ and performance of CQL on five Atari games (average across 5 runs) in the offline RL setting using the 5% DQN Replay. Rank values degrade significantly with prolonged training and this corresponds to a sharp drop in performance. Note that lower rank values than a DQN is likely because CQL trains Q-functions with an additional regularizer.

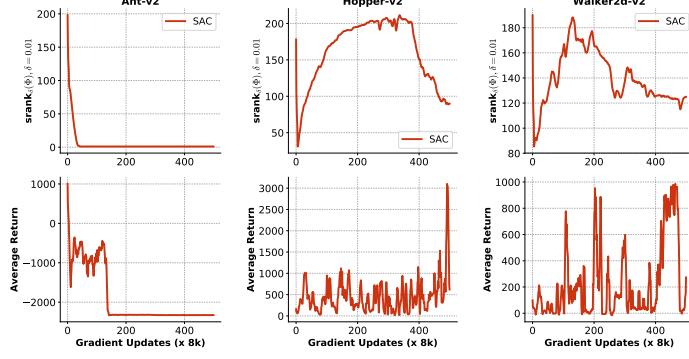


Figure A.4: **Offline Control on MuJoCo.** $\text{srank}_\delta(\Phi)$ and performance of SAC on three Gym environments the offline RL setting. Implicit under-parameterization is conspicuous from the rank reduction, which highly correlates with performance degradation. We use 20% uniformly sampled data from the entire replay experience of an online SAC agent, similar to the 20% setting from [Agarwal et al. \(2020\)](#).

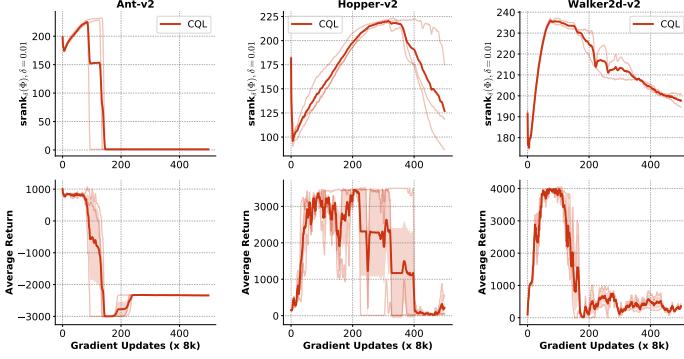


Figure A.5: **Offline Control on MuJoCo.** $\text{srank}_\delta(\Phi)$ and performance of CQL on three Gym environments the offline RL setting. Implicit under-parameterization is conspicuous from the rank reduction, which highly correlates with performance degradation. We use 20% uniformly sampled data from the entire replay experience of an online SAC agent, similar to the 20% setting from [Agarwal et al. \(2020\)](#).

A.2 DATA EFFICIENT ONLINE RL

In the data-efficient online RL setting, we verify the presence of implicit under-parameterization on both DQN and Rainbow ([Hessel et al., 2018](#)) algorithms when larger number of gradient updates are made per environment step. In these settings we find that more gradient updates per environment step lead to a larger decrease in effective rank, whereas effective rank can increase when the amount of data re-use is reduced by taking fewer gradient steps.

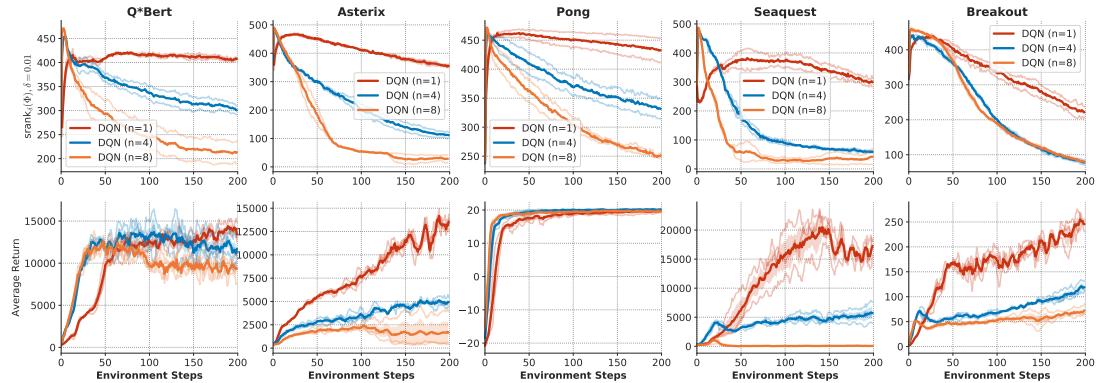


Figure A.6: **Online DQN on Atari.** $\text{srank}_\delta(\Phi)$ and performance of DQN on 5 Atari games in the online RL setting, with varying numbers of gradient steps per environment step (n). Rank collapse happens earlier with more gradient steps, and the corresponding performance is poor. This indicates that implicit under-parameterization aggravates as the rate of data re-use is increased.

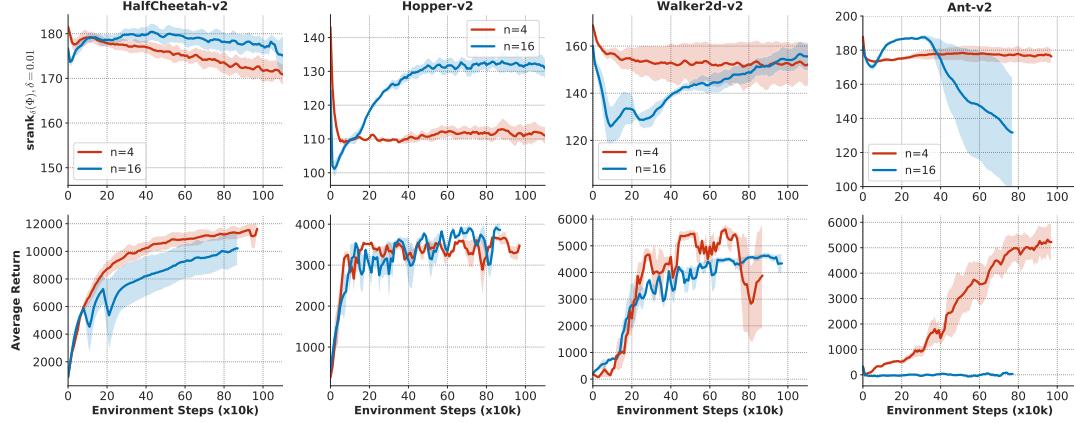


Figure A.7: **Online SAC on MuJoCo.** $srank_{\delta}(\Phi)$ and performance of SAC on three Gym environments the online RL setting, with varying numbers of gradient steps per environment step (n). While in the simpler environments, HalfCheetah-v2, Hopper-v2 and Walker2d-v2 we actually observe an increase in the values of effective rank, which also corresponds to good performance with large n values in these cases, on the more complex Ant-v2 environment rank decreases with larger n , and the corresponding performance is worse with more gradient updates.

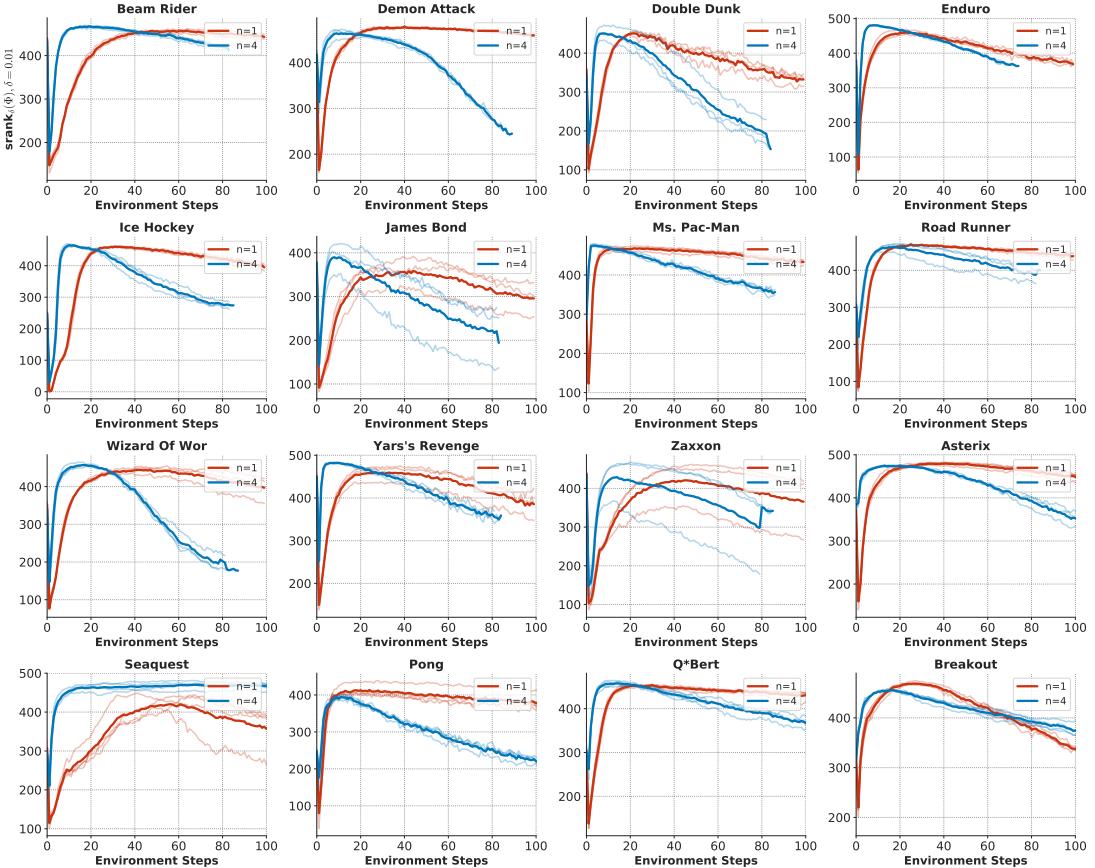


Figure A.8: **Online Rainbow on Atari.** $srank_{\delta}(\Phi)$ Rainbow on 16 Atari games in the data-efficient online setting, with varying numbers of gradient steps per environment step (n). Rank collapse happens earlier with more gradient steps, and the corresponding performance is poor. This plot indicates the multi-step returns, prioritized replay or distributional C51 does not address the implicit under-parameterization issue.

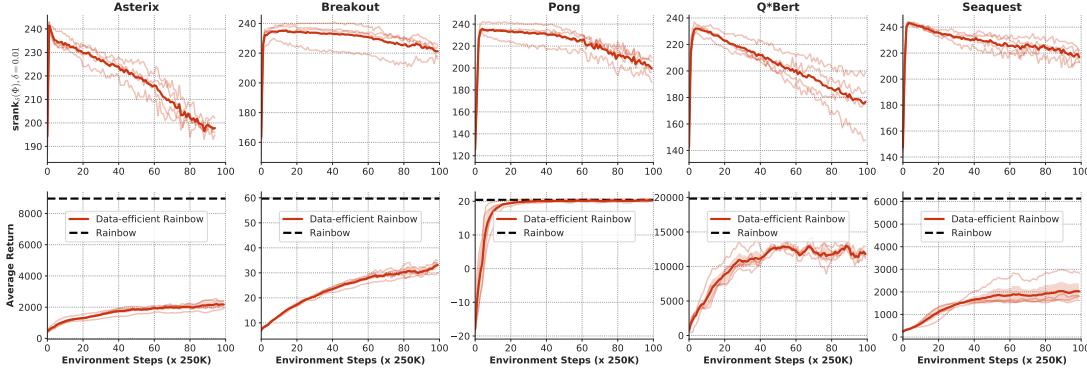


Figure A.9: **Data Efficient Rainbow on Atari.** $\text{rank}_{\delta}(\Phi)$ data-efficient Rainbow on 5 Atari games in the online setting. The horizontal line shows the performance of the Rainbow agent at the end of 100 iterations where each iteration is 250K environment steps. Data-efficient Rainbow performs poorly compared to online Rainbow.

A.3 DOES BOOTSTRAPPING CAUSE IMPLICIT UNDER-PARAMETERIZATION?

In this section, we provide additional evidence to support our claim from Section 3 that suggests that bootstrapping-based updates are a key component behind the existence of implicit under-parameterization. To do so, we empirically demonstrate the following points empirically:

- Implicit under-parameterization occurs even when the form of the bootstrapping update is changed from Q-learning that utilizes a $\max_{a'}$ backup operator to a policy evaluation (fitted Q-evaluation) backup operator, that computes an expectation of the target Q-values under the distributions specified by a different policy. **Thus, with different bootstrapped updates, the phenomenon still appears.**

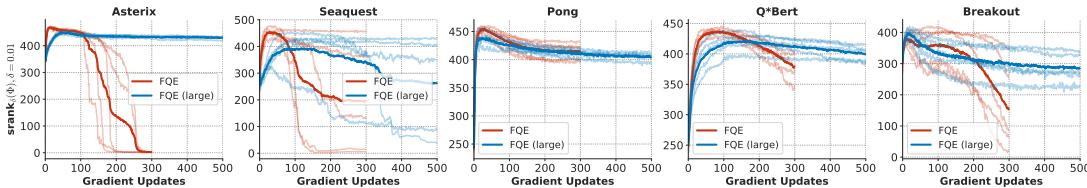


Figure A.10: **Offline Policy Evaluation on Atari.** $\text{rank}_{\delta}(\Phi)$ and performance of offline policy evaluation (FQE) on 5 Atari games in the offline RL setting using the 5% and 20% (marked as “large”) DQN Replay dataset (Agarwal et al., 2020). The rank degradation shows that under-parameterization is not specific to the Bellman optimality operator but happens even when other bootstrapping-based backup operators are combined with gradient descent. Furthermore, the rank degradation also happens when we increase the dataset size.

- Implicit under-parameterization does not occur when Monte-Carlo regression targets - that compute regression targets for the Q-function by computing a non-parametric estimate the future trajectory return, *i.e.*, $y_k(s_t, a_t) = \sum_{t'=t}^{\infty} \gamma^{t'} r(s_{t'}, a_{t'})$ and do not use bootstrapping. In this setting, we find that the values of effective rank actually increase over time and stabilize, unlike the corresponding case for bootstrapped updates. **Thus, other factors kept identically the same, implicit under-parameterization happens only when bootstrapped updates are used.**

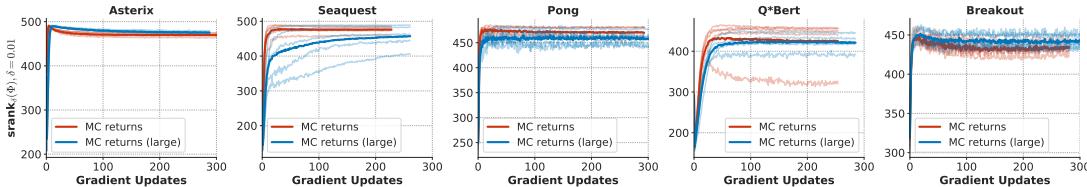


Figure A.11: **Monte Carlo Offline Policy Evaluation.** $\text{rank}_{\delta}(\Phi)$ on 5 Atari games in when using Monte Carlo returns for targets and thus removing bootstrapping updates. Rank collapse does not happen in this setting implying that bootstrapping was essential for under-parameterization. We perform the experiments using 5% and 20% (marked as “large” in the figure) DQN replay dataset from Agarwal et al. (2020).

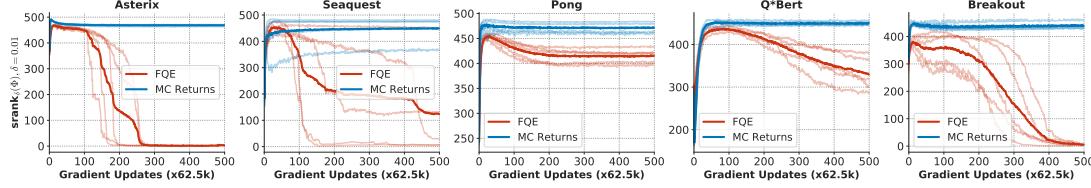


Figure A.12: **FQE vs. Monte Carlo Offline Policy Evaluation.** Trend of $\text{srank}_\delta(\Phi)$ for policy evaluation based on bootstrapped updates (FQE) vs Monte-Carlo returns (no bootstrapping). Note that rank-collapse still persists with reinitialization and FQE, but goes away in the absence of bootstrapping. We perform the experiments using 5% DQN replay dataset from Agarwal et al. (2020).

A.4 HOW DOES IMPLICIT REGULARIZATION INHIBIT DATA-EFFICIENT RL?

Implicit under-parameterization leads to a trade-off between minimizing the TD error *vs.* encouraging low rank features as shown in Figure 6. This trade-off often results in decrease in effective rank, at the expense of increase in TD error, resulting in lower performance. Here we present additional evidence to support this.

Figure A.13 shows a gridworld problem with one-hot features, which naturally leads to reduced state-aliasing. In this setting, we find that the amount of rank drop with respect to the supervised projection of oracle computed Q^* values is quite small and the regression error to Q^* actually decreases unlike the case in Figure 5, where it remains same or even increases. The method is able to learn policies that attain good performance as well. Hence, this justifies that when there's very little rank drop, for example, 5 rank units in the example on the right, FQI methods are generally able to learn Φ that is able to fit Q^* . This provides evidence showing that typical Q-networks learn Φ that can fit the optimal Q-function when rank collapse does not occur.

In Atari, we do not have access to Q^* , and so we instead measure the error in fitting the target value estimates, $\mathbf{R} + \gamma P^\pi \mathbf{Q}_k$. As rank decreases, the TD error increases (Figure A.14) and the value function is unable to fit the target values, culminating in a performance plateau (Figure A.6).

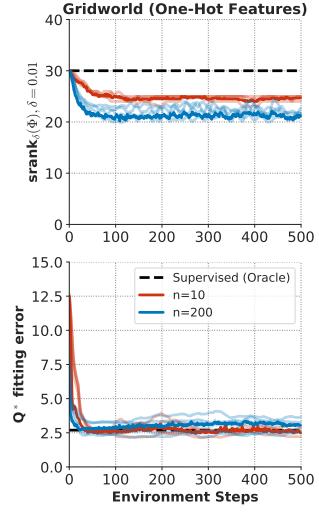


Figure A.13: Q^* fitting error and srank in a one-hot variant of the gridworld environment.

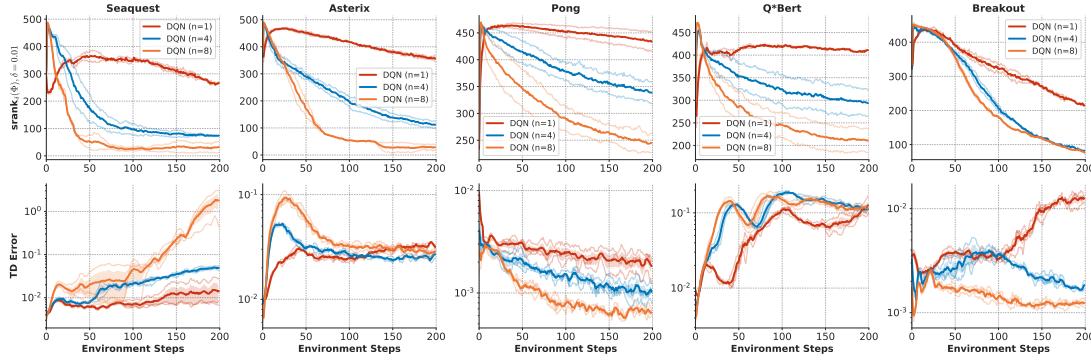


Figure A.14: **TD error vs. Effective rank on Atari.** We observe that Huber-loss TD error is often higher when there is a larger implicit under-parameterization, measured in terms of drop in effective rank. The results are shown for the data-efficient online RL setting.

A.5 TRENDS IN VALUES OF EFFECTIVE RANK WITH PENALTY.

In this section, we present the trend in the values of the effective rank when the penalty $\mathcal{L}_p(\Phi)$ is added. In each plot below, we present the value of $\text{srank}_\delta(\Phi)$ with and without penalty respectively.

A.5.1 OFFLINE RL: DQN

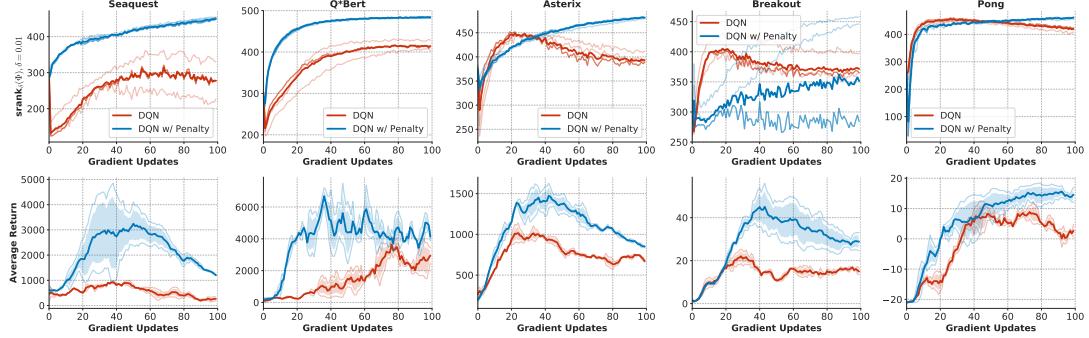


Figure A.15: **Effective rank values with the penalty on DQN.** Trends in effective rank and performance for offline DQN. Note that the performance of DQN with penalty is generally better than DQN and that the penalty (blue) is effective in increasing the values of effective rank. We report performance at the end of 100 epochs, as per the protocol set by Agarwal et al. (2020) in Figure 10.

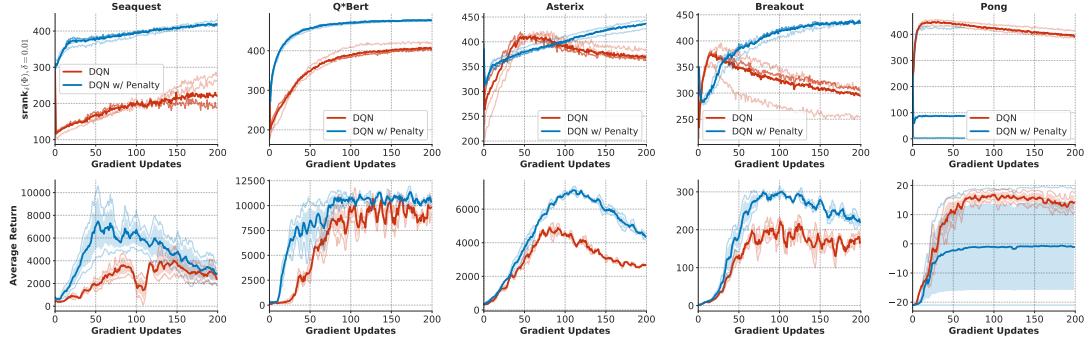


Figure A.16: **Effective rank values with the penalty on DQN on a 4x larger dataset.** Trends in effective rank and performance for offline DQN with a 4x larger dataset, where distribution shift effects are generally removed. Note that the performance of DQN with penalty is generally better than DQN and that the penalty (blue) is effective in increasing the values of effective rank in most cases. Infact in PONG, where the penalty is not effective in increasing rank, we observe suboptimal performance (blue vs. red).

A.5.2 OFFLINE RL: CQL WITH $L_p(\Phi)$ PENALTY

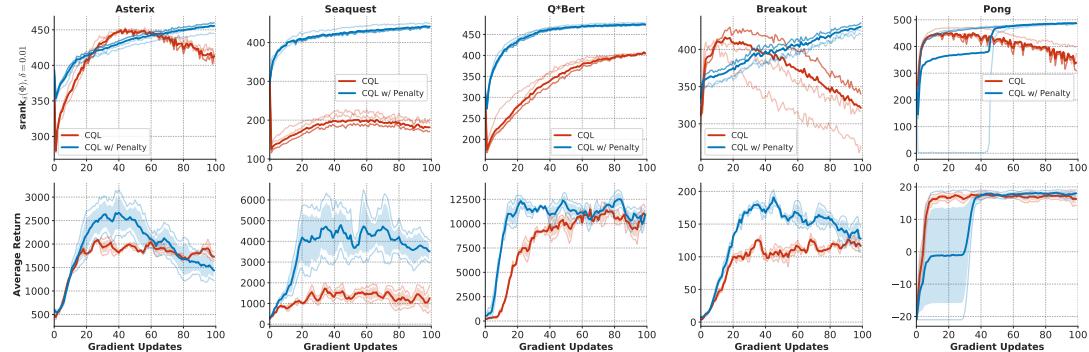


Figure A.17: **Effective rank values with the penalty $L_p(\Phi)$ on CQL.** Trends in effective rank and performance for offline DQN. Note that the performance of CQL with penalty is generally better than vanilla CQL and that the penalty (blue) is effective in increasing the values of effective rank. We report performance at the end of 100 epochs, as per the protocol set by Agarwal et al. (2020) in Figure 10.

A.5.3 OFFLINE RL: PERFORMANCE IMPROVEMENT WITH $L_p(\Phi)$ PENALTY

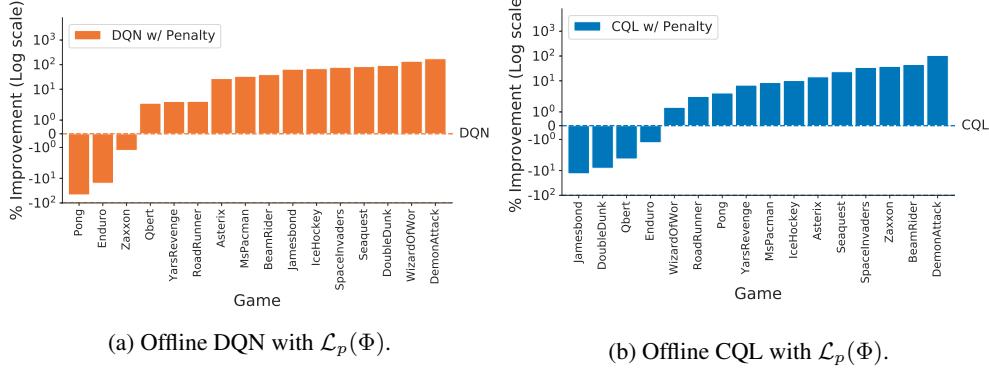


Figure A.18: Performance improvement of (a) offline DQN and (b) offline CQL with the $\mathcal{L}_p(\Phi)$ penalty on 20% Atari dataset, i.e., the dataset referred to as **4x** large in Figure 2.

A.6 DATA-EFFICIENT ONLINE RL: RAINBOW

A.6.1 RAINBOW WITH $L_p(\Phi)$ PENALTY: RANK PLOTS

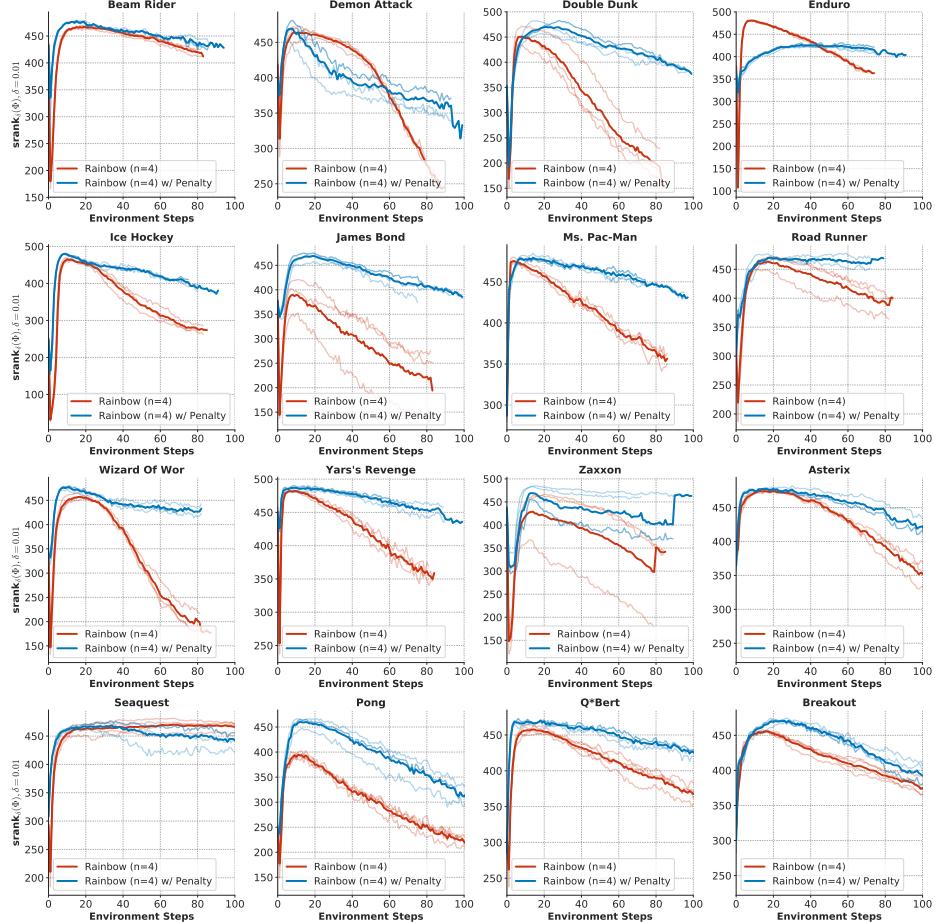


Figure A.19: **Effective rank values with the penalty on Rainbow in the data-efficient online RL setting.** Trends in effective rank and performance for online Rainbow, where distribution shift effects are generally removed. Note that the performance of DQN with penalty is generally better than DQN and that the penalty (blue) is effective in increasing the values of effective rank in most cases. Infact in PONG, where the penalty is not effective in increasing rank, we observe suboptimal performance (blue vs. red).

A.6.2 RAINBOW WITH $L_p(\Phi)$ PENALTY: PERFORMANCE

In this section, we present additional results for supporting the hypothesis that preventing rank-collapse leads to better performance. In the first set of experiments, we apply the proposed \mathcal{L}_p penalty to Rainbow in the data-efficient online RL setting ($n = 4$). In the second set of experiments, we present evidence for prevention of rank collapse by comparing rank values for different runs.

As we will show in the next section, the state-of-the-art Rainbow (Hessel et al., 2018) algorithm also suffers from rank collapse in the data-efficient online RL setting when more updates are performed per gradient step. In this section, we applied our penalty \mathcal{L}_p to Rainbow with $n = 4$, and obtained a median **20.66%** improvement on top of the base method. This result is summarized below.

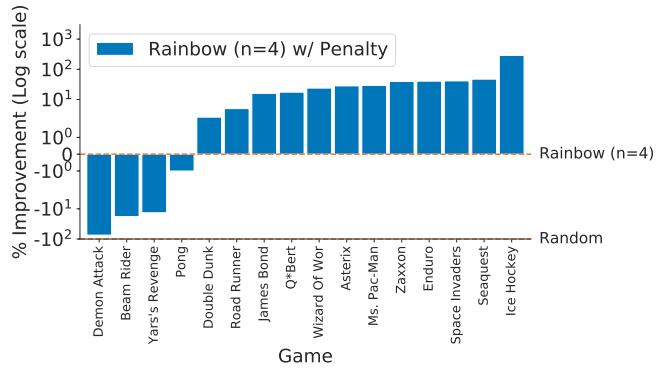


Figure A.20: Performance of Rainbow ($n = 4$) and Rainbow ($n = 4$) with the $\mathcal{L}_p(\Phi)$ penalty (Equation 8). Note that the penalty improves on the base Rainbow in **12/16** games.

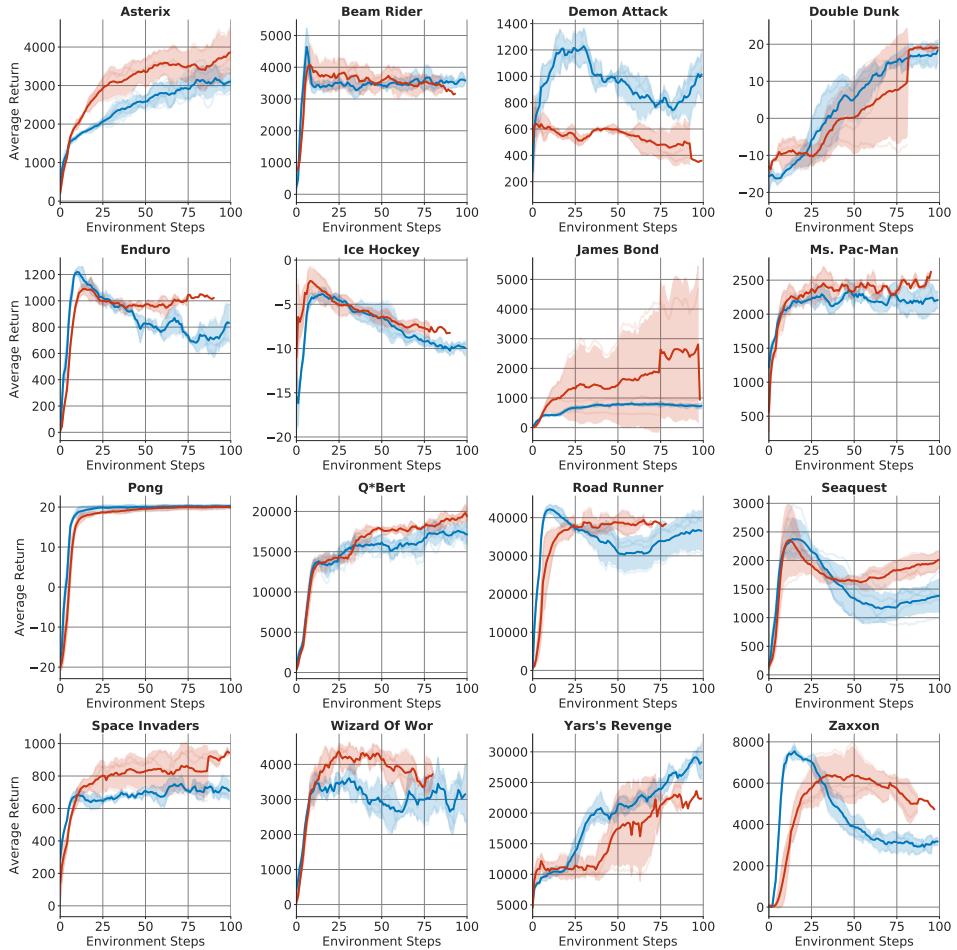


Figure A.21: Learning curves with $n = 4$ gradient updates per environment step for Rainbow (Blue) and Rainbow with the $\mathcal{L}_p(\Phi)$ penalty (Equation 8) (Red) on 16 games, corresponding to the bar plot above. One unit on the x-axis is equivalent to 1M environment steps.

Table B.1: Hyperparameters used by the offline and online RL agents in our experiments.

Hyperparameter	Setting (for both variations)	
Sticky actions	Yes	
Sticky action probability	0.25	
Grey-scaling	True	
Observation down-sampling	(84, 84)	
Frames stacked	4	
Frame skip (Action repetitions)	4	
Reward clipping	[-1, 1]	
Terminal condition	Game Over	
Max frames per episode	108K	
Discount factor	0.99	
Mini-batch size	32	
Target network update period	every 2000 updates	
Training environment steps per iteration	250K	
Update period every	4 environment steps	
Evaluation ϵ	0.001	
Evaluation steps per iteration	125K	
Q -network: channels	32, 64, 64	
Q -network: filter size	$8 \times 8, 4 \times 4, 3 \times 3$	
Q -network: stride	4, 2, 1	
Q -network: hidden units	512	
Hardware	Tesla P100 GPU	
Hyperparameter	Online	Offline
Min replay size for sampling	20,000	-
Training ϵ (for ϵ -greedy exploration)	0.01	-
ϵ -decay schedule	250K steps	-
Fixed Replay Memory	No	Yes
Replay Memory size (Online)	1,000,000 steps	-
Fixed Replay size (5%)	-	2,500,000 steps
Fixed Replay size (20%)	-	10,000,000 steps
Replay Scheme	Uniform	Uniform
Training Iterations	200	500

B HYPERPARAMETERS & EXPERIMENT DETAILS

B.1 ATARI EXPERIMENTS

We follow the experiment protocol from [Agarwal et al. \(2020\)](#) for all our experiments including hyperparameters and agent architectures provided in Dopamine and report them for completeness and ease of reproducibility in Table B.1. We only use hyperparameter selection over the regularization experiment α_p based on results from 5 Atari games (Asterix, Seaquest, Pong, Breakout and Seaquest). We will also open source our code to further aid in reproducing our results.

Evaluation Protocol. Following [Agarwal et al. \(2020\)](#), the Atari environments used in our experiments are stochastic due to sticky actions, *i.e.*, there is 25% chance at every time step that the environment will execute the agent’s previous action again, instead of the agent’s new action. All agents (online or offline) are compared using the best evaluation score (averaged over 5 runs) achieved during training where the evaluation is done online every training iteration using a ϵ -greedy policy with $\epsilon = 0.001$. We report offline training results with same hyperparameters over 5 random seeds of the DQN replay data collection, game simulator and network initialization.

Offline Dataset. As suggested by [Agarwal et al. \(2020\)](#), we randomly subsample the DQN Replay dataset containing 50 millions transitions to create smaller offline datasets with the same data distribution as the original dataset. We use the 5% DQN replay dataset for most of our experiments. We also report results using the 20% dataset setting (4x larger) to show that our claims are also valid even when we have higher coverage over the state space.

Optimizer related hyperparameters. For existing off-policy agents, step size and optimizer were taken as published. We used the DQN (Adam) algorithm for all our experiments, given its superior performance over the DQN (Nature) which uses RMSProp, as reported by Agarwal et al. (2020).

Rainbow agent. Our empirical investigations in this paper are based on the Dopamine Rainbow agent (Castro et al., 2018). This is an open source implementation of the original agent (Hessel et al., 2018), but makes several simplifying design choices. The original agent augments DQN through the use of (a) a distributional learning objective, (b) multi-step returns, (c) the Adam optimizer, (d) prioritized replay, (e) double Q-learning, (f) duelling architecture, and (g) noisy networks for exploration. The Dopamine Rainbow agent uses just the first four of these adjustments, which were identified as the most important aspects of the agent in the original analysis of Hessel et al. (2018). For data efficient Rainbow, we incorporate the main changes suggested by van Hasselt et al. (2019) including the change of architecture, larger number of updates per environment step, and use of multi-step returns.

Atari 2600 games used. For all our experiments in Section 3, we used the same set of 5 games as utilized by Agarwal et al. (2020); Bellemare et al. (2017) to present analytical results. For our empirical evaluation in Appendix A.5, we use the set of games employed by Fedus et al. (2020b) which are deemed suitable for offline RL by Gulcehre et al. (2020). Similar in spirit to Gulcehre et al. (2020), we use the set of 5 games used for analysis for hyperparameter tuning for offline RL methods.

5 games subset: ASTERIX, QBERT, PONG, SEAQUEST, BREAKOUT

16 game subset: In addition to 5 games above, the following 11 games: GRAVITAR, JAMES BOND, MS. PACMAN, SPACE INVADERS, ZAXXON, WIZARD OF WOR, YARS’ REVENGE, ENDURO, ROAD RUNNER, BEAMRIDER, DEMON ATTACK

B.2 GRIDWORLD EXPERIMENTS

We use the gridworld suite from Fu et al. (2019) to obtain gridworlds for our experiments. All of our gridworld results are computed using the 16×16 GRID16SMOOTH0BS environment, which consists of a 256-cell grid, with walls arising randomly with a probability of 0.2. Each state allows 5 different actions (subject to hitting the boundary of the grid): move left, move right, move up, move down and no op. The goal in this environment is to minimize the cumulative discounted distance to a fixed goal location where the discount factor is given by $\gamma = 0.95$. The features for this Q-function are given by randomly chosen vectors which are smoothed spatially in a local neighborhood of a grid cell (x, y) .

We use a deep Q-network with two hidden layers of size (64, 64), and train it using soft Q-learning with entropy coefficient of 0.1, following the code provided by authors of Fu et al. (2019). We use a first-in-first out replay buffer of size 10000 to store past transitions.

C PROOFS FOR SECTION 4.1

In this section, we provide the technical proofs from Section 4.1. We first derive a solution to optimization problem Equation 1 and show that it indeed comes out to have the form described in Equation 3. We first introduce some notation, including definition of the kernel \mathbf{G} which was used for this proof. This proof closely follows the proof from Mabahi et al. (2020).

Definitions. For any universal kernel u , the Green’s function (Duffy, 2015) of the linear kernel operator L given by: $[L\mathbf{Q}](\mathbf{s}, \mathbf{a}) := \sum_{(\mathbf{s}', \mathbf{a}')} u((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}')) \mathbf{Q}(\mathbf{s}', \mathbf{a}')$ is given by the function $g((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}'))$ that satisfies:

$$\sum_{(\mathbf{s}, \mathbf{a})} u((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}')) g((\mathbf{s}', \mathbf{a}'), (\bar{\mathbf{s}}, \bar{\mathbf{a}})) = \delta((\mathbf{s}, \mathbf{a}) - (\bar{\mathbf{s}}, \bar{\mathbf{a}})), \quad (\text{C.1})$$

where δ is the Dirac-delta function. Thus, Green’s function can be understood as a kernel that “inverts” the universal kernel u to the identity (Dirac-delta) matrix. We can then define the matrix \mathbf{G} as the matrix of vectors $\mathbf{g}_{(\mathbf{s}, \mathbf{a})}$ evaluated on the training dataset, \mathcal{D} , however note that the functional $\mathbf{g}_{(\mathbf{s}, \mathbf{a})}$ can be evaluated for other state-action tuples, not present in \mathcal{D} .

$$\mathbf{G}((\mathbf{s}_i, \mathbf{a}_i), (\mathbf{s}_j, \mathbf{a}_j)) := \mathbf{g}((\mathbf{s}_i, \mathbf{a}_i), (\mathbf{s}_j, \mathbf{a}_j)) \quad \text{and} \quad \mathbf{g}_{(\mathbf{s}, \mathbf{a})}[i] = \mathbf{g}((\mathbf{s}, \mathbf{a}), (\mathbf{s}_i, \mathbf{a}_i)) \quad \forall (\mathbf{s}_i, \mathbf{a}_i) \in \mathcal{D}. \quad (\text{C.2})$$

Lemma C.0.1. *The solution to Equation 1 is given by Equation 3.*

Proof. This proof closely follows the proof of Proposition 1 from (Mobahi et al., 2020). We revisit key aspects the key parts of this proof here.

We restate the optimization problem below, and solve for the optimum \mathbf{Q}_k to this equation by applying the functional derivative principle.

$$\min_{\mathbf{Q} \in \mathcal{Q}} J(\mathbf{Q}) := \sum_{\mathbf{s}_i, \mathbf{a}_i \in \mathcal{D}} (\mathbf{Q}(\mathbf{s}_i, \mathbf{a}_i) - y_k(\mathbf{s}_i, \mathbf{a}_i))^2 + c \sum_{(\mathbf{s}, \mathbf{a})} \sum_{(\mathbf{s}', \mathbf{a}')} u((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}')) \mathbf{Q}(\mathbf{s}, \mathbf{a}) \mathbf{Q}(\mathbf{s}', \mathbf{a}').$$

The functional derivative principle would say that the optimal \mathbf{Q}_k to this problem would satisfy, for any other function f and for a small enough $\varepsilon \rightarrow 0$,

$$\forall f \in \mathcal{Q} : \left. \frac{\partial J(\mathbf{Q}_k + \varepsilon f)}{\partial \varepsilon} \right|_{\varepsilon=0} = 0. \quad (\text{C.3})$$

By setting the gradient of the above expression to 0, we obtain the following stationarity conditions on \mathbf{Q}_k (also denoting $(\mathbf{s}_i, \mathbf{a}_i) := \mathbf{x}_i$) for brevity:

$$\sum_{\mathbf{x}_i \in \mathcal{D}} \delta(\mathbf{x} - \mathbf{x}_i) (\mathbf{Q}_k(\mathbf{x}_i) - \mathbf{y}_k(\mathbf{x}_i)) + c \sum_{\mathbf{x}} u(\mathbf{x}, \mathbf{x}') \mathbf{Q}_k(\mathbf{x}') = 0. \quad (\text{C.4})$$

Now, we invoke the definition of the Green's function discussed above and utilize the fact that the Dirac-delta function can be expressed in terms of the Green's function, we obtain a simplified version of the above relation:

$$\sum_{\mathbf{x}} u(\mathbf{x}, \mathbf{x}') \sum_{\mathbf{x}_i \in \mathcal{D}} (\mathbf{Q}_k(\mathbf{x}_i) - \mathbf{y}_k(\mathbf{x}_i)) g(\mathbf{x}', \mathbf{x}_i) = -c \sum_{\mathbf{x}} u(\mathbf{x}, \mathbf{x}') \mathbf{Q}_k(\mathbf{x}'). \quad (\text{C.5})$$

Since the kernel $u(\mathbf{x}, \mathbf{x}')$ os universal and positive definite, the optimal solution $\mathbf{Q}_k(\mathbf{x})$ is given by:

$$\mathbf{Q}_k(\mathbf{s}, \mathbf{a}) = -\frac{1}{c} \sum_{(\mathbf{s}_i, \mathbf{a}_i) \in \mathcal{D}} (\mathbf{Q}_k(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_k(\mathbf{s}_i, \mathbf{a}_i)) \cdot g((\mathbf{s}, \mathbf{a}), (\mathbf{s}_i, \mathbf{a}_i)). \quad (\text{C.6})$$

Finally we can replace the expression for residual error, $\mathbf{Q}_k(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_k(\mathbf{s}_i, \mathbf{a}_i)$ using the green's kernel on the training data by solving for it in closed form, which gives us the solution in Equation 3.

$$\mathbf{Q}_k(\mathbf{s}, \mathbf{a}) = -\frac{1}{c} \mathbf{g}_{(\mathbf{s}, \mathbf{a})}^T (\mathbf{Q}_k - \mathbf{y}_k) = \mathbf{g}_{(\mathbf{s}, \mathbf{a})}^T (c\mathbf{I} + \mathbf{G})^{-1} \mathbf{y}_k. \quad (\text{C.7})$$

□

Next, we now state and prove a slightly stronger version of Theorem 4.1 that immediately implies the original theorem.

Theorem C.1. Let \mathbf{S} be a shorthand for $\mathbf{S} = \gamma P^\pi \mathbf{A}$ and assume \mathbf{S} is a normal matrix. Then there exists an infinite, strictly increasing sequence of fitting iterations, $(k_l)_{l=1}^\infty$ starting from $k_1 = 0$, such that, for any two singular-values $\sigma_i(\mathbf{S})$ and $\sigma_j(\mathbf{S})$ of \mathbf{S} with $\sigma_i(\mathbf{S}) \leq \sigma_j(\mathbf{S})$,

$$\forall l \in \mathbb{N} \text{ and } l' \geq l, \quad \frac{\sigma_i(\mathbf{M}_{k_{l'}})}{\sigma_j(\mathbf{M}_{k_{l'}})} < \frac{\sigma_i(\mathbf{M}_{k_l})}{\sigma_j(\mathbf{M}_{k_l})} \leq \frac{\sigma_i(\mathbf{S})}{\sigma_j(\mathbf{S})}. \quad (\text{C.8})$$

Therefore, the effective rank of \mathbf{M}_k satisfies: $\text{srank}_\delta(\mathbf{M}_{k_{l'}}) \leq \text{srank}_\delta(\mathbf{M}_{k_l})$. Furthermore,

$$\forall l \in \mathbb{N} \text{ and } t \geq k_l, \quad \frac{\sigma_i(\mathbf{M}_t)}{\sigma_j(\mathbf{M}_t)} < \frac{\sigma_i(\mathbf{M}_{k_l})}{\sigma_j(\mathbf{M}_{k_l})} + \mathcal{O}\left(\left(\frac{\sigma_i(\mathbf{S})}{\sigma_j(\mathbf{S})}\right)^{k_l}\right). \quad (\text{C.9})$$

Therefore, the effective rank of \mathbf{M}_t , $\text{srank}_\delta(\mathbf{M}_t)$, outside the chosen subsequence is also controlled above by the effective rank on the subsequence $(\text{srank}_\delta(\mathbf{M}_{k_l}))_{l=1}^\infty$.

To prove this theorem, we first show that for any two fitting iterations, $t < t'$, if \mathbf{S}^t and $\mathbf{S}^{t'}$ are positive semi-definite, the ratio of singular values and the effective rank decreases from t to t' . As an immediate consequence, this shows that when \mathbf{S} is positive semi-definite, the effective rank decreases at every iteration, i.e., by setting $k_l = l$ (Corollary C.1.1).

To extend the proof to arbitrary normal matrices, we show that for any \mathbf{S} , a sequence of fitting iterations $(k_l)_{l=1}^\infty$ can be chosen such that \mathbf{S}^{k_l} is (approximately) positive semi-definite. For this subsequence of fitting iterations, the ratio of singular values and effective rank also decreases. Finally, to control the ratio and effective rank on fitting iterations t outside this subsequence, we construct an upper bound on the ratio $f(t)$: $\frac{\sigma_i(\mathbf{M}_t)}{\sigma_j(\mathbf{M}_t)} < f(t)$, and relate this bound to the ratio of singular values on the chosen subsequence.

Lemma C.1.1 ($\text{srank}_\delta(\mathbf{M}_k)$ decreases when \mathbf{S}^k is PSD.). Let \mathbf{S} be a shorthand for $\mathbf{S} = \gamma P^\pi \mathbf{A}$ and assume \mathbf{S} is a normal matrix. Choose any $t, t' \in \mathbb{N}$ such that $t < t'$. If \mathbf{S}^t and $\mathbf{S}^{t'}$ are positive semi-definite, then for any two singular-values $\sigma_i(\mathbf{S})$ and $\sigma_j(\mathbf{S})$ of \mathbf{S} , such that $0 < \sigma_i(\mathbf{S}) < \sigma_j(\mathbf{S})$:

$$\frac{\sigma_i(\mathbf{M}_{t'})}{\sigma_j(\mathbf{M}_{t'})} < \frac{\sigma_i(\mathbf{M}_t)}{\sigma_j(\mathbf{M}_t)} \leq \frac{\sigma_i(\mathbf{S})}{\sigma_j(\mathbf{S})}. \quad (\text{C.10})$$

Hence, the effective rank of \mathbf{M}_k decreases from t to t' : $\text{srank}_\delta(\mathbf{M}_{t'}) \leq \text{srank}_\delta(\mathbf{M}_t)$.

Proof. First note that \mathbf{M}_k is given by:

$$\mathbf{M}_k := \sum_{i=1}^k \gamma^{k-i} (P^\pi \mathbf{A})^{k-1-i} = \gamma \sum_{i=1}^{k-1} \mathbf{S}^{k-i-1}. \quad (\text{C.11})$$

From hereon, we omit the leading γ term since it is a constant scaling factor that does not affect ratio or effective rank. Since \mathbf{S} is normal, \mathbf{S} admits a complex orthogonal eigendecomposition, where the eigenvalues and singular values are related as $\sigma_k(S) = |\lambda_k(S)|$. Thus, we can write $\mathbf{S} := \mathbf{U}\lambda(\mathbf{S})\mathbf{U}^H$. And any power of \mathbf{S} , i.e., \mathbf{S}^i can be expressed as: $\mathbf{S}^i = \mathbf{U}\lambda(\mathbf{S})^i\mathbf{U}^H$, and hence, we can express \mathbf{M}_k as:

$$\mathbf{M}_k := \mathbf{U} \left(\sum_{i=1}^k \lambda(\mathbf{S})^i \right) \mathbf{U}^H = \mathbf{U} \cdot \mathbf{diag} \left(\frac{1 - \lambda(\mathbf{S})^k}{1 - \lambda(\mathbf{S})} \right) \cdot \mathbf{U}^H. \quad (\text{C.12})$$

Then, the singular values of \mathbf{M}_k can be expressed as

$$\sigma_i(\mathbf{M}_k) := \left| \frac{1 - \lambda_i(\mathbf{S})^k}{1 - \lambda_i(\mathbf{S})} \right|, \quad (\text{C.13})$$

When \mathbf{S}^k is positive semi-definite, $\lambda_i(\mathbf{S})^k = \sigma_i(\mathbf{S})^k$, enabling the following simplification:

$$\sigma_i(\mathbf{M}_k) = \frac{|1 - \sigma_i(\mathbf{S})^k|}{|1 - \lambda_i(\mathbf{S})|}. \quad (\text{C.14})$$

To show that the ratio of singular values decreases from t to t' , we need to show that $f(\sigma) = \frac{|1 - \sigma^{t'}|}{|1 - \sigma^t|}$ is an increasing function of σ when $t' > t$. It can be seen that this is the case, which implies the desired result.

To further show that $\text{srank}_\delta(\mathbf{M}_t) \geq \text{srank}_\delta(\mathbf{M}_{t'})$, we can simply show that $\forall i \in [1, \dots, n]$, $h_k(i) := \frac{\sum_{j=1}^i \sigma_j(\mathbf{M}_k)}{\sum_{j=1}^n \sigma_j(\mathbf{M}_k)}$ increases with k , and this would imply that the $\text{srank}_\delta(\mathbf{M}_k)$ cannot increase from $k = t$ to $k = t'$. We can decompose $h_k(i)$ as:

$$h_k(i) = \sum_{j=1}^i \frac{\sigma_j(\mathbf{M}_k)}{\sum_l \sigma_l(\mathbf{M}_k)} = \frac{1}{1 + \frac{\sum_{j=i+1}^n \sigma_j(\mathbf{M}_k)}{\sum_{j=1}^i \sigma_j(\mathbf{M}_k)}}. \quad (\text{C.15})$$

Since $\sigma_j(\mathbf{M}_k)/\sigma_l(\mathbf{M}_i)$ decreases over time k for all j, l if $\sigma_j(\mathbf{S}) \leq \sigma_l(\mathbf{S})$, the ratio in the denominator of $h_k(i)$ decreases with increasing k implying that $h_k(i)$ increases from t to t' . \square

Corollary C.1.1 ($\text{srank}_\delta(\mathbf{M}_k)$ decreases for PSD \mathbf{S} matrices.). Let \mathbf{S} be a shorthand for $\mathbf{S} = \gamma P^\pi \mathbf{A}$. Assuming that \mathbf{S} is positive semi-definite, for any $k, t \in \mathbb{N}$, such that $t > k$ and that for any two singular-values $\sigma_i(\mathbf{S})$ and $\sigma_j(\mathbf{S})$ of \mathbf{S} , such that $\sigma_i(\mathbf{S}) < \sigma_j(\mathbf{S})$,

$$\frac{\sigma_i(\mathbf{M}_t)}{\sigma_j(\mathbf{M}_t)} < \frac{\sigma_i(\mathbf{M}_k)}{\sigma_j(\mathbf{M}_k)} \leq \frac{\sigma_i(\mathbf{S})}{\sigma_j(\mathbf{S})}. \quad (\text{C.16})$$

Hence, the effective rank of \mathbf{M}_k decreases with more fitting iterations: $\text{srank}_\delta(\mathbf{M}_t) \leq \text{srank}_\delta(\mathbf{M}_k)$.

In order to now extend the result to arbitrary normal matrices, we must construct a subsequence of fitting iterations $(k_l)_{l=1}^\infty$ where \mathbf{S}^{k_l} is (approximately) positive semi-definite. To do so, we first prove a technical lemma that shows that rational numbers, i.e., numbers that can be expressed as $r = \frac{p}{q}$, for integers $p, q \in \mathbb{Z}$ are “dense” in the space of real numbers.

Lemma C.1.2 (Rational numbers are dense in the real space.). *For any real number α , there exist infinitely many rational numbers $\frac{p}{q}$ such that α can be approximated by $\frac{p}{q}$ upto $\frac{1}{q^2}$ accuracy.*

$$\left| \alpha - \frac{p}{q} \right| \leq \frac{1}{q^2}. \quad (\text{C.17})$$

Proof. We first use Dirichlet's approximation theorem (see Hlawka et al. (1991) for a proof of this result using a pigeonhole argument and extensions) to obtain that for any real numbers α and $N \geq 1$, there exist integers p and q such that $1 \leq q \leq N$ and,

$$|q\alpha - p| \leq \frac{1}{|N| + 1} < \frac{1}{N}. \quad (\text{C.18})$$

Now, since $q \geq 1 > 0$, we can divide both sides by q , to obtain:

$$\left| \alpha - \frac{p}{q} \right| \leq \frac{1}{Nq} \leq \frac{1}{q^2}. \quad (\text{C.19})$$

To obtain infinitely many choices for $\frac{p}{q}$, we observe that Dirichlet's lemma is valid only for all values of N that satisfy $N \leq \frac{1}{|q\alpha - p|}$. Thus if we choose an N' such that $N' \geq N_{\max}$ where N_{\max} is defined as:

$$N_{\max} = \max \left\{ \frac{1}{|q'\alpha - p'|} \mid p', q' \in \mathbb{Z}, 1 \leq q' \leq q \right\}. \quad (\text{C.20})$$

Equation C.20 essentially finds a new value of N , such that the current choices of p and q , which were valid for the first value of N do not satisfy the approximation error bound. Applying Dirichlet's lemma to this new value of N' hence gives us a new set of p' and q' which satisfy the $\frac{1}{q'^2}$ approximation error bound. Repeating this process gives us countably many choices of (p, q) pairs that satisfy the approximation error bound. As a result, rational numbers are dense in the space of real numbers, since for any arbitrarily chosen approximation accuracy given by $\frac{1}{q^2}$, we can obtain atleast one rational number, $\frac{p}{q}$ which is closer to α than $\frac{1}{q^2}$. This proof is based on Johnson (2016). \square

Now we utilize Lemmas C.1.1 and C.1.2 to prove Proposition 4.1.

Proof of Proposition 4.1 and Theorem C.1 Recall from the proof of Lemma C.1.1 that the singular values of \mathbf{M}_k are given by:

$$\sigma_i(\mathbf{M}_k) := \left| \frac{1 - \lambda_i(\mathbf{S})^k}{1 - \lambda_i(\mathbf{S})} \right|, \quad (\text{C.21})$$

Bound on Singular Value Ratio: The ratio between $\sigma_i(\mathbf{M}_k)$ and $\sigma_j(\mathbf{M}_k)$ can be expressed as

$$\frac{\sigma_i(\mathbf{M}_k)}{\sigma_j(\mathbf{M}_k)} = \left| \frac{1 - \lambda_i(\mathbf{S})^k}{1 - \lambda_j(\mathbf{S})^k} \right| \left| \frac{1 - \lambda_j(\mathbf{S})}{1 - \lambda_i(\mathbf{S})} \right|. \quad (\text{C.22})$$

For a normal matrix \mathbf{S} , $\sigma_i(\mathbf{S}) = |\lambda_i(\mathbf{S})|$, so this ratio can be bounded above as

$$\frac{\sigma_i(\mathbf{M}_k)}{\sigma_j(\mathbf{M}_k)} \leq \frac{1 + \sigma_i(\mathbf{S})^k}{|1 - \sigma_j(\mathbf{S})^k|} \left| \frac{1 - \lambda_j(\mathbf{S})}{1 - \lambda_i(\mathbf{S})} \right|. \quad (\text{C.23})$$

Defining $f(k)$ to be the right hand side of the equation, we can verify that f is a monotonically decreasing function in k when $\sigma_i < \sigma_j$. This shows that this ratio of singular values is bounded above and in general, must decrease towards some limit $\lim_{k \rightarrow \infty} f(k)$.

Construction of Subsequence: We now show that there exists a subsequence $(k_l)_{l=1}^{\infty}$ for which \mathbf{S}^{k_l} is approximately positive semi-definite. For ease of notation, let's represent the i -th eigenvalue as $\lambda_i(\mathbf{S}) = |\lambda_i(\mathbf{S})| \cdot e^{i\theta_i}$, where $\theta_i > 0$ is the polar angle of the complex value $\lambda_i(\mathbf{S})$ and $|\lambda_i(\mathbf{S})|$ is its magnitude (norm). Now, using Lemma C.1.2, we can approximate any polar angle, θ_i using a rational approximation, i.e., we apply lemma C.1.2 on $\frac{\theta_i}{2\pi}$

$$\exists p_i, q_i \in \mathbb{N}, \text{ s.t. } \left| \frac{\theta_i}{2\pi} - \frac{p_i}{q_i} \right| \leq \frac{1}{q_i^2}. \quad (\text{C.24})$$

Since the choice of q_i is within our control we can estimate θ_i for all eigenvalues $\lambda_i(\mathbf{S})$ to infinitesimal accuracy. Hence, we can approximate $\theta_i \approx 2\pi \frac{p_i}{q_i}$. We will now use this approximation to construct an infinite sequence $(k_l)_{l=1}^\infty$, shown below:

$$k_l = l \cdot \text{LCM}(q_1, \dots, q_n) \quad \forall j \in \mathbb{N}, \quad (\text{C.25})$$

where LCM is the least-common-multiple of natural numbers q_1, \dots, q_n .

In the absence of any approximation error in θ_i , we note that for any i and for any $l \in \mathbb{N}$ as defined above, $\lambda_i(\mathbf{S})^{k_l} = |\lambda_i(\mathbf{S})|^{k_l} \cdot \exp\left(2i\pi \cdot \frac{p_i}{q_i} \cdot k_l\right) = |\lambda_i(\mathbf{S})|^{k_l}$, since the polar angle for any k_l is going to be a multiple of 2π , and hence it would fall on the real line. As a result, \mathbf{S}^{k_l} will be positive semi-definite, since all eigenvalues are positive and real. Now by using the proof for Lemma C.1.1, we obtain the ratio of i and j singular values are increasing over the sequence of iterations $(k_j)_{j=1}^\infty$. Since the approximation error in θ_i can be controlled to be infinitesimally small to prevent any increase in the value of srank_δ due to it (this can be done given the discrete form of srank_δ), we note that the above argument applies even with the approximation, proving the required result on the subsequence.

Controlling All Fitting Iterations using Subsequence:

We now relate the ratio of singular values within this chosen subsequence to the ratio of singular values elsewhere. Choose $t, l \in \mathbb{N}$ such that $t > k_l$. Earlier in this proof, we showed that the ratio between singular values is bounded above by a monotonically decreasing function $f(t)$, so

$$\frac{\sigma_i(\mathbf{M}_t)}{\sigma_j(\mathbf{M}_t)} \leq f(t) < f(k_l). \quad (\text{C.26})$$

Now, we show that that $f(k_l)$ is in fact very close to the ratio of singular values:

$$f(k_l) = \frac{|1 - \sigma_i(\mathbf{S})^{k_l}|}{|1 - \sigma_j(\mathbf{S})^{k_l}|} \left| \frac{1 - \lambda_j(\mathbf{S})}{1 - \lambda_i(\mathbf{S})} \right| \leq \frac{\sigma_i(\mathbf{M}_t)}{\sigma_j(\mathbf{M}_t)} + \frac{2\sigma_i(\mathbf{S})^{k_l}}{|1 - \sigma_j(\mathbf{S})^{k_l}|} \left| \frac{1 - \lambda_j(\mathbf{S})}{1 - \lambda_i(\mathbf{S})} \right|. \quad (\text{C.27})$$

The second term goes to zero as k_l increases; algebraic manipulation shows that this gap be bounded by

$$f(k_l) \leq \frac{\sigma_i(\mathbf{M}_{k_l})}{\sigma_j(\mathbf{M}_{k_l})} + \underbrace{\left(\frac{\sigma_i(S)}{\sigma_j(S)} \right)^{k_l} \frac{2\sigma_j(\mathbf{S})}{|1 - \sigma_j(\mathbf{S})|} \left| \frac{1 - \lambda_j(\mathbf{S})}{1 - \lambda_i(\mathbf{S})} \right|}_{\text{constant}}. \quad (\text{C.28})$$

Putting these inequalities together proves the final statement,

$$\frac{\sigma_i(\mathbf{M}_t)}{\sigma_j(\mathbf{M}_t)} \leq \frac{\sigma_i(\mathbf{M}_{k_l})}{\sigma_j(\mathbf{M}_{k_l})} + \mathcal{O}\left(\left(\frac{\sigma_i(S)}{\sigma_j(S)}\right)^{k_l}\right). \quad (\text{C.29})$$

D PROOFS FOR SECTION 4.2

In this section, we provide technical proofs from Section 4.2. We start by deriving properties of optimization trajectories of the weight matrices of the deep linear network similar to Arora et al. (2018) but customized to our set of assumptions, then prove Proposition 4.1, and finally discuss how to extend these results to the fitted Q-iteration setting and some extensions not discussed in the main paper. Similar to Section 4.1, we assume access to a dataset of transitions, $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r(\mathbf{s}_i, \mathbf{a}_i), \mathbf{s}'_i)\}$ in this section, and assume that the same data is used to re-train the function.

Notation and Definitions. The Q-function is represented using a deep linear network with at least 3 layers, such that

$$Q(\mathbf{s}, \mathbf{a}) = \mathbf{W}_N \mathbf{W}_{N-1} \cdots \mathbf{W}_1[\mathbf{s}; \mathbf{a}], \quad \text{where } N \geq 3, \mathbf{W}_N \in \mathbb{R}^{1 \times d_{N-1}}, \quad (\text{D.1})$$

and $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$ for $i = 1, \dots, N-1$. We index the weight matrices by a tuple (k, t) : $\mathbf{W}_j(k, t)$ denotes the weight matrix \mathbf{W}_j at the t -th step of gradient descent during the k -th fitting iteration (Algorithm 1). Let the end-to-end weight matrix $\mathbf{W}_N \mathbf{W}_{N-1} \cdots \mathbf{W}_1$ be denoted shorthand as $\mathbf{W}_{N:1}$, and let the features of the penultimate layer of the network, be denoted as $\mathbf{W}_\phi(k, t) :=$

$\mathbf{W}_{N-1}(k, t) \cdots \mathbf{W}_1(k, t)$. $\mathbf{W}_\phi(k, t)$ is the matrix that maps an input $[\mathbf{s}; \mathbf{a}]$ to corresponding features $\Phi(\mathbf{s}, \mathbf{a})$. In our analysis, it is sufficient to consider the effective rank of $\mathbf{W}_\phi(k, t)$ since the features Φ are given by: $\Phi(k, t) = \mathbf{W}_\phi(k, t)[\mathcal{S}; \mathcal{A}]$, which indicates that:

$$\text{rank}(\Phi(k, t)) = \text{rank}(\mathbf{W}_\phi(k, t)[\mathcal{S}; \mathcal{A}]) \leq \min(\text{rank}(\mathbf{W}_\phi(k, t)), \text{rank}([\mathcal{S}; \mathcal{A}])).$$

Assuming the state-action space has full rank, we are only concerned about $\text{rank}(\mathbf{W}_\phi(k, t))$ which justifies our choice for analyzing $\text{srank}_\delta(\mathbf{W}_\phi(k, t))$.

Let $L_{k+1}(\mathbf{W}_{N:1}(k, t))$ denote the mean squared Bellman error optimization objective in the k -th fitting iteration.

$$L_{k+1}(\mathbf{W}_{N:1}(k, t)) = \sum_{i=1}^{|\mathcal{D}|} (\mathbf{W}_N(k, t)\mathbf{W}_\phi(k, t)[\mathbf{s}_i; \mathbf{a}_i] - \mathbf{y}_k(\mathbf{s}_i, \mathbf{a}_i))^2, \text{ where } \mathbf{y}_k = \mathbf{R} + \gamma P^\pi \mathbf{Q}_k.$$

When gradient descent is used to update the weight matrix, the updates to $\mathbf{W}_i(k, t)$ are given by:

$$\mathbf{W}_j(k, t+1) \leftarrow \mathbf{W}_j(k, t) - \eta \frac{\partial L_{k+1}(\mathbf{W}_{N:1}(k, t))}{\partial \mathbf{W}_j(k, t)}.$$

If the learning rate η is small, we can approximate this discrete time process with a continuous-time differential equation, which we will use for our analysis. We use $\dot{W}(k, t)$ to denote the derivative of $W(k, t)$ with respect to t , for a given k .

$$\dot{\mathbf{W}}_j(k, t) = -\eta \frac{\partial L_{k+1}(\mathbf{W}_{N:1}(k, t))}{\partial \mathbf{W}_j(k, t)} \quad (\text{D.2})$$

In order to quantify the evolution of singular values of the weight matrix, $\mathbf{W}_\phi(k, t)$, we start by quantifying the evolution of the weight matrix $\mathbf{W}_\phi(k, t)$ using a more interpretable differential equation. In order to do so, we make an assumption similar to but not identical as Arora et al. (2018), that assumes that all except the last weight matrix are “balanced” at initialization $t = 0, k = 0$. i.e.

$$\forall i \in [0, \dots, N-2] : \mathbf{W}_{i+1}^T(0, 0)\mathbf{W}_{i+1}(0, 0) = \mathbf{W}_i(0, 0)\mathbf{w}_i(0, 0)^T. \quad (\text{D.3})$$

Note the distinction from Arora et al. (2018), the last layer is not assumed to be balanced. As a result, we may not be able to comment about the learning dynamics of the end-to-end weight matrix, but we prevent the vacuous case where all the weight matrices are rank 1. Now we are ready to derive the evolution of the feature matrix, $\mathbf{W}_\phi(k, t)$.

Lemma D.0.1 (Adaptation of balanced weights (Arora et al., 2018) across FQI iterations). *Assume the weight matrices evolve according to Equation D.2, with respect to L_k for all fitting iterations k . Assume balanced initialization only for the first $N-1$ layers, i.e., $\mathbf{W}_{j+1}(0, 0)^T\mathbf{W}_{j+1}(0, 0) = \mathbf{W}_j(0, 0)\mathbf{W}_j(0, 0)^T, \forall j \in 1, \dots, N-2$. Then the weights remain balanced throughout, i.e.*

$$\forall k, t \quad \mathbf{W}_{j+1}(k, t)^T\mathbf{W}_{j+1}(k, t) = \mathbf{W}_j(k, t)\mathbf{W}_j(k, t)^T, \forall j \in 1, \dots, N-2. \quad (\text{D.4})$$

Proof. First consider the special case of $k = 0$. To begin with, in order to show that weights remain balanced throughout training in $k = 0$ iteration, we will follow the proof technique in Arora et al. (2018), with some modifications. First note that the expression for $\frac{\partial L_{k+1}(\mathbf{W}_{N:1}(k, t))}{\partial \mathbf{W}_j(k, t)}$ can be expressed as:

$$\begin{aligned} \frac{\partial L_{k+1}(\mathbf{W}_{N:1}(k, t))}{\partial \mathbf{W}_j(k, t)} &= \left(\prod_{i=j+1}^N \mathbf{W}_i^T \right) \cdot \frac{dL_0(\mathbf{W}_{N:1})}{d\mathbf{W}_{N:1}} \cdot \left(\prod_{i=1}^{j-1} \mathbf{W}_i^T \right) \\ &= (\mathbf{W}_{j+1}^T \mathbf{W}_{j+2}^T \cdots \mathbf{W}_N^T) \cdot \frac{dL_0(\mathbf{W}_{N:1})}{d\mathbf{W}_{N:1}} \cdot \left(\prod_{i=1}^{j-1} \mathbf{W}_i^T \right). \end{aligned}$$

Now, since the weight matrices evolve as per Equation D.2, by multiplying the similar differential equation for \mathbf{W}_j with $\mathbf{W}_j^T(k, t)$ on the right and multiplying evolution of \mathbf{W}_{j+1} with $\mathbf{W}_{j+1}^T(k, t)$ from the left, and adding the two equations, we obtain:

$$\forall j \in [0, \dots, N-2] : \mathbf{W}_{j+1}^T(0, t)\dot{\mathbf{W}}_{j+1}(0, t) = \dot{\mathbf{W}}_j(0, t)\mathbf{W}_j^T(0, t). \quad (\text{D.5})$$

We can then take transpose of the equation above, and add it to itself, to obtain an easily integrable expression:

$$\begin{aligned} \frac{d \mathbf{W}_{j+1}(0, t) \mathbf{W}_{j+1}(0, t)^T}{dt} &= \mathbf{W}_{j+1}^T(0, t) \dot{\mathbf{W}}_{j+1}(0, t) + \dot{\mathbf{W}}_{j+1}(0, t) \mathbf{W}_{j+1}^T(0, t) = \\ &\dot{\mathbf{W}}_j(0, t) \mathbf{W}_j^T(0, t) + \mathbf{W}_j(0, t) \dot{\mathbf{W}}_j^T(0, t) = \frac{d \mathbf{W}_j^T(0, t) \mathbf{W}_j(0, t)}{dt}. \end{aligned} \quad (\text{D.6})$$

Since we have assumed the balancedness condition at the initial timestep 0, and the derivatives of the two quantities are equal, their integral will also be the same, hence we obtain:

$$\mathbf{W}_{j+1}^T(0, t) \mathbf{W}_{j+1}(0, t) = \mathbf{W}_j(0, t) \mathbf{W}_j^T(0, t). \quad (\text{D.7})$$

Now, since the weights after T iterations in fitting iteration $k = 0$ are still balanced, the initialization for $k = 1$ is balanced. Note that since the balancedness property does not depend on which objective gradient is used to optimize the weights, as long as \mathbf{W}_j and \mathbf{W}_{j+1} utilize the same gradient of the loss function. Formalizing this, we can show inductively that the weights will remain balanced across all fitting iterations k and at all steps t within each fitting iteration. Thus, we have shown the result in Equation D.4. \square

Our next result aims at deriving the evolution of the feature-matrix that under the balancedness condition. We will show that the feature matrix, $\mathbf{W}_\phi(k, t)$ evolves according to a similar, but distinct differential equation as the end-to-end weight matrix, $\mathbf{W}_{N:1}(k, t)$, which still allows us to appeal to techniques and results from Arora et al. (2019) to study properties of the singular value evolution and hence, discuss properties related to the effective rank of the matrix, $\mathbf{W}_\phi(k, t)$.

Lemma D.0.2 ((Extension of Theorem 1 from Arora et al. (2018)). *Under conditions specified in Lemma D.0.1, the feature matrix, $\mathbf{W}_\phi(k, t)$ evolves as per the following continuous-time differential equation, for all fitting iterations k :*

$$\dot{\mathbf{W}}_\phi(k, t) = -\eta \sum_{j=1}^{N-1} [\mathbf{W}_\phi(k, t) \mathbf{W}_\phi(k, t)^T]^{\frac{N-j}{N-1}} \cdot \mathbf{W}_N(k, t)^T \frac{dL_k(\mathbf{W}_{N:1}(k, t))}{d\mathbf{W}_{N:1}} \cdot [\mathbf{W}_\phi(k, t)^T \mathbf{W}_\phi(k, t)]^{\frac{j-1}{N-1}}.$$

Proof. In order to prove this statement, we utilize the fact that the weights upto layer $N - 2$ are balanced throughout training. Now consider the singular value decomposition of any weight \mathbf{w}_j (unless otherwise states, we use \mathbf{W}_j to refer to $\mathbf{W}_j(k, t)$ in this section, for ease of notation). $\mathbf{W}_j = \mathbf{U}_j \Sigma_j \mathbf{V}_j^T$. The balancedness condition re-written using SVD of the weight matrices is equivalent to

$$\mathbf{V}_{j+1} \Sigma_{j+1}^T \Sigma_{j+1} \mathbf{V}_{j+1}^T = \mathbf{U}_j \Sigma_j \Sigma_j^T \mathbf{U}_j^T. \quad (\text{D.8})$$

Thus for all j on which the balancedness condition is valid, it must hold that $\Sigma_{j+1}^T \Sigma_{j+1} = \Sigma_j \Sigma_j^T$, since these are both the eigendecompositions of the same matrix (as they are equal). As a result, the weight matrices \mathbf{W}_j and \mathbf{W}_{j+1} share the same singular value space which can be written as $\rho_1 \geq \rho_2 \geq \dots \geq \rho_m$. The ordering of eigenvalues can be different, and the matrices \mathbf{U} and \mathbf{V} can also be different (and be rotations of one other) but the unique values that the singular values would take are the same. Note the distinction from Arora et al. (2018), where they apply balancedness on all matrices, and that in our case would trivially give a rank-1 matrix.

Now this implies, that we can express the feature matrix, also in terms of the common singular values, $(\rho_1, \rho_2, \dots, \rho_m)$, for example, as $\mathbf{W}_j(k, t) = \mathbf{U}_{j+1} \text{Diag}(\sqrt{\rho_1}, \dots, \sqrt{\rho_m}) \mathbf{V}_j^T$, where $\mathbf{U}_j = \mathbf{V}_{j+1} \mathbf{O}_j$, where \mathbf{O}_j is an orthonormal matrix. Using this relationship, we can say the following:

$$\begin{aligned} \prod_{i=j}^{N-1} \mathbf{W}_i(k, t) \prod_{i=j}^{N-1} \mathbf{W}_i(k, t)^T &= [\mathbf{W}_\phi(k, t) \mathbf{W}_\phi^T(k, t)]^{\frac{N-j}{N-1}} \\ \prod_{i=1}^j \mathbf{W}_i(k, t)^T \prod_{i=1}^j \mathbf{W}_i(k, t) &= [\mathbf{W}_\phi(k, t)^T \mathbf{W}_\phi(k, t)]^{\frac{j}{N-1}}. \end{aligned}$$

Now, we can use these expressions to obtain the desired result, by taking the differential equations governing the evolution of $\mathbf{W}_i(k, t)$, for $i \in [1, \dots, N - 1]$, multiplying the i -th equation by $\prod_{i+1}^{N-1} \mathbf{W}_j(k, t)$ from the left, and $\prod_1^{i-1} \mathbf{W}_j(k, t)$ to the right, and then summing over i .

$$\begin{aligned}\dot{\mathbf{W}}_\phi(k, t) &= \sum_{i=1}^{N-1} \left(\prod_{j=i+1}^{N-1} \mathbf{W}_j(k, t) \right) \dot{\mathbf{W}}_i(k, t) \left(\prod_{j=1}^{i-1} \mathbf{W}_j(k, t) \right) \\ &= -\eta \sum_{i=1}^{N-1} \left(\prod_{i+1}^{N-1} \mathbf{W}_j(k, t) \prod_{i+1}^N \mathbf{W}_j(k, t)^T \right) \frac{dL_0(\mathbf{W}_{N:1})}{d\mathbf{W}_{N:1}} \left(\prod_{j=1}^{i-1} \mathbf{W}_j(k, t)^T \prod_{j=1}^{i-1} \mathbf{W}_j(k, t) \right)\end{aligned}$$

The above equation simplifies to the desired result by taking out $\mathbf{W}_N(k, t)$ from the first summation, and using the identities above for each of the terms. \square

Comparing the previous result with Theorem 1 in Arora et al. (2018), we note that the resulting differential equation for weights holds true for arbitrary representations or features in the network provided that the layers from the input to the feature layer are balanced. A direct application of Arora et al. (2018) restricts the model class to only fully balanced networks for convergence analysis and the resulting solutions to the feature matrix will then only have one active singular value, leading to less-expressive neural network configurations.

Proof of Proposition 4.1. Finally, we are ready to use Lemma D.0.2 to prove the relationship with evolution on singular values. This proof can be shown via a direct application of Theorem 3 in Arora et al. (2019). Given that the feature matrix, $\mathbf{W}_\phi(k, t)$ satisfies a very similar differential equation as the end-to-end matrix, with the exception that the gradient of the loss with respect to the end-to-end matrix is pre-multiplied by $\mathbf{W}_N(k, t)^T$. As a result, we can directly invoke Arora et al. (2019)'s result and hence, we have $\forall r \in [1, \dots, \dim(W)]$ that:

$$\dot{\sigma}_r(k, t) = -N \cdot (\sigma_r^2(k, t))^{1 - \frac{1}{N-1}} \cdot \left\langle \mathbf{W}_N(k, t)^T \frac{dL_{N,k}(\mathbf{W}_{K,t})}{d\mathbf{W}}, \mathbf{u}_r(k, t) \mathbf{v}_r(k, t)^T \right\rangle. \quad (\text{D.9})$$

Further, as another consequence of the result describing the evolution of weight matrices, we can also obtain a result similar to Arora et al. (2019) that suggests that the goal of the gradient update on the singular vectors $\mathbf{U}(k, t)$ and $\mathbf{V}(k, t)$ of the features $\mathbf{W}_\phi(k, t) = \mathbf{U}(k, t) \mathbf{S}(k, t) \mathbf{V}(k, t)^T$, is to align these spaces with $\mathbf{W}_N(k, t)^T \frac{dL_{N,k}(\mathbf{W}_{K,t})}{d\mathbf{W}}$.

D.1 EXPLAINING RANK DECREASE BASED ON SINGULAR VALUE EVOLUTION

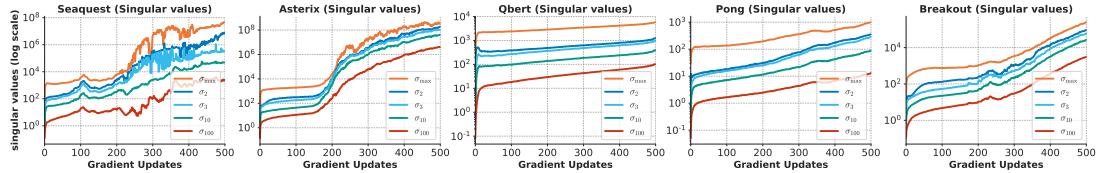


Figure D.1: **Evolution of singular values of Φ on Atari.** The larger singular values of the feature matrix Φ grow at a disproportionately higher rate than other smaller singular values as described by equation 5.

In this section, we discuss why the evolution of singular values discussed in Equation 5 indicates a decrease in the rank of the feature matrix *within* a fitting iteration k . To see this, let's consider the case when gradient descent has been run long enough (*i.e.*, the data-efficient RL case) for the singular vectors to stabilize, and consider the evolution of singular values post timestep $t \geq t_0$ in training. First of all, when the singular vectors stabilize, we obtain that $\mathbf{u}_r(k, t)^T \mathbf{W}_N(k, t)^T \frac{dL_{N,k}(\mathbf{W}_{K,t})}{d\mathbf{W}} \mathbf{v}_r(k, t)$ is diagonal (extending result of Corollary 1 from Arora et al. (2019)). Thus, we can assume that

$$\mathbf{u}_r^T(k, t) \mathbf{W}_N(k, t)^T \frac{dL_{N,k}(\mathbf{W}_{K,t})}{d\mathbf{W}} \mathbf{v}_r(k, t) = \mathbf{f}(k, t) \cdot \mathbf{e}_r \cdot \mathbf{d}_r,$$

where \mathbf{e}_r is given by the unit basis vector for the standard orthonormal basis, \mathbf{f} is a shorthand for the gradient norm of the loss function pre-multiplied by the transpose of the last layer weight matrix, and

\mathbf{d}_r denotes the singular values of the state-action input matrix, $[\mathcal{S}, \mathcal{A}]$. In this case, we can re-write Equation 5 as:

$$\dot{\sigma}_r(k, t) = -N (\sigma_r^2(k, t))^{1-1/N} \cdot \mathbf{f}(k, t) \cdot \mathbf{e}_r \cdot \mathbf{d}_r. \quad (\text{D.10})$$

Note again that unlike Arora et al. (2019), the expression for $\mathbf{f}(k, t)$ is different from the gradient of the loss, since the weight matrix $\mathbf{W}_N(k, t)$ is multiplied to the expression of the gradient in our case. By using the fact that the expression for $\mathbf{f}(k, t)$ is shared across all singular values, we can obtain differential equations that are equal across different $\sigma_r(k, t)$ and $\sigma_{r'}(k, t)$. Integrating, we can show that depending upon the ratio $\frac{\mathbf{e}_r \cdot \mathbf{d}_r}{\mathbf{e}_{r'} \cdot \mathbf{d}_{r'}}$, and the value of N , the singular values $\sigma_r(k, t)$ and $\sigma_{r'}(k, t)$ will take on different magnitude values, in particular, they will grow at disproportionate rates. By then appealing to the result from Proposition 4.1 in Section 4.1 (kernel regression argument), we can say that disproportionately growing singular values would imply that the value of $\text{srank}_\delta(\mathbf{W}_\phi(k, t))$ decreases within each iteration k .

D.2 PROOF FOR THEOREM 4.2: RANK COLLAPSE NEAR A FIXED POINT

Now, we will prove Theorem 4.2 by showing that when the current Q-function iterate \mathbf{Q}_k is close to a fixed point but have not converged yet, i.e., when $\|\mathbf{Q}_k - (\mathbf{R} + \gamma P^\pi \mathbf{Q}_k)\| \leq \varepsilon$, then rank-decrease happens. To prove this, we evaluate the (infinitesimal) change in the singular values of the features of $\mathbf{W}_\phi(k, t)$ as a result of an addition in the value of ε . In this case, the change (or differential) in the singular value matrix, $\mathbf{S}(k, T)$, ($\mathbf{W}_\phi(k, T) = \mathbf{U}(k, T)\mathbf{S}(k, T)\mathbf{V}(k, T)^T$) is given by:

$$d\mathbf{S}(k, T) = \mathbf{I}_d \circ \left[\mathbf{U}(k, T)^T \cdot d\mathbf{W}_\phi(k, t) \cdot \mathbf{V}(k, T) \right], \quad (\text{D.11})$$

using results on computing the derivative of the singular value decomposition (Townsend, 2016).

Proof strategy: Our high-level strategy in this proof is to design a form for ε such that the value of effective rank for feature matrix obtained when the resulting target value $\mathbf{y}_k = \mathbf{Q}_{k-1} + \varepsilon$ is expressed in the deep linear network function class, let's say with feature matrix, $\mathbf{W}'_\phi(k, T)$ and the last layer weights, $\mathbf{W}_N(k, T) = \mathbf{W}_N(k-1, T)$, then the value of $\text{srank}_\delta(\mathbf{W}'_\phi(k, T))$ can be larger than $\text{srank}_\delta(\mathbf{W}_\phi(k-1, T))$ by only a bounded limited amount α that depends on ε . More formally, we desire a result of the form

$$\begin{aligned} \text{srank}_\delta(\mathbf{W}'_\phi(k, T)) &\leq \text{srank}_\delta(\mathbf{W}_\phi(k-1, T)) + \alpha, \\ \text{where } \|\varepsilon\| &\ll \|\mathbf{Q}_{k-1}\|, \text{ and } \mathbf{y}_k(\mathbf{s}, \mathbf{a}) = \mathbf{W}_N(k-1, T)\mathbf{W}'_\phi(k, T)[\mathbf{s}; \mathbf{a}]. \end{aligned} \quad (\text{D.12})$$

Once we obtain such a parameterization of \mathbf{y}_k in the linear network function class, using the argument in Section 4.2 about self-training, we can say that just copying over the weights $\mathbf{W}'_\phi(k, T)$ is sufficient to obtain a bounded increase in $\text{srank}_\delta(\mathbf{W}_\phi(k, T))$ at the cost of incurring 0 TD error (since the targets can be exactly fit). As a result, the optimal solution found by Equation 6 will attain lower $\text{srank}_\delta(\mathbf{W}_\phi(k, T))$ value if the TD error is non-zero. Specifically,

$$\text{srank}_\delta(\mathbf{W}'_\phi(k, T)) + \frac{\|\mathbf{Q}_k - \mathbf{y}_k\|}{\lambda_k} \leq \text{srank}_\delta(\mathbf{W}_\phi(k-1, T)) + \alpha \quad (\text{D.13})$$

As a result we need to examine the factors that affect $\text{srank}_\delta(\mathbf{W}_\phi(k, T))$ – (1) an increase in $\text{srank}_\delta(\mathbf{W}_\phi(k, T))$ due to an addition of α to the rank, and (2) a decrease in $\text{srank}_\delta(\mathbf{W}_\phi(k, T))$ due to the implicit behavior of gradient descent.

Proof: In order to obtain the desired result (Equation D.12), we can express $\varepsilon(\mathbf{s}, \mathbf{a})$ as a function of the last layer weight matrix of the *current* Q-function, $\mathbf{W}_N(k-1, T)$ and the state-action inputs, $[\mathbf{s}; \mathbf{a}]$. Formally, let $\varepsilon(\mathbf{s}, \mathbf{a}) = \mathbf{W}_N(k-1, T)\zeta^T[\mathbf{s}; \mathbf{a}]$, where ζ is a matrix such that $\|\zeta\|_\infty \ll \tau = \mathcal{O}(\|\mathbf{W}_\phi(k, T)\|_\infty)$, i.e., ζ has entries with ignorable magnitude compared the actual feature matrix, $\mathbf{W}_\phi(k, T)$.

Using this form of $\varepsilon(\mathbf{s}, \mathbf{a})$, we note that the targets \mathbf{y}_k used for the backup can be written as

$$\begin{aligned} \mathbf{y}_k &= \mathbf{Q}_{k-1} + \varepsilon = \mathbf{W}_N(k-1, T)\mathbf{W}_\phi(-1, T)[\mathcal{S}; \mathcal{A}] + \mathbf{W}_N(k-1, T)\zeta[\mathcal{S}; \mathcal{A}] \\ &= \mathbf{W}_N(k-1, T) \cdot \underbrace{(\mathbf{W}_\phi(k-1, T) + \zeta)}_{\mathbf{W}'_\phi(k, T)} [\mathcal{S}; \mathcal{A}] \end{aligned}$$

Using the equation for sensitivity of singular values due to a change in matrix entries, we obtain the maximum change in the singular values of the resulting “effective” feature matrix of the targets \mathbf{y}_k in the linear function class, denoted as $\mathbf{W}'_\phi(k, T)$, is bounded:

$$d\mathbf{S}'(k, T) = \mathbf{I}_d \circ \left[\mathbf{U}(k, T) \cdot d\mathbf{W}_\phi(k, T) \cdot \mathbf{V}(k, T)^T \right] \implies \|d\mathbf{S}'(k, T)\|_\infty \leq \zeta. \quad (\text{D.14})$$

Now, let's use this inequality to find out the maximum change in the function, $h_k(i)$ used to compute $\text{srank}_\delta(\mathbf{W}_\phi)$: $h_k(i) = \frac{\sum_{j=1}^i \sigma_j(\mathbf{W}_\phi(k, T))}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi(k, T))}$ ($\text{srank}_\delta(\mathbf{W}_\phi(k, T)) = \min \{j : h_k(j) \geq 1 - \delta\}$).

$$\begin{aligned} |dh_k(i)| &= \left| \frac{\sum_{j=1}^i d\sigma_j(\mathbf{W}_\phi)}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)} - \left(\frac{\sum_{j=1}^i \sigma_j(\mathbf{W}_\phi)}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)} \right) \left(\frac{\sum_{j=1}^N d\sigma_j(\mathbf{W}_\phi)}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)} \right) \right| \\ &\leq \frac{i\zeta}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)} + \underbrace{\left(\frac{\sum_{j=1}^i \sigma_j(\mathbf{W}_\phi)}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)} \right)}_{\leq 1} \frac{N\zeta}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)} \\ &\leq \frac{(i+N)\zeta}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)}. \end{aligned}$$

The above equation implies that the maximal change in the effective rank of the feature matrix generated by the targets, \mathbf{y}_k (denoted as $\mathbf{W}'_\phi(k, T)$) and the effective rank of the features of the current Q-function \mathbf{Q}_k (denoted as $\mathbf{W}_\phi(k, T)$) are given by:

$$\text{srank}_\delta(\mathbf{W}'_\phi(k, T)) - \text{srank}_\delta(\mathbf{W}_\phi(k, T)) \leq \alpha$$

where α can be formally written by the cardinality of the set:

$$\alpha = \left| \left\{ j : h_k(j) - \frac{(j+N)\zeta}{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)} \geq (1-\delta), h_k(j) \leq (1-\delta) \right\} \right|. \quad (\text{D.15})$$

Note that by choosing $\varepsilon(s, a)$ and thus ζ to be small enough, we can obtain $\mathbf{W}'_\phi(k, T)$ such that $\alpha = 0$. Now, the self-training argument discussed above applies and gradient descent in the next iteration will give us solutions that reduce rank.

Assuming $r > 1$ and $\text{srank}_\delta(\mathbf{W}_\phi(k, T)) = r$, we know that $h_k(r-1) < 1 - \delta$, while $h_k(r) \geq 1 - \delta$. Thus, $\text{srank}_\delta(\mathbf{W}'_\phi(k, T))$ to be equal to r , it is sufficient to show that $h_k(r) - |dh_k(r)| \geq 1 - \delta$ since both $h_k(i)$ and $dh_k(i)$ are increasing for $i = 0, 1, \dots, r$. Thus, $\text{srank}_\delta(\mathbf{W}'_\phi(k, T)) = r \forall \zeta$, whenever

$$\begin{aligned} \zeta &\leq \frac{\sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)}{r+N} (h_k(r) - 1 - \delta) \\ &= \frac{\sum_{j=1}^r \sigma_j(\mathbf{W}_\phi) - (1-\delta) \sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)}{r+N} \end{aligned}$$

This implies that $\boxed{\varepsilon \leq \|W_N(k, T)\|_\infty \frac{\sum_{j=1}^r \sigma_j(\mathbf{W}_\phi) - (1-\delta) \sum_{j=1}^N \sigma_j(\mathbf{W}_\phi)}{r+N}}.$

Proof summary: We have thus shown that there exists a neighborhood around the optimal fixed point of the Bellman equation, parameterized by $\varepsilon(s, a)$ where bootstrapping behaves like self-training. In this case, it is possible to reduce srank while the TD error is non-zero. And of course, this would give rise to a rank reduction close to the optimal solution.