OVERCOMING THE SPECTRAL BIAS OF NEURAL VALUE APPROXIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Value approximation using deep neural networks is at the heart of model-free reinforcement learning, and is the primary module that provides learning signals to the rest of the algorithm. While deep networks are universal function approximators, recent works have suggested the presence of a *spectral bias*, where learning high-frequency components of the target function require exponentially more gradient updates than the low-frequency ones. Such dynamics contribute to slow convergence when fitting value functions that are intrinsically complex. In this work, we investigate neural value approximation through the lens of Neural Tangent Kernels (NTK), and propose the use of random Fourier Features (RFF), with a dimension-agnostic initialization scheme, to improve neural value fitting. We demonstrate the advantage of this simple parameter change on the challenging high-dimensional continuous control domains including Ant and Humanoid, where a multi-layer perceptron equipped with RFF produce faster convergence and higher performance.

1 INTRODUCTION

Approximating value functions with deep neural networks is a critical component of state-of-the-art reinforcement learning (RL) algorithms. It has been shown that even for one-dimensional continuous state space and a simple (i.e., piece-wise linear) forward dynamics, there exists many MDPs with highly non-linear optimal Q-functions (Dong et al., 2020). The authors further suggest that, many real-world problems have similarly complex value functions that are challenging to approximate using a deep neural network. To understand these challenges, we re-created one of the toy MDPs from Dong et al. (2020). The forward dynamics and the corresponding optimal Q-function of this MDP are shown in Figure 1.

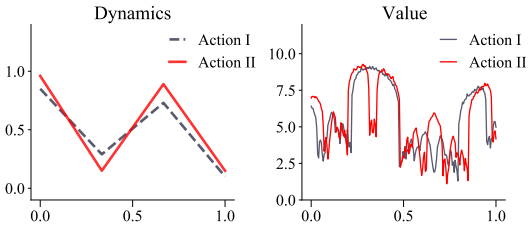
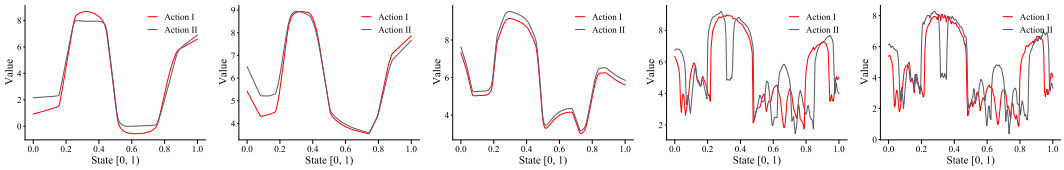


Figure 1: A Toy MDP with a simple forward dynamics, and complex value function.

We use a multi-layer perceptron (MLP) (4 layers deep with 400 hidden units each layer) to learn the optimal Q-function of the toy MDP with offline fitted Q iteration (FQI). We use a dataset of transitions collected by first uniformly sampling state-action pairs from the toy MDP and then taking the sampled action on the sampled state to get the next state. Despite training for 400 epochs, we observe the MLP under-fitting the optimal Q-function. It is natural to ask why the under-fitting occurs despite MLPs being universal function approximators. There could be several possible reasons which we now investigate.

Prior work of Kumar et al. (2021) has argued that minimizing TD (Temporal-Difference) error in value-based deep RL through bootstrapping resembles self-distillation (Mobahi et al., 2020), which is known to induce under-parameterization. To check if the primary cause of under-fitting is bootstrapping, we learn the Q-function using the same MLP architecture with supervised learning using ground truth Q values. Figure 2 shows that the under-fitting still persists after 400 epochs, implying a potential issue with the MLP architecture / training paradigm. To test whether under-fitting is the result of an under-parameterized MLP architecture, we use a 12-layer deep (with 400 hidden units



(a) FQI + MLP@400 (b) Sup. MLP@400 (c) Sup. 12-layer (d) Sup. MLP@2k (e) FQI RFN @400

Figure 2: Description of a toy MDP with $\mathcal{S} = [0, 1)$, $\mathcal{A} = \{0, 1\}$, $\gamma = 0.9$ and reward function $r(s, a) = s$. (a) is a 4-layer MLP with 400 latent neurons learned with fitted Q iteration (FQI) (Riedmiller, 2005) for 400 epochs. (b) is supervised directly via the ground-truth value target, learned using SGD with a minibatch of 32 for 400 epochs. (c) uses a network that is 3x deeper (12-layers) also at 400 epochs. (d) is the same 4-layer network, but optimized longer, for 2000 epochs. (e) is the RFN network at only 400 epochs, without a target network.

each layer) MLP to learn the Q-function with supervised learning. However, as seen in Figure 2, under-fitting still persists implying that the under-parameterization in the MLP architecture might not be the issue. Finally, to test whether under-fitting is caused by the training paradigm, we train the original MLP architecture for 2000 epochs to learn the Q-function with supervised learning. From Figure 2, we see that the MLP fit to the Q-function improves, thereby implying that training for a large number of epochs resolves the issue.

Given that more gradient updates improve the MLP's fit to the Q-function, could we solve the under-fitting issue by making the training period arbitrarily long? No, given a long training period increases compute requirements in offline FQI and additionally causes data inefficiency in online RL. This is because data collection is interleaved with Q learning in online RL. When MLP fit to the Q-function is bad, the resulting policy might take suboptimal actions with low returns resulting in data that might not be useful for Q learning and would need to be recollected. A potential solution to improve data efficiency in online RL is by taking more gradient steps for learning Q-functions between data collection. However, as pointed in Fu et al. (2019), this can cause MLPs to overfit to off-policy samples thereby degrading the overall performance. Therefore, it is imperative to understand the reasons behind the training inefficiency in learning Q-functions so that we can speedup the training.

Recent works (Rahaman et al., 2019; Shah et al., 2020; Yang & Salman, 2019; Huh et al., 2021) have suggested presence of *spectral bias* in MLPs which encourages them to learn the low-frequency components of the target function before the high frequency components. Since the optimal Q-function of the toy MDP has a large number of high-frequency components in its spectrum (as seen from Figure 1), the spectral bias in MLPs might explain why they take a large number of epochs to fit the Q function. To evade the spectral bias of MLPs, some recent works (Huh et al., 2021; Sinha et al., 2020) proposed use of residual connections to increase the rank of MLP features. While this approach improves the performance of RL algorithms on low-dimensional continuous control tasks such as HalfCheetah (state dim of 17), it fails to provide any dividends in complex high-dimensional continuous control tasks such as Ant (state dimension of 111) and Humanoid (state dimension of 376) (Sinha et al., 2020).

Another solution, proposed in the graphics literature (Tancik et al., 2020; Sitzmann et al., 2020; Mildenhall et al., 2020), is to transform the input with random Fourier features before passing it through a MLP. Taking cues from these works, we propose Random Fourier Network (RFN) , as a drop-in replacement of standard MLPs, to improve neural value estimation in MDPs with complex optimal Q-functions. Using RFN (4 layers deep with 400 hidden units each layer) to learn the optimal Q-function of the toy MDP with FQI leads to a near-perfect fit after 400 epochs, as seen in Figure 2. This shows that RFN is successful in evading the spectral bias of the MLPs. In coming sections, we will first describe RFN in detail and finally demonstrate the advantage of using them with actor-critic algorithms (RFAC) on challenging continuous control tasks such as Ant and Humanoid.

2 PRELIMINARIES

A Markov decision process (MDP, Puterman 2014) can be defined by the tuple $\langle S, A, R, P, \mu, \gamma \rangle$ where S and A are the state and action spaces, $P : S \times A \mapsto S$ is the transition function, R :

$S \times A \mapsto \mathbb{R}$ is the reward and $\mu(s)$ is the initial state distribution. We consider an infinite horizon problem where γ is the discount factor. The goal of reinforcement learning is to maximize the expected future discounted return $\mathcal{J} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$.

We define $Q^\pi(s, a)$ function as expected future discounted return following the policy π when starting from state s and taking action a i.e. $Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | (s_0, a_0) = (s, a)\right]$. Furthermore, $\mathcal{J} = \mathbb{E}_{s\ \mu(s), a\ \pi(a|s)}\left[Q^\pi(s, a)\right]$. The optimal $Q^*(s, a)$ is the fixed point of the Bellman optimality operator $\mathcal{B}^*$

$$\mathcal{B}^* Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)}[\max_{a*} Q(s', a^*)] \tag{1}$$

When P isn't known but accessible through a sampled dataset $\mathcal{D}$, we can learn $Q^*(s, a)$ using a parameterized model $Q_\theta(s, a)$ with offline fitted-Q-iteration (FQI) (Riedmiller, 2005) that optimizes the TD loss with Adam (Kingma & Ba, 2015)

$$\min_\theta \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \| Q_\theta(s, a) - B^* Q_{\theta'}(s, a) \| \tag{2}$$

Here, θ' is a running smoothed average of θ from past iterations of Adam optimizer and $Q_{\theta'}$ is called the target Q-network. We use offline FQI for our offline RL experiments. Additionally, we use DDPG as our base RL algorithm for online RL experiments.

2.1 A Kernel-spectral Perspective on The Simplicity-bias

Rahaman et al. (2019) recently observed that it took less gradient steps for a piece-wise linear network with a ReLU activation to fit to a low-frequency signal than a high-frequency one. This intrinsic bias, called the "spectral bias", is a variant of the "simplicity bias" that is observed by Zhang et al. (2021) and Valle-Perez et al. (2018) on different occasions. To understand the source of these biases, the community has made two important connections. The first is *Conjugate Kernel* (CK) or the equivalent *Gaussian process kernel*. CK describes the network parameterization at initialization by computing the gram matrix of the final feature layer (right before the output) over the support. The second is the *Neural Tangent Kernel*, or NTK, which is the gram matrix of the flattened gradient vectors. Its spectrum governs convergence under gradient based updates, in a linear fashion (Rahaman et al., 2019; Yang & Salman, 2019). When the data is not uniformly distributed, the convergence is linear with respect to the density of the data (Fan & Wang, 2020).

Connecting Effective Rank with Kernel Spectrum Consider a random matrix A sampled from a distribution $p(A)$ with the discrete spectrum $\sigma = \{\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_Q\}$, the *effective rank* (Roy & Vetterli, 2007) is defined as the spectral entropy of the nominal distribution given by the spectrum of the matrix according to the following formula

$$\text{erank}(A) = \exp H\left(\left\{\frac{\sigma_i}{\|\sigma\|_1}\right\}\right) \quad \text{where} \quad i \sim \{1, \ldots Q\}. \tag{3}$$

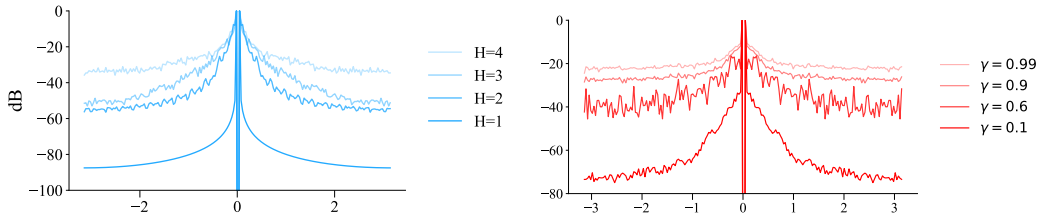
The ℓ^1 norm $\|\sigma\|_1$ is called the *spectral norm*, which acts as the partition function of this distribution. $H(\sigma)$ is called the Shannon entropy. In other words, the effective rank is the exponential of the log likelihood-weighted, normalized sum of the kernel spectrum that describes the volume span by the random matrices.

3 Overcoming The Spectral Bias of Neural Value Estimation

We now apply spectral analysis directly to a toy MDP problem.

Setup We extend the toy MDP problem from Dong et al. 2020 to consider a general class of toy Markov decision process $\mathcal{M}$ with a state space $\mathcal{S} = \mathbb{R}^{[0,1)}$ defined on the real-line. The reward is a sine function $R(s) = \sin(2\pi s)$ with a simple spectral signature. The action space is a discrete set $\{0, 1\}$, each corresponds to a distinct forward dynamics that is randomly sampled from the space of piece-wise linear functions with k "kinks". For all of the examples below, we use a fixed number of kinks $k = 10$, and uniformly sample the value of each turning point in this dynamic function between 0 and 1. The result is a distribution $\mathcal{M}$ that we can sample from

$$M \sim \mathcal{M} = p(M). \tag{4}$$



(a) Value Spectrum vs Horizon (H) (b) Value Spectrum vs γ

Figure 3: Value function spectra over (a) longer time-horizon and (b) larger discount factor γ. Higher frequency component noticeably grows in decibel when the unroll horizon increases.

Fitted Q Iteration Consider a sequence of value functions under fitted Q iteration (Ernst et al., 2005) $\{Q_0(s,a),\ Q_1(s,a),\ Q_2(s,a),\ \cdots Q_H(s,a)\}$ starting from $Q_0(s,a) = R(s,a)$, generated via the repeated application of the Bellman optimality operator

$$\mathcal{B}Q(s,a) = R(s,a) + \gamma \max_{a^*} Q(s', a^*) \text{ where } s' \sim \mathcal{T}(s,a). \tag{5}$$

The Bellman optimality operator is non-linear and imprints a non-trivial spectrum to the resulting Q_i as it is applied recursively. We can get an intuitive understanding of the evolution of this process by looking at the spectra of Q_i at various horizon H, or by changing the discount factor γ. We present the evolving spectra in figure 3. As the horizon of the problem increases, the value function attains a larger weight in the higher-frequency part of the spectrum. The same effect also takes place if we increase the discount factor from a small value 0.1 all the way up to 0.99. We fix the horizon to 200 in this case, and observe a similar increase in the higher frequency components with a longer effective recursion depth.

We can quantitatively describe this trend by taking the effective rank of each value function, which is equivalent to computing the Shannon entropy of the spectrum. We present these results in figure 4. In other words, the complexity of the value function comes from the repeated application of the Bellman optimality operator in a process that is not dissimilar to an "infinite corridor". The spectrum of the Bellman operated gets folded exponentially into each Q iteration. To match this complexity, we need a function approximator class that has a similar long-tail in the higher-frequency spectrum, to avoid an exponential convergence drop-off in these areas.

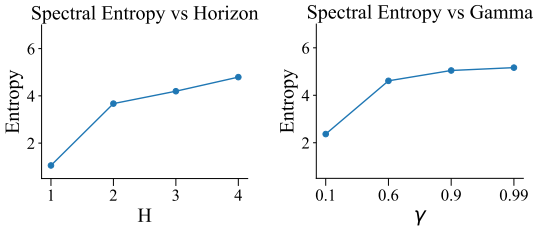


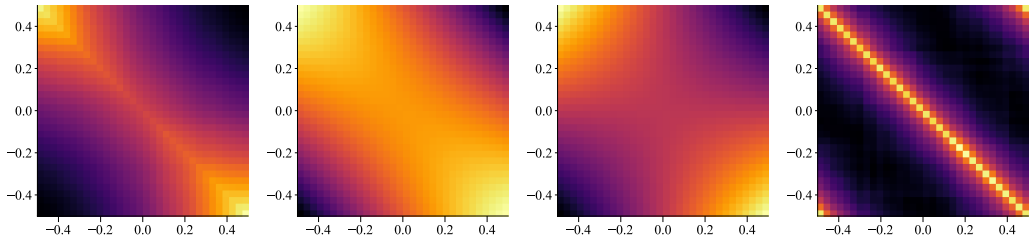
Figure 4: The Spectral entropy of the value function over (a) longer time-horizon and (b) larger discount factor γ. Showing an overall trend that increase over more recursion.

3.1 The Spectral Bias of Neural Value Approximation (MLP)

Recent discussion on the implicit bias of deep neural networks during gradient-based learning uncover a disturbing bias against high-frequency features (Rahaman et al., 2019). While the literature in computer vision primarily focuses on the generalization benefits of a band-limited representation, commonly referred to as the *simplicity bias* (Zheng et al., 2021; Huh et al., 2021), it makes the implicit assumption that the target function is low-rank.

This type of rank bias can be understood with more nuance by looking at the neural tangent kernel. The kernel matrix of the gradient vectors effectively regulates the dynamics of the network during training. A lower effective rank of the gradient vector means the flow in the parameter space happens on a lower dimensional manifold. This is the reason why the the spectrum and effective rank of the NTK directly impacts the solutions we find via gradient updates and neural networks.

Figure 5 compares between four network architecture. The first three, a multi-layer perceptron (MLP) with ReLU activation, with hyperbolic tangent, and residual connection all produce non-stationary kernels that lack a clear diagonal.



(a) MLP + ReLU (b) MLP + Tanh (c) Dense Net (D2RL) (d) RFN

Figure 5: NTK comparison between (a) MLP with ReLU activation, (b) MLP with tanh activation, (c) Dense network used in D2RL and (d) RFN. (a-c) are not stationary, and lack a clear diagonal. The spectrum is concentrated around the low-frequency components, making the gradient vectors low-rank.

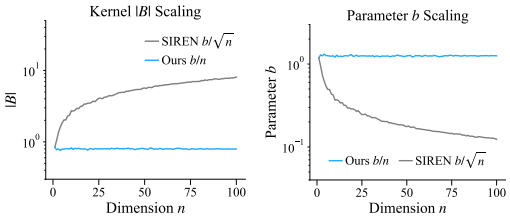
3.2 Random Fourier Networks for Deep Reinforcement Learning

Random Fourier features are shown to improve the ability for networks to learn high-frequency components in computer graphics (Mildenhall et al., 2020; Tancik et al., 2020; Sitzmann et al., 2020).

A key issue with devising random Fourier networks for reinforcement learning problems, is that the state space can have up to hundreds of dimensions (376 for Humanoid, 111 for Ant for instance). This is in stark contrast with the 3-dimensional coordinate-based networks that NeRF (Tancik et al., 2020) and Siren (Sitzmann et al., 2020) considered. Sitzmann et al. (2020) in particular, uses an Xiavier-like initialization scheme for the linear weights that takes following form

$$w_{i,j} \sim \mathcal{U}(-c/\sqrt{n}, c/\sqrt{n}) \text{ where } c = \sqrt{6}. \tag{6}$$

The pre-activation is a sum of the input dimension that ought to grow linearly with the dimension n, and a dimension-sensitive initialization scheme could produce a narrow band, if any, of suitable band-limit parameter b. To mitigate this dimension sensitivity, SIREN proposes a *magic number* $c_0 = 30$ for the very first layer, which has only 2 or 3 input dimensions, as oppose to in later layers that has dimensions in the hundreds. In practice SIREN kernels also tend to be sensitive to this initialization hyperparameter, and tend to be unstable during training.



(a) Pre-activation Scaling (b) Parameter Scaling

Figure 6: We compare the scaling of (a) the kernel bandwidth $|\boldsymbol{B}|$ versus the input dimension, and (b) the bandwidth parameter $\boldsymbol{b}$ needed to attain the same kernel spectrum. Our initialization is dimension-agnostic, and remains a constant. In contrast, the SIREN band parameter require adjustment according to a power law.

To accommodate the large latitude of different state input dimensions across the control benchmark suite, we propose the following alternative for weights and biases

$$\text{RFF}(x)_j = \sin(\sum_i w_{i,j} x^i + b_j) \text{ where } w_{i,j} \sim \mathcal{N}(0, \pi b/n) \tag{7}$$

$$b_j \sim \mathcal{U}(-\pi, \pi) \tag{8}$$

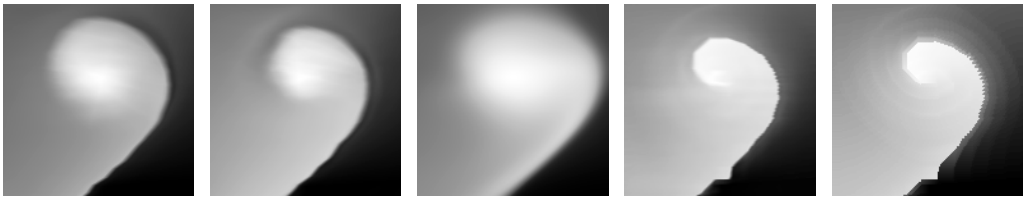
Here, RFF projects x from input dimension to *fourier dimension*. We find this initialization scheme work well across all domains, with minimum requirement on tuning, even when stacked as learnable units. To normalize the state inputs and make sure that they fall between 0 and 1, we subtract the mean and divide by either the standard deviation or min and maximum of the data. *State aliasing* is another concern for weight initialization, as the range of the data needs to be shorter than the *longest* wavelength in the feature mapping. In this case, the lowest spatial frequency is arbitrarily low because we uniformly initialize the weights across the zero-point. Our experiments show that this particular combination of initialization and feature mapping produces the best results on both

toy domains and the more challenging control problems. When this parameterization of Random Fourier Features (RFF) is stacked with MLPs, we call this **Random Fourier Networks (RFN)**.

4 RESULTS

4.1 MOUNTAIN CAR

The Mountain Car environment (Moore, 1990; Sutton, 2000) has a two-dimensional state space that is easy to visualize. In figure 7, we compare the value estimate from three baselines: the ground-truth value estimate acquired using tabular value iteration; one obtained from a 4-layer ReLU network using fitted Q iteration; and one from the same network, but using a 16 dimensional Fourier features on the input. All networks use 400 latent neurons and are optimized for 2000 epochs.



(a) 4-layer MLP (b) 12-layer MLP (c) MLP + tanh (d) 4-layer RFN (e) Tabular

Figure 7: Visualizing the value for action 0 in Mountain Car. (a) shows the action-conditioned value estimate from a 4 layer perceptron network. (b) is from an 8-layer network (c) is a 4-layer network using the hyperbolic-tangent activation, which is commonly used in deepr reinforcement learning. (d) is from a 4-layer RFN. (e) the ground-truth produced by running value iteration using a value table. The two axis corresponds to position and velocity of the car.

4.2 MUJOCO DOMAIN SETUP

In addition to experiments on toy domains, we show that using Random Fourier Network (RFN), in lieu of standard multi-layer perceptrons (MLP), improves performance of RL on continuous control tasks.

Environment We use HalfCheetah, Ant and Humanoid continuous control tasks from OpenAI gym Brockman et al. (2016) for evaluating our method. While HalfCheetah is comparatively low-dimensional with a state dimension of 17, Ant and Humanoid are high dimensional with a state dimension of 111 and 376 respectively. The high-dimensional state space of Ant and Humanoid makes them challenging.

Baselines We use DDPG as our baseline RL algorithm and adapt drqv2 (Yarats et al., 2021) codebase to state space for our implementation. We use a standard 4 layer MLP for both actor and critic. To ensure that DDPG doesn't perform worse due to network under-parameterization, we include DDPG with 6 layer MLP for actor and critic as a baseline (DDPG-deep). Recent works Sinha et al. (2020); Huh et al. (2021) showed that residual connections He et al. (2016) help MLP avoid simplicity bias. Therefore, we included DDPG with D2RL Sinha et al. (2020) architecture as another baseline (DDPG-dense). In section 3, we showed that MLP can avoid its spectral bias when trained with more gradient steps. Hence, we include DDPG baselines where the critic is updated for 5 (DDPG (grad 5)) and 10 (DDPG (grad 10)) gradient steps for every 2 environment step. Note that DDPG takes 1 gradient step for its critic for every 2 environment step by default. We train all the baselines for $1e6$ environment steps with standard hyperparameters as reported in the respective original papers. See Appendix A.7 for more details.

Random Fourier Actor Critic (RFAC) We combine RFN with an actor-critic algorithm DDPG (RFAC) to demonstrate the ability of RFN to improve neural value estimation thereby leading to improved performance on control tasks. An RFN is similar to an MLP, except that the first layer transforms the input with Random Fourier Features (Equation 7) which isn't learned. We use a standard 4-layer RFN for both the actor and the critic having the same number of hidden units as the MLP layers in the actor and the critic of the DDPG baseline to ensure a fair comparison. We try to

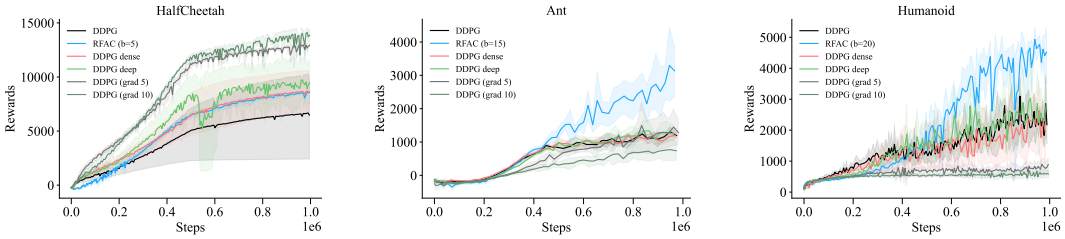


Figure 8: We compare RFAC with DDPG and other baselines on HalfCheetah, Ant and Humanoid control tasks. It improves over DDPG and either matches or outperforms DDPG-dense on all the tasks. DDPG-deep improves over DDPG and performs similar to RFAC and DDPG-dense on HalfCheetah. On Ant and Humanoid, it performs similar to DDPG implying that overparameterization doesn't always help.

maintain $\frac{\text{Fourier dim}}{\text{input dim}}$ to be at least approximately around $40 - 50$. Hence, we use a Fourier dimension of 800, 4000 and 14000 for HalfCheetah, Ant and Humanoid states, and a Fourier dimension of 400, 400 and 800 for HalfCheetah, Ant and Humanoid actions. We train RFAC for $1e6$ environment steps. RFAC introduces only one additional hyperparameter i.e. bandwidth b. We do a hyperparameter search over the values $\{5, 10, 15, 20\}$ to select an best performing bandwidth b. We keep the other hyperparamters to be same as that of DDPG. See Appendix A.7 for more details.

4.3 Performance on continous control tasks

Figure 8 shows that RFAC significantly improve over DDPG for HalfCheetah, Ant and Humanoid control tasks. As shown in section 3, RFN improves Q value estimation by correcting the spectral bias of the standard MLPs. We hypothesize that RFAC improves over DDPG due to improved Q value estimation. DDPG-dense is another method which tries to evade the simplicity bias of the MLPs by using residual connections. While it improves over DDPG on low-dimensional control task like HalfCheetah, it fails to provide any dividends and matches the performance of DDPG on high-dimensional continuous control tasks such as Ant and Humanoid. This trend was also observed in Sinha et al. (2020). In contrast, RFAC improves over DDPG in all the tasks, matches the performance of DDPG-dense on Half-Cheetah and outperforms DDPG-dense on Ant and Humanoid.

To ensure that the poor performance of DDPG isn't due to under-parameterization, we observe the performance of DDPG-deep. It improves over DDPG on HalfCheetah and performs similar to DDPG-dense and RFAC. However, it performs similar to DDPG on Ant and Humanoid. This shows that over-parameterization doesn't always help, thereby implying that under-parameterization might not be the primary reason behind poor performance of DDPG.

Finally, to check if more gradient steps for critic help it avoid the simplicity bias, we observe the performance of DDPG (grad 5) and DDPG (grad 10). Both these methods outperform all the other methods on HalfCheetah. But, they only match the performance of DDPG on Ant and perform really poorly on Humanoid. While more gradient steps for critic can help it evade the spectral bias in some tasks (like HalfCheetah), it also causes over-fitting to off-policy samples Fu et al. (2019) in other tasks thereby leading to poor performance (like Ant and Humanoid). Hence, DDPG (grad 5) and DDPG (grad 10) are overall unreliable.

4.4 Sensitivity to Hyperparameters

To better understand RFN (and RFAC), we analyze how different hyperparameters affects performance of RFAC. Since RFN introduces one additional hyperparameter, i.e. bandwidth b, it is natural to ask how the choice of b affects the performance of RFAC. Figure 9 shows that the performance of RFAC remains the same over different values of b on HalfCheetah and Ant. However, its performance vary a lot for different values of b on Humanoid. Furthermore, the best performing value of b differs with task, as seen from Figure 9. This difference arises from the fact that the optimal Q value function for different environments have different spectral bias and hence requires different bandwidth b for Random Fourier Network.

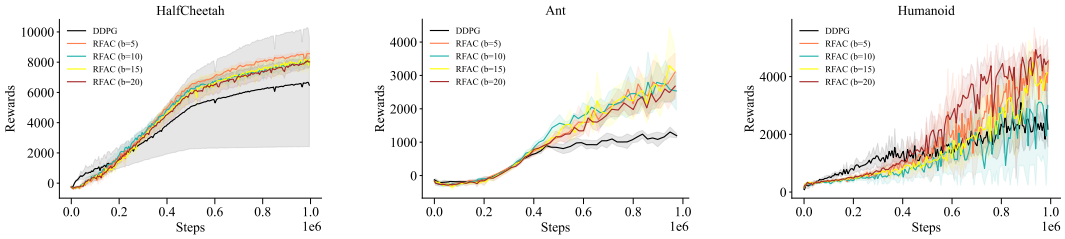


Figure 9: We analyze the performance of RFAC for different values of bandwidth b on HalfCheetah, Ant and Humanoid control tasks. While RFAC performs similarly on HalfCheetah and Ant for different values of bandwidth b, its performance varies on Humanoid with different values of b.

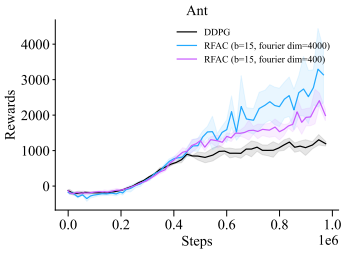


Figure 10: We analyze the performance of RFAC for state fourier dimensions on Ant. It suffers a significant drop in performance if we decrease its state fourier dimension from 4000 to 400.

In addition to the bandwidth b, we need to choose the Fourier dimension for state and actions with RFAC. We tried to maintain $\frac{\text{Fourier dim}}{\text{input dim}}$ to be approximately around $40 - 50$.

Due to high state dimension in Ant and Humanoid, we ended up with 4000 and 14000 Fourier dimension for their states respectively. But can we get away with a lower Fourier dimension? Figure 10 shows that RFAC suffers a significant drop in performance on Ant if we decrease the state Fourier dimension from 4000 to 400. Furthermore, RFAC completely fails if we decrease its state Fourier dimension from 14000 to even 4000. This shows that it is important to maintain $\frac{\text{Fourier dim}}{\text{input dim}}$ to be at least around $40 - 50$.

Till now, we have used full rank matrix (with sin activation) to parameterize Random Fourier Features (in RFN) which projects the input into fourier dimension. Given we try to maintain $\frac{\text{Fourier dim}}{\text{input dim}}$ to be at least around $40 - 50$, the fourier dimension can get large due to high dimensional input and as a result, the full rank matrix, used to parameterize Random Fourier Features, gets even larger. It is natural to ask if there's more efficient way to parameterize Random Fourier Features. An alternate parameterization is to use 1d convolution (with sin activation) for Random Fourier Features. We then transform each input dimension independently with Random Fourier Features into separate vectors which finally gets concatenated. However, figure 11 shows that RFAC suffers a significant drop in performance if we use this alternate 1d parameterization. This implies that it is important to use full rank matrix to parameterize Random Fourier Features in RFN.

5 Related Works

Coordinate-based MLPs There have been recent works (Mildenhall et al., 2020; Tancik et al., 2020; Lin et al., 2021; Sitzmann et al., 2020) in using coordinate-based MLPs with sinusoidal activations for volumetric rendering and view-synthesis. Other works (Vaswani et al., 2017; Zheng et al., 2021) have used the same for positional encoding in attention for processing sequential data. Our work is inspired from these works and make use of coordinate-based MLPs with sinusoidal activations to model complex value functions in reinforcement learning.

Simplicity bias in MLPs Past works (Huh et al., 2021; Shah et al., 2020; Rahaman et al., 2019; Tancik et al., 2020; Yang & Salman, 2019) have noted that MLPs are biased towards simpler functions and take a longer time to converge to complex functions. While Huh et al. (2021) study this

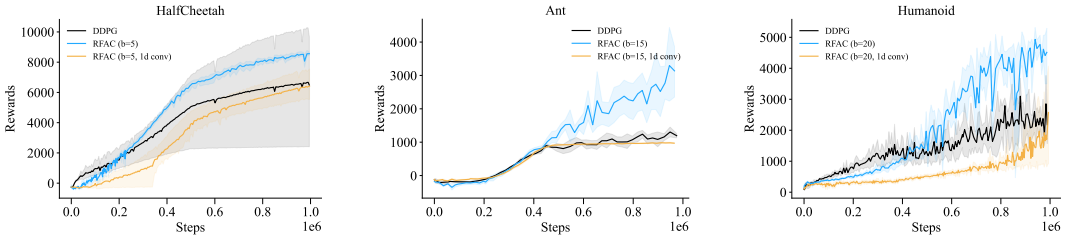


Figure 11: RFAC's performance drops significantly on HalfCheetah, Ant, and Humanoid if we take Random Fourier Features independently per dimension via 1d convolution and then concatenate them.

property of MLP in context of rank of the learned features, Shah et al. (2020) study the same using the decision boundary of learned MLP. Rahaman et al. (2019); Yang & Salman (2019); Tancik et al. (2020) study the simplicity bias by analysis the spectrum of the learned features. Other works (Arpit et al., 2017; Poggio et al., 2017) have studied this implicit bias from the perspective of generalization. In our work, we observe this implicit bias when learning complex value functions and we use coordinate-based MLPs with sinusoidal activations to evade the implicit simplicity bias.

Fourier Features in Control Prior works have used linear combination of basis (Konidaris et al., 2011) or radial basis functions (Rajeswaran et al., 2017) to represent policies to make reinforment learning simpler. Other works have used random features in control either in the attention mechanism (Choromanski et al., 2021) or in learning transition dynamics with contrastive learning (Nachum & Yang). In contrast, we use random features with MLPs to evade their spectral bias and improve neural value estimation in MDPs with complex Q-functions.

6 Discussion and Future Work

In this work, we address how to improve value estimation in MDPs where Q-functions have spectral bias towards high frequency components. Taking cues from recent development in graphics (Sitzmann et al., 2020; Mildenhall et al., 2020; Tancik et al., 2020), we make use of Random Fourier Network (RFN), a drop-in replacement for multi-layer perceptron, to enable faster convergence towards high frequency components of Q-function. This allows us to improve performance over existing standard RL algorithms (DDPG) as well as other baselines on challenging continuous control tasks such as Ant and Humanoid. Furthermore, to better highlight our choices in RFAC, we analyze its performance over different values of bandwidth b, different Fourier dimension and different ways to apply RFN to inputs.

In this work, we limited ourselves to single task settings and studied how improving neural value estimation for a single task using RFN leads to improved performance on that task. However, changes in value estimation also affects the ability of value function to generalize to new tasks. Therefore, studying generalization properties of RFN, in context of multi-task settings and/or goal-conditioned environments, is an interesting avenue for future work.

References

Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 233–242. PMLR, 2017. URL https://proceedings.mlr.press/v70/arpit17a.html.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Krzysztof Choromanski, Deepali Jain, Jack Parker-Holder, Xingyou Song, Valerii Likhosherstov, Anirban Santara, Aldo Pacchiano, Yunhao Tang, and Adrian Weller. Unlocking pixels for reinforcement learning via implicit attention. *arXiv preprint arXiv:2102.04353*, 2021.

Kefan Dong, Yuping Luo, Tianhe Yu, Chelsea Finn, and Tengyu Ma. On the expressivity of neural networks for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 2627–2637. PMLR, 2020.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Zhou Fan and Zhichao Wang. Spectra of the conjugate kernel and neural tangent kernel for linear-width neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, pp. 2021–2030. PMLR, 2019.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *International Conference on Learning Representations*, 2021.

Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. *arXiv preprint arXiv:2104.06405*, 2021.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Hossein Mobahi, Mehrdad Farajtabar, and Peter L. Bartlett. Self-distillation amplifies regularization in hilbert space. *Advances in Neural Information Processing Systems*, 33, 2020.

Andrew W. Moore. Efficient memory-based learning for robot control. 1990.

Ofir Nachum and Mengjiao Yang. Provable representation learning for imitation with contrastive fourier features. In *Advances In Neural Information Processing Systems*.

Tomaso Poggio, Kenji Kawaguchi, Qianli Liao, Brando Miranda, Lorenzo Rosasco, Xavier Boix, Jack Hidary, and Hrushikesh Mhaskar. Theory of deep learning iii: explaining the non-overfitting puzzle. *arXiv preprint arXiv:1801.00173*, 2017.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. pp. 5301–5310, 2019.

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. *arXiv preprint arXiv:1703.02660*, 2017.

Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo (eds.), *Machine Learning: ECML 2005*, pp. 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. pp. 606–610, 2007.

Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *arXiv preprint arXiv:2006.07710*, 2020.

Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning. *arXiv preprint arXiv:2010.09163*, 2020.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

R Sutton. Mountain car software. http://www.cs.ualberta.ca/~sutton/MountainCar/MountainCar.html, December 2000. URL http://www.cs.ualberta.ca/~sutton/MountainCar/MountainCar.html. Accessed: 2021-10-3.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

Guillermo Valle-Perez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *arXiv preprint arXiv:1805.08522*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Greg Yang and Hadi Salman. A fine-grained spectral perspective on neural networks. *arXiv preprint arXiv:1907.10599*, 2019.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

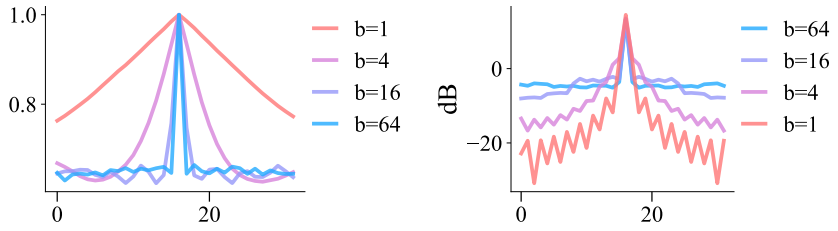
Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Jianqiao Zheng, Sameera Ramasinghe, and Simon Lucey. Rethinking positional encoding. *arXiv preprint arXiv:2107.02561*, 2021.

A APPENDIX

A.1 NEURAL TANGENT KERNEL SPECTRUM

The neural tangent kernel of the random Fourier feature network is stationary. This means we can visualize the kernel spectrum by taking the Fourier transform of a single row of the kernel matrix. We following the analytical procedure from (Tancik et al., 2020), and compare the kernel spectrum under different band-limit parameter b below. Larger band parameter leads to sharper kernels. In the position domain, this indicate better "isolation" between near-by data points. In the spectral domain, this corresponds to a broader spectrum that has a larger fraction of high-frequency components.



(a) NTK Cross-section (b) The Spectra of The Tunnable NTK

Figure 12: The kernel cross-section and spectra for random kernels. (a) The cross-section of the kernel at different band-limit. (b) The spectra at different band-limit, showing an overall bias towards higher frequency bands when the band-limit parameter is higher.

A.2 COMPARING VARIOUS FOURIER FEATURES

Using random Fourier features to correct the spectral-bias is a general technique that goes beyond a particular parameterization. Hence we present comparison between

- **vanilla MLP** is a stack of t linear layers with ReLU activation

$$\text{MLP}(x) = f^t \circ \text{ReLU} \circ \cdots f^2 \circ \text{ReLU} \circ f^1(x)$$

- **Random Fourier Features (RFF) (Ours)** uses sine activation with random phase-shift, to replace the first layer of the network.

$$RFF(x) = \sin(Wx + c), \quad W_{i,j} \sim \mathcal{N}(0, \pi b/n), c_i \sim \mathcal{U}(-\pi, \pi)$$

  so that the network is

$$\text{Net}(x) = f^t \circ \text{ReLU} \circ \cdots f^2 \circ RFF(x)$$

- **Learned Fourier Features (LFF) (ours)** is the same as RFF except that the weights and biases are learnable. We indicate this with torch pseudo code below

$$LFF(x) = RFF.\text{requires_grad_(True)}$$

- **RFF (Tancik et al., 2020)** that uses sine and cosine pairs concatenated together

$$RFF_{\text{Tancik}}(x) = [\sin(2\pi Wx), \cos(2\pi Wx)], \quad W_{i,j} \sim \mathcal{N}(0, \sigma^2)$$

- **SIREN network (Sitzmann et al., 2020)** that stacks learned Fourier layers through our the entire network, using the Sitzmann initialization according to

$$\mathit{lff}(x) = \sin(Wx + c), \quad W_{i,j} \sim \mathcal{U}(-\sqrt{6}/\sqrt{n}, \sqrt{6}/\sqrt{n}), c_i \sim \mathcal{U}(-\pi, \pi)$$

  where the t-layer network

$$\text{SIREN}(x) = \mathit{lff}^t \circ \cdots \mathit{lff}^2 \circ \mathit{lff}^1(x).$$

  It is critical to note what each layer has a distinct bandwidth scaling parameter. Hence instead of a single scaling hyperparameter b for the random matrices, Siren has a set $\{b_i\}$ that each need to be tuned.

We present the results in Figure 13. For brevity, we refer to each of the network architecture above by the name of the features. So an RFF-MLP network becomes *RFF*, and an MLP with a learned Fourier features (LFF) becomes simply LFF. The Siren network (SIREN) is a stack of learned Fourier layers with sine activation. For a fair comparison between $\text{RFF}_{\text{Tancik}}$ and RFF (ours), we use the same initialization with $\sigma = \sqrt{\frac{\pi b}{n}}$.

Despite of the similarity between these architectures, RFF (ours) has a small edge compared to other random Fourier feature parameterizations. SIREN is known to be sensitive to the distribution of its latent distributions and the way one initialize the network, and the flat-lined response reflects this issue.

A second notable observation is that in comparison to vanilla MLP, Fourier feature network variants tend to have much smaller variance. This observation will remain consistent in other experiments in this paper.

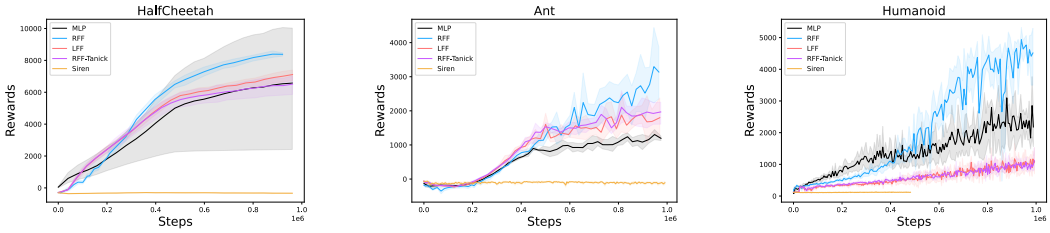


Figure 13: We compare MLPs with different fourier feature parameterizations on how they perform on continous control tasks. We use DDPG as the base reinforcement learning algorithm.

A.3 Measuring the Complexity of Learned Value Functions

Section 4.3 shows that RFAC is able to outperform DDPG and other baselines as RFN is able to avoid the simplicity bias of MLPs and improve the value function fit. To further verify this, we use the effective rank (ρ) (Roy & Vetterli, 2007) as a proxy for the simplicity bias of the network. This metric has also been used in recent works (Kumar et al., 2021; Huh et al., 2021) to measure the complexity of a learned network.

Table 1: Effective rank (Roy & Vetterli, 2007) of Q value function features learned by RFAC and DDPG on HalfCheetah, Ant and Humanoid.

	Half-Cheetah (200)	Ant (100k)	Humanoid
RFAC	2.94 ± 1.12	2.86 ± 1.32	2.94 ± 1.26
DDPG	1.01 ± 1.00	1.08 ± 1.07	1.49 ± 1.22

To calculate the effective rank (ρ), we concatenate Q value function features (from 2nd last layer) into a matrix A ($\in R^{n,d}$) and then use the formula

$$\rho(A) = \exp\left(-\sum_{i=1}^{d} \hat{\sigma}_i \log(\hat{\sigma}_i)\right)$$

where $\hat{\sigma}_i = \sigma_i / \sum_{i=1}^{d} \sigma_i$ are normalized singular values.

From table 1, we see that the effective rank of the Q value function features learned by RFN is higher than that learned by DDPG across all the environments. This shows that RFN was successful in learning Q value functions of higher complexity and therefore avoid the simplicity bias of MLPs.

A.4 RFN's performance on DMC domain

To check the applicability of RFN to different continous control domains, we test it on the DeepMind Control suite domain, Hopper-hop (see Figure 15). The DDPG baseline has a large amount of

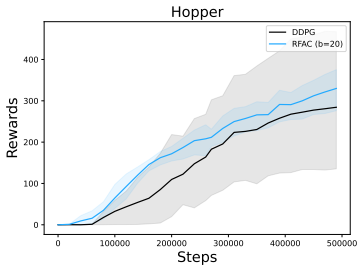


Figure 14: DeepMind Control suite domain: Hopper-hop. RFF network has lower variance, and out-performs the vanilla MLP baseline on average.

variance in this domain. The RFF baseline has much lower variance, and out-performs the baseline on average. Results aggregate from 5 random seeds.

A.5 RFN'S PERFORMANCE WITH TD3

Random Fourier Networks is a general technique that can be plugged into wide range of algorithms. In this experiment, we apply RFN to the current state of the art algorithm, TD3, and demonstrate an even larger gain on performance. The gap produced by RFN (see figure 15) indicates that even with much stronger learning procedures, there is still room for improvement that can be closed by using a better function approximator class.

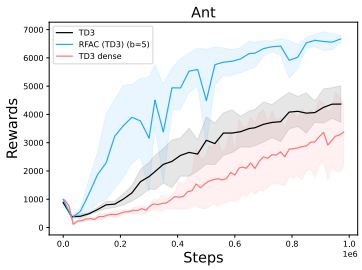


Figure 15: RFF network applied to state-of-the-art TD3 algorithm on Ant. Showing significant performance gain, and a new state-of-the-art result.

A.6 TOY MDP EXPERIMENT DETAILS

Offline Data Generation We divided the 1-dimensional state space into 1000 discrete bins and used the midpoints of the bins as initial states. We then took both the actions from these initial states to get corresponding next states. We used the dataset of these transitions (2000 total transitions) as our offline dataset.

Optimization details We use 4-layer MLP with ReLU Activation, with 400 latent neurons. We use Adam optimization with a learning rate of 1e-4, and optimize for 400 epochs. We use gradient descent with a batch size of 200. For a deeper network, we use a 12-layer MLP, with 400 latent neurons but keep other optimization hyperparameters fixed. To show that longer training period help MLPs evade the spectral bias, we use a 4-layer MLP, with 400 latent neurons and train it for 2000 epochs. All the optimization hyperparameters are kept the same.

A.7 EXPERIMENT DETAILS

We now describe the actor and critic architecture used by DDPG, DDPG-dense, DDPG-deep and RFAC on Hopper(-hop), HalfCheetah, Ant and Humanoid control tasks.

RFAC uses a 4 layered RFN for both actor and critic for all the tasks. RFN is same as MLP except the first layer transforms the input with random fourier features rather than with a linear layer followed by a ReLU activation. For actor, RFN first transforms the state with Random Fourier Features into a vector of dimension H_s which then gets passed into a 3 layered MLP with $[4000, 400]$ hidden units. For critic, RFN separately transforms the state and the action with Random Fourier Features into vectors of dimension H_s and H_a respectively which are they concatenated into a vector of dimension $H_s + H_a$. The concatenated vector gets passed into a 3 layered MLP with $[4000, 400]$ hidden units. We use $H_s = 800$ and $H_a = 400$ for HalfCheetah and Hopper, $H_s = 4000$ and $H_a = 400$ for Ant and $H_s = 14000$ and $H_a = 800$ for Humanoid.

DDPG uses a 4 layered MLP, with $[H_s, 4000, 400]$ hidden units for actor and $[H_s + H_a, 4000, 400]$ hidden units for critic. We ensure it uses the same number of parameters (both learanble and non-learnable) as RFAC for fair comparison. Therefore, we use $H_s = 800$ and $H_a = 400$ for HalfCheetah and Hopper, $H_s = 4000$ and $H_a = 400$ for Ant and $H_s = 14000$ and $H_a = 800$ for Humanoid.

DDPG-deep uses a 6 layered MLP, with $[H_s, 4000, 400, 400, 400]$ hidden units for actor and $[H_s + H_a, 4000, 400, 400, 400]$ hidden units for critic. We use $H_s = 800$ and $H_a = 400$ for HalfCheetah, $H_s = 4000$ and $H_a = 400$ for Ant and $H_s = 14000$ and $H_a = 800$ for Humanoid.

DDPG-dense uses same MLP architecture as DDPG for both actor and critic except it appends the hidden units of MLP with inputs.