OVERCOMING THE SPECTRAL BIAS OF NEURAL VALUE APPROXIMATION

Ge Yang[*], **Anurag Ajay**[*] **& Pulkit Agrawal**
NSF AI Institute of AI and Fundamental Interactions (IAIFI)
Computer Science and Artificial Intelligence Laboratory (CSAIL)
Massachusetts Institute Technology

ABSTRACT

Value approximation using deep neural networks is at the heart of off-policy deep reinforcement learning, and is often the primary module that provides learning signals to the rest of the algorithm. While multi-layer perceptrons are universal function approximators, recent works in neural kernel regression suggest the presence of a *spectral bias*, where fitting high-frequency components of the value function requires exponentially more gradient update steps than the low-frequency ones. In this work, we re-examine off-policy reinforcement learning through the lens of kernel regression and propose to overcome such bias via a composite neural tangent kernel. With just a single line-change, our approach, the Fourier feature networks (FFN) produce state-of-the-art performance on challenging continuous control domains with only a fraction of the compute. Faster convergence and better off-policy stability also make it possible to remove the target network without suffering catastrophic divergences, which further reduces TD(0)'s bias to overestimate the value. Code and analysis available at https://geyang.github.io/ffn.

1 INTRODUCTION

At the heart of reinforcement learning is the question of how to attribute credits or blame to specific actions that the agent took in the past. This is referred to as the *credit assignment* problem (Minsky, 1961). Correctly assigning credits requires reasoning over temporally-extended sequences of observation and actions, so that trade-offs could be made, to choose a less desirable action at a current step in exchange for higher rewards in the future. Temporal difference methods such as TD(λ) and Watkins Q-learning (Sutton, 1988; Watkins, 1989) stitch together the immediate rewards local to each state transition, to estimates the discounted sum of rewards over longer horizons. This is an incremental method for dynamic programming (Watkins & Dayan, 1992) that successively improves the value estimate at each step, which reduces the computation that is otherwise needed to plan ahead at decision time.

A key tension in scaling TD learning to more challenging domains is that the state space (and action spaces in continuous control problems) can be very large. In Neurogrammon (Tesauro, 1991), for example, the complexity of the state space is on the scale of $\mathcal{O}(10^{20})$, which prevents one from storing all state-action pairs in a table. Such constrain and the need to generalize to unseen board arrangements prompted Tesauro (1991) to replace the look-up table with an artificial neural network to great effect, achieving master-level play on backgammon.

Adopting neural networks as the value approximator in general, however, introduces two new issues. First, the standard perceptron network does not offer explicit ways to specify how it generalizes, making it impossible to control the *bias-variance* trade-off to better align with the value approximation task at hand (Canatar et al., 2021). Recent analysis indicates an “expressivity gap” that causes the value approximator to underfit (Dong et al., 2020), further indicates that there exists a *misalignment* between the model and the task, and that we are on the wrong side of the “double-descent” phase transition (Canatar et al., 2021). Modern state-of-the-art off-policy algorithms also tend to be bottlenecked by compute, making the convergence dynamics a key candidate for potential improvements in both sample efficiency and asymptotic performance.

[*] Equal contribution, order determined by rolling a dice.

A second, and older issue around neural function approximators is *off-policy divergence* (Sutton & Barto, 2018), where the iterative Q learning procedure tends to "crash" when over-parameterized function approximators are combined with value bootstrapping and off-policy samples (Bertsekas, 1995; Baird, 1995). A number of contributing causes have been investigated, and corresponding practical techniques have been developed to address such issues. Riedmiller (2005a) employ *experience replay* (Lin, 1992) from a buffer of transition tuples to reduce the cross-talk between states during gradient updates. To make the bootstrapped regression target more stationary and delay the divergence, Mnih et al. (2015) introduced a target value network that changes at a slower speed. Additional bias occurs in the TD(0) objective due to the "winner's curse" (Thaler, 1988), which motivated ensemble based variance-reduction techniques such as double Q-learning (Hasselt, 2010) and bootstrapped Q-ensembles (Lee et al., 2021). In continuous control tasks, computing the value target further require participation of the actor that is often sub-optimal. This introduces additional variance in actor-critic methods that motivated delayed actor updates (Fujimoto et al., 2018), entropy regularization (Haarnoja et al., 2018), and trusted region for the policy optimization (Neumann et al., 2009; Abdolmaleki et al., 2018). None of these instabilities occur, however, under the tabular case. Nor with many compact parameterization schemes such as decision trees (Ernst et al., 2005), Gaussian kernels (Ormoneit & Sen, 2002; Smart & Kaelbling, 2000), or learned value eigenbasis (Mahadevan & Maggioni, 2007), to name a few.

Recent analysis on neural kernel regression (Neal, 1996; Jacot et al., 2018) uncovers a *spectral-bias* in multilayer perceptron networks, whose convergence slows down exponentially on high-frequency harmonics (Kawaguchi & Huang, 2019; Bietti & Mairal, 2019; Ronen et al., 2019). Although originally derived with shallow, infinitely-wide networks, such result has been extended to deeper networks at finite-width of any "reasonable" architecture via the *tensor program* (Yang, 2019; 2020; Yang & Littwin, 2021; Yang & Salman, 2019). This spectral bias for the neural tangent kernel to concentrate towards low-frequency bands produce unwanted generalization and cross-talks during gradient descent. We identify this spectral bias as the key issue, and address both the *expressivity gap* and *off-policy divergence*, by explicitly controlling the bias-variance trade-off via an a tunable, and adaptive Gaussian mixture kernel (Rahimi & Recht, 2008; Yang et al., 2015). Our solution involves changing just *a single line of code*, yet it achieves state-of-the-art sample efficiency and asymptotic performance on challenging continuous control domains with just a fraction of the compute needed by a vanilla perceptron network.

Our main contributions are twofold. First, we show that use of Fourier features enable faster convergence on high frequency spectral components during value approximation, thereby improving the sample efficiency of off-policy reinforcement learning. Second, we show that the improved neural tangent kernel is more *localized* with much smaller off-diagonal cross-talk. Combined with faster convergence, the improved off-policy stability enabled us to remove the target network on a few domains without suffering catastrophic divergence.

2 A Motivating Example

We motivate through a toy MDP adapted from Dong et al. (2020), and show that neural fitted Q iteration significantly underfits the optimal value function. Despite of the simple-looking reward and forward dynamics, the value function is quite complex. Our spectral analysis in Section 3.1 indicates that such complexity arises from the recursive application of the Bellman operator.

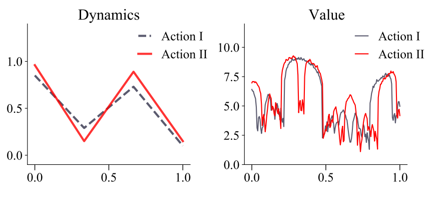
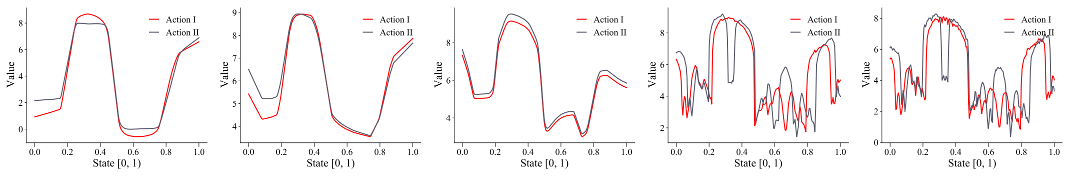


Figure 1: A Toy MDP with a simple forward dynamics and complex value function, adapted from Dong et al. (2020).

Toy MDP Distribution Consider a class of toy Markov decision processes M. The state space is defined on the real-line as $\mathcal{S} = \mathbb{R}^{[0,1)}$. The reward is the identity function. The action space is a discrete set with two elements $\{0, 1\}$, 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 $k = 10$, 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

$$M \sim \mathcal{M} = p(M). \tag{1}$$



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

Figure 2: Q-value approximation on the toy MDP. All baselines collected at 400 epochs unless otherwise mentioned. (a) Fitted Q iteration using a 4-layer MLP with 400 hidden neurons. (b) Supervised by the ground-truth value target, at 400 epochs. (c) using a larger network that is 3x deeper (12-layers). (d) the same 4-layer network, but optimized longer, for 2000 epochs. (e) is a 4-layer FFN of the same size optimized for 400 epochs, without a target network.

A curious failure of neural fitted Q iteration When we apply standard neural fitted Q iteration (FQI, see Riedmiller 2005b) to this toy problem using a simple four-layer multi-layer perceptron (MLP) with 400 hidden units at each layer, we observe that the learned MLP value approximator, produced in Figure 2a, significantly underfits in comparison to the ground-truth value function. One hypothesis is that Q learning is to blame, and in fact prior work such as Kumar et al. 2021 has argued that underfitting results from minimizing the TD (Temporal-Difference) error, because bootstrapping resembles iteratively applying self-distillation (Mobahi et al., 2020), which is known to lead to under-parameterization in the learned neural network features. To rule-out this hypothesis, we can use the same MLP architecture, but this time directly regress towards the ground truth Q function via supervised learning. Figure 2b shows that the under-fitting still persists under supervised learning, and increasing the depth to 12 layers (see Figure 2c) fails to fix the problem. This shows an over-parameterized network alone is insufficient to reduce under-fitting. Finally, if we increase the number of training iterations from 400 epochs used in these experiments to 2000, the original four-layer MLP attains a good fit. This shows that solely focusing on the expressivity of the neural network by making it bigger (Sinha et al., 2020) can be misguided, as the type of function that can be approximated also depends on the available budget over the number of gradient updates.

3 Background And Notations

We consider an agent learning to act in a Markov decision process (MDP), parameterized via 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 with the discount factor γ. The goal of the agent is to maximize the expected future discounted return $\mathcal{J} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$ by learning a policy $\pi(a|s)$ that maps a state s to a distribution over actions. The state-action value function (Q-function) is defined as $Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | (s_0, a_0) = (s, a)\right]$. The optimal $Q^*(s, a)$ is the fixed point of the Bellman optimality operator $\mathcal{B}^*$

$$\mathcal{B}^* Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)}[\max_{a*} Q(s', a^*)]. \tag{2}$$

The fitted Q iteration family of algorithms (Ernst et al., 2003; Riedmiller, 2005a) iteratively finds the optimal Q^* by recursively applying the gradient update

$$\theta' = \theta - \eta \nabla_{\theta} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left\| Q_{\theta}(s, a) - \mathcal{B}^* Q_{\theta}(s, a) \right\|_2^2 . \tag{3}$$

Letting $\boldsymbol{\mathcal{X}} = Q_{\theta}(s, a) - \mathcal{B}^* Q_{\theta}(s, a)$ and apply the chain-rule. The update 3 becomes

$$\theta' = \theta - 2\eta \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \Big[\boldsymbol{\mathcal{X}} \nabla_{\theta} Q_{\theta}(s, a) \Big] \tag{4}$$

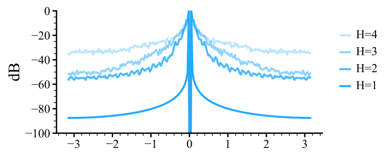
We can approximate the updated Q in function form through its first-order Taylor expansion

$$Q_{\theta'}(s, a) \approx Q_{\theta}(s, a) - 2\eta\, \mathbb{E}_{(s',a') \sim \mathcal{D}} \left[\boldsymbol{\mathcal{K}}(s, a; s', a') \boldsymbol{\mathcal{X}} \right] \tag{5}$$

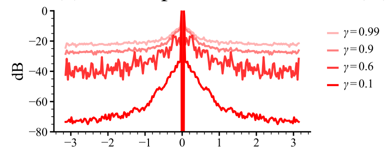
where the bilinear form $\boldsymbol{\mathcal{K}}(s, a; s', a') = \nabla_{\theta} Q_{\theta}(s, a)^T \nabla_{\theta} Q_{\theta}(s', a')$ is the *neural tangent kernel* (NTK, see Jacot et al. 2018 and Section 4). In the tabular case, this procedure can be simplified as producing a sequence $\{Q_0, Q_1, Q_2, \ldots\}$ using the iterative rule, $Q_{i+1} = \mathcal{B}^*(Q_i)$ starting with $Q_0(s, a) = R(s, a)$. The ith item in the sequence, Q_i, is the optimal Q function of a derived MDP with a shorter horizon $H = i$. Therefore each application of the Bellman optimality operator effectively extends the horizon by one time-step, starting with the 1-step reward $R(s, a)$.

3.1 Spectral Signature of the Toy MDP

The Bellman optimality operator imprints a non-trivial spectrum to the resulting Q_i as it is applied recursively during fitted Q iteration. We present the evolving spectra marginalized over $\mathcal{M}$ in Figure 3. As the effective horizon increases, the value function attains a larger weight in the higher-frequency part of the spectrum, which corresponds to less correlation between the value at nearby states. In a second experiment, we fix the horizon to 200 while increasing the discount factor from 0.1 all the way up to 0.99. We observe a similar increase in the higher frequency components, at a longer effective recursion depth. 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 "infinity mirror." The spectrum of the Bellman operator gets folded into the resulting Q function upon each iteration step. Although our analysis focuses on the state space, the same effect can be intuitively extrapolated to the joint state-action space.



(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.

3.2 Kernel View on Off-policy Divergence

Following the formulation of convergence in Dayan (1992); Tsitsiklis (1994); Jaakkola et al. (1994):

Definition: Consider a complete metric space S with the norm $\|\cdot\|$. An automorphism f on S is a *contraction* if $\forall a, b \sim S$, $\|f(a) - f(b)\| \leq \gamma \|a - b\|$. Here $\gamma \in [0, 1)$ is called the *contraction modulus*. When $\gamma = 1$, f is a *nonexpansion*.

Banach fixed-point theorem. Let S be non-empty with a contraction mapping f. Then f admits a unique fixed-point $x^* \in S$ *s.t.* $f(x^*) = x^*$. Furthermore, $\forall x_0 \in S$, x^* is the limit of the sequence given by $x_{i+1} = f(x_i)$. *a.k.a* $x^* = \lim\limits_{i \to \infty} x_i$.

Without lost of generality, we can discretize the state and action space S and A. The NTK becomes the gram matrix $\mathcal{K} \in \mathcal{R}^{|S||A| \times |S||A|}$. Transition data are sampled from a distribution $\rho(s, a)$.

Theorem. *(Achiam et al., 2019) Let indices i, j refer to state-action pairs. Suppose that $\mathcal{K}$, η and ρ satisfy the conditions:*

$$\forall i, 2\eta\mathcal{K}_{ii}\rho_i < 1, \tag{6}$$

$$\forall i, (1+\gamma)\sum_{j \neq i} |\mathcal{K}_{ij}|\rho_j \leq (1-\gamma)\mathcal{K}_{ii}\rho_i, \tag{7}$$

Then, Equation 5 induces a contraction on Q_θ in the sup norm, with fixedpoint Q_{θ^*} and the TD loss optimization converges with enough optimization steps.

For relatively large γ (for instance, $\gamma \in (0.99, 0.999)$), the above theorem implies that small off-diagonal terms in the NTK matrix are sufficient conditions for convergence.

4 Spectral-Bias and Neural Kernel Regression

Consider a simple regression problem where we want to learn a function $f(\xi; \theta) \in \mathbb{R}$. ξ is a sample from the dataset. To understand how the output of the network changes w.r.t small perturbations to the parameters, we can Taylor expand around θ

$$f(\xi; \theta + \delta\theta) - f(\xi; \theta) \approx \langle \nabla\theta f(\xi; \theta),\ \delta\theta \rangle. \tag{8}$$

During stochastic gradient descent using a training sample $\hat{\xi}$ with a loss function $\mathcal{L}$, the parameter update is given by the product between the loss derivative $\mathcal{L}' \circ f(\hat{\xi})$ and the *neural tangent kernel* $\mathcal{K}$ (Jacot et al., 2018)

$$\delta\theta = -\eta\mathcal{L}'(f(\hat{\xi}))\mathcal{K}(\xi, \hat{\xi}) \quad \text{where} \quad \mathcal{K}(\xi, \hat{\xi}) = \langle \nabla_\theta f(\xi; \theta), \nabla_\theta f(\hat{\xi}; \theta) \rangle. \tag{9}$$

In the infinite-limit with over-parameterized networks, the function remains close to initialization during training (Chizat et al., 2019). The learning dynamics in this "lazy" regime, under an L_2 regression loss behaves as a minimum norm least-square solution

$$f_t - f^* = e^{-\eta \mathcal{K} t(\xi,\hat{\xi})}(f_0 - f^*), \tag{10}$$

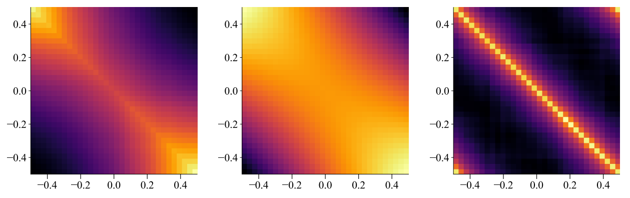
where f_t is the function under training at time t and f_0 is the neural network at initialization. Replacing $\mathcal{K}$ with the Gramian matrix $\boldsymbol{\mathcal{K}}$ between all pairs of the training data, we can re-write Equation 4 in its spectral form $\boldsymbol{\mathcal{K}} = \boldsymbol{O}\boldsymbol{\Lambda}\boldsymbol{O}^T$ where each entry in the diagonal matrix $\boldsymbol{\Lambda}$ is the eigenvalue $\lambda_i > 0$ for the basis function O_i in the orthogonal matrix $\boldsymbol{O}$. Using the identity $e^{\boldsymbol{A}} = \boldsymbol{O}e^{\boldsymbol{\Lambda}}\boldsymbol{O}^T$, we can decompose the learning dynamics in Equation 4 into

$$\boldsymbol{O}^T(f_t - f^*) = e^{-\eta\boldsymbol{\Lambda}t}\boldsymbol{O}^T(f_0 - f^*). \tag{11}$$

The key observation is that the convergence rate on the component O_i, $\eta\lambda_i$, depends exponentially on its eigenvalue λ_i. *a.k.a* the absolute error

$$|O_i^T(f_t - f^*)| = e^{-\eta\lambda_i t}|O_i^T(f_0 - f^*)| \tag{12}$$

Multiple work (Rahaman et al., 2019; Shah et al., 2020; Yang & Salman, 2019; Huh et al., 2021) have shown that the NTK spectrum (λ_i) of a regular ReLU network decays rapidly at higher frequency. In particular, Bietti & Mairal (2019) provided a bound of $\Omega(k^{-d-1})$ for the kth spherical harmonics[1]. Such results have been extended to finite-width networks of arbitrary architecture and depth via the *tensor programs* (Yang, 2019; 2020).



(a) MLP + ReLU (b) MLP + Tanh (c) FFN (Ours)

Figure 4: NTK comparison between (a) MLP with ReLU activation, (b) MLP with tanh activation, and (c) Fourier feature networks (Ours). The MLP NTK in (a,b) both contain large off-diagonal elements. This shows the addressing by the gradient vectors is not specific, causing divergence.

The Gramian matrix form of the NTK also offers an intuitive way to inspect state-aliasing during gradient descent. This is because the off-diagonal entries corresponds to the similarity between gradient vectors for different state-action pairs. The kernel of an multi-layer perceptron is not stationary when the input is not restricted to a hypersphere as in Lee et al. (2017). We can, however, compute $\boldsymbol{\mathcal{K}}$ of popular network architectures over $\mathbb{R}^{[0,1)}$, shown in Figure 4. The spectral-bias of the NTK of both ReLU networks and hyperbolic tangent networks produce large off-diagonal elements in the input space due to such bias, causing instability.

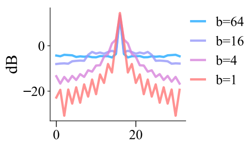
5 Overcoming the Spectral-Bias of Neural Value Approximation

Our basic idea is the following: To correct the spectral-bias of the perceptron network, we can constructing a composite kernel where a random map $\boldsymbol{z} : \mathbb{R}^d \mapsto \mathbb{R}^D$ first "*lifts*" the input into a randomized harmonics basis. This explicit kernel lifting trick was introduced by Rahimi & Recht (2007) and it allowed the authors to fit complex datasets using a linear kernel. The mixing brings high-frequency input signals down to a lower and more acceptable band for the perceptron network. Data also appears more sparse in the higher-dimensional spectral basis, further simplifies learning. To emulate arbitrary shift-invariant kernel $\mathcal{K}$, Rahimi & Recht (2007) offered a procedure that sample directly from the nominal distribution given by $\mathcal{K}$'s spectrum

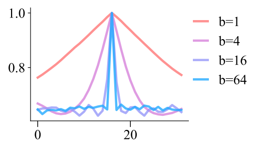
$$k(x,y) = \langle\phi(\xi),\phi(\hat{\xi})\rangle \approx \boldsymbol{z}(\xi)^T\boldsymbol{z}(\hat{\xi}) \text{ where } z(\xi) = \sum_i w_i e^{2\pi k_i} \text{ and } w_i \sim \mathscr{F}(\mathcal{K}). \tag{13}$$

We choose to initialize FFN by sampling the weights from an isotropic multivariate Gaussian with a tunable cutoff frequency b. We modify the normalization scheme from Rahimi & Recht (2007) and

[1]the input is restricted to a sphere. This bound is loose, and for a tighter bound refer to Cao et al. (2019).



(a) FFN kernel Spectrum



(b) NTK Cross-section

Figure 5: FFN provide direct control over generalization. (a) The kernel spectra. Peak in the center is due to windowing effect. Higher band-limit leads to flatter spectra. (b) The cross-section of the kernel at different band-limit.

Algorithm Learned Fourier Features (LFF)

```
class LFF(nn.Linear):
  def __init__(self, inp, out, b_scale):
    super().__init__(inp, out)
    init.normal_(self.weight, std=b_scale/inp)
    init.uniform_(self.bias, -1.0, 1.0)

  def forward(self, x):
    x = np.pi * super().forward(x)
    return torch.sin(x)
```

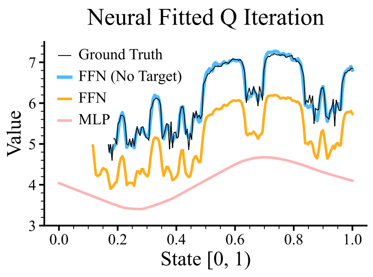
divide b with the input dimension d, *s.t.* similar bandwidth values could be applied across a wide variety of reinforcement learning problems with drastically different state and action space dimensions. We slightly abuse the notion by using b_j as the bias parameters, and b (without subscript) for the *bandwidth*

$$\text{RFF}(x)_j = \sin(\sum_{i=1}^{d} w_{i,j} x^i + b_j) \text{ where } w_{i,j} \sim \mathcal{N}(0, \pi b/d) \text{ and } b_j \sim \mathcal{U}(-\pi, \pi). \tag{14}$$

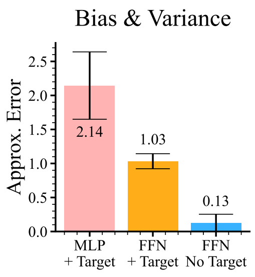
To attain better performance, Rahimi & Recht (2008) learns a *weighted sum* of these random kitchen sink features, whereas Yang et al. (2015) adapts the sampled spectral mixture itself through gradient descent. Using modern deep learning tool chain, we can view the entire network including the random Fourier features as a *learnable* kernel. For small neural networks with limited expressivity, we found that enabling gradient updates on the RFF parameters is important for performance. We refer to this adaptive variant as the *learned Fourier features* (LFF), and the shallow MLP with an adaptive Fourier feature expansion as the **Fourier feature networks** (FFN).

Improved NTK produce faster convergence The improved composite kernel leads to faster convergence by having a flat kernel spectrum all the way up to the band-limit. we can compute the kernel spectrum numerically and inspect the improvement in the higher frequencies (see Figure 5). By tuning the band-limit, we can make explicit *bias-variance trade-offs* and control the way the network generalizes. On the toy domain used to motivate this work FFN achieves a perfect fit with just 400 optimization epochs whereas the MLP baseline requires at least two thousand (see Figure 2e). The gain is even more prominent on more challenging environments in the results section (Quadruped run, see Figure 8). Optimization overhead is a key bottleneck in off-policy learning, therefore faster convergence also translates into better sample efficiency, and faster wallclock time.

Improving Off-policy Stability "cross-talks" between the gradients of near-by states is key factor in off-policy divergence. Such "cross-talk" manifest as similarity between gradient vectors, the measure of which is captured by the Gram matrix of the NTK. The Fourier feature network kernel is both *localized* (Figure 4c) and *tunable* 5), offering direct control over the bias-variance trade-off. The improved learning stability allows us to remove the target network on a number of domains while retaining substantial performance.



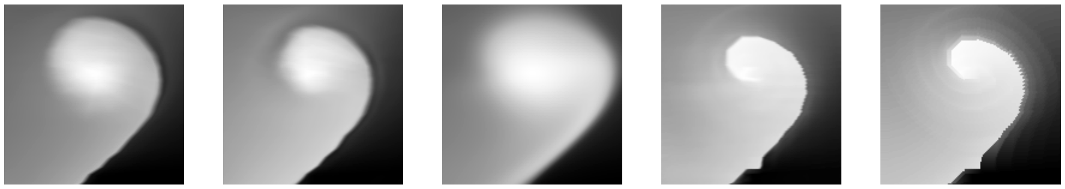
(a) Learned Q Function



(b) Bias Statistics

Figure 6: Removing the target network further reduces bias. (a) Comparison of the learned Q function (for action II). (b) FFN with a target network still contains an offset, whereas removing the target network eliminates such bias.

The Mountain Car environment (Moore, 1990; Sutton, 2000) has a simple, two-dimensional state space that can be directly visualized to show the learned value function. We compare the value estimate from three baselines In Figure 7: 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 FFN (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 deep reinforcement learning. (d) is from a 4-layer FFN. (e) the ground-truth produced by running value iteration using a value table. The two axis corresponds to position and velocity of the car.

6 EXPERIMENTS

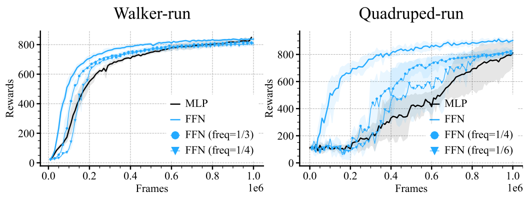
Setup We scale the use of FFN to high-dimensional continuous control tasks in state DMC domains (Tassa et al., 2020). We use soft actor critic (SAC) (Haarnoja et al., 2018) as our RL algorithm. We build off the pytorch SAC codebase (Yarats & Kostrikov, 2020). We derive the FFN from a MLP by replacing its first layer with a LFF layer. To use FFN with SAC, we replace the 3-layer MLP in the actor and the critic with a 3-layer FFN.

Results We present the result with SAC on 4 DMC domains in Figure 10. It shows that FFN improve over MLP in complex environments while matching its performance in simpler environments. We leave the complete results with SAC and DDPG in Appendix A.3. We also present ablations and hyperparameter sensitivity analysis in Appendix A.4. As described in section 5, the benefit of FFN in RL comes from its ability to estimate the value function of the task with fewer gradient updates, thereby improving the sample efficiency of the RL algorithm. We empirically verify this observation in Figure 11 where FFN reduces the value estimation error with more sample efficiency than a MLP. We leave the complete results on 8 DMC domains in the Appendix A.3.

6.1 FASTER CONVERGENCE VIA FOURIER FEATURE NETWORKS

We hypothesize FFN achieves faster convergence in neural value approximation due to its better initialization obtained by sampling from a wide space of frequency and phase. If this is indeed the case, then FFN's parameters should undergo less change from the training of an RL agent compared to MLP's parameters. Let $\mathcal{W}_t$ and $\mathcal{B}_t$ refer to the concatenated weight matrices (in vector form) and the concatenated bias vectors respectively, from different layers of critic of an RL agent after training on t environment frames (steps). Figure 12 shows evolution of $\|\mathcal{W}_t - \mathcal{W}_0\|$ and $\|\mathcal{B}_t - \mathcal{B}_0\|$ for MLP and FFN during training of SAC agents for *Walker-run* and *Quadruped-run*. We leave the complete results on 8 DMC domains in the Appendix A.3.

Since FFN estimates value function with fewer gradient updates, can we reduce the update frequency during training of an RL agent while still matching the performance of MLP? Figure 8 shows that we can reduce the update frequency by a factor 4 and 6 in *Walker-walk* and *Quadruped-run* using FFN



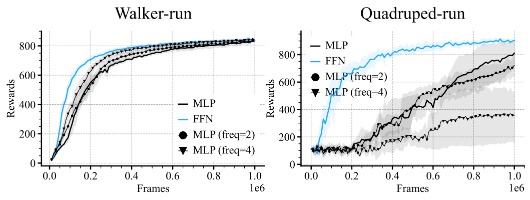


Figure 8: Learning curve showing the improved NTK needing only a fraction of the compute to match the performance by the original MLP on both Walker-run and Quadruped-run.

Figure 9: Learning curve showing increasing the optimization ratio improve MLP performance on Walker-run, but on Quadruped-run it leads to over-fitting, and degrades performance.

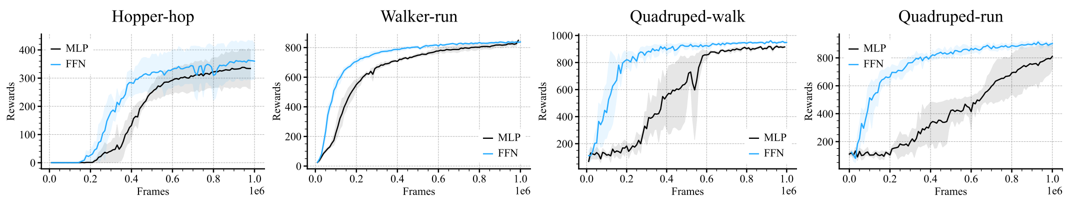


Figure 10: Learning curve with FFN applied to *SAC* on the DeepMind control suite. Domains are ordered by input dimension, showing an overall trend where domains with higher state dimension benefits more from Fourier features. We use a (Fourier) feature-input ratio of $40 : 1$.

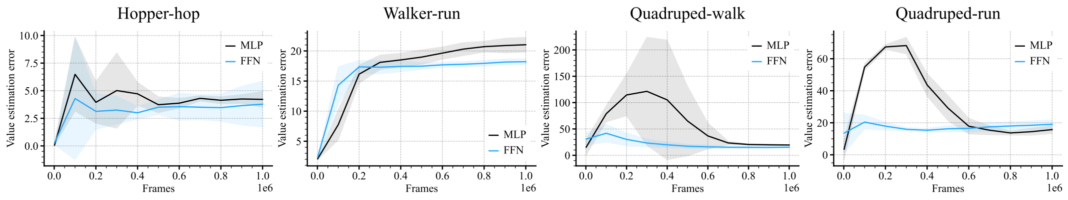


Figure 11: Value estimation error with FFN applied to *SAC* on the DeepMind control suite. FFN is more sample efficient in reducing value estimation error, thereby improving the sample efficiency of *SAC*. The divergence is especially prominet at the begining of learning when the data is sparse. FFN reduces the variance in these regimes by making more appropriate bias-variance trade-off.

while matching the performance of MLP. Thus, FFN can help achieve compute efficiency in RL. Conversely, can we improve the performance of MLP by increasing the update frequency? Figure 9 shows that increasing the update frequency can help MLP get closer to FFN in terms of performance in *Walker-walk*. However, this isn't reliable as it can also lead to overfitting in actor and hurt MLP's performance, as seen in *Quadruped-run*.

6.2 Fourier Features Improve Off-Policy Stability

We now analyze if use of FFN removes the need for a target network. Target (value) network (Mnih et al., 2013) is a past instantiation of value function which gets updated at a slower rate and is used for calculating targets during bellman update. Given MLPs often take many gradient updates to fit to a target, the slow changing target network stabilizes the training of the value function. However, since FFN requires fewer gradient updates to fit to a target, we hypothesize that there will be less of a need for a target network. Figure 13 shows performance of both MLP and FFN without a target network. While MLP completely fails in all environments, FFN either matches its performance with target networks or suffers small degradation in most environments except *Hopper-hop*. We leave the complete results on 8 DMC domains in the Appendix A.3.

7 Improving the Convolution Kernel

In a standard convolution neural network, the filter in the first layer suffers the same spectral bias in the RGB color-space as the state and action space above. We can in principle extend our spectral-bias fix to convolution networks by replacing the first layer with a 1×1 convolutional layer and a sinusoidal non-linearity. In this experiment, we use the same initialization scheme as those described in Equation 14. Our experiment is conducted with the DrQv2 implementation (Yarats et al., 2021), where we replace the CNN in both the actor and the critic with this Fourier-CNN (F-CNN) architecture. We present the results in Figure 14. While F-CNN's performance is within the variance of CNN's performance, its variance is much lower. Such technique has also been used by Kingma et al. (2021) to improve image generation with diffusion models. For more detailed theoretical overview of the CNN NTK, one can refer to Arora et al. (2019) and Li et al. (2019).

8 Discussion

The inspiration of this work comes from our realization that techniques used by the graphics community (Tancik et al., 2020; Sitzmann et al., 2020; Mildenhall et al., 2020) to learn high-fidelity continuous neural representations represent a new way to explicitly control generalization in neural networks. We refer to Achiam et al. (2019)'s pioneering work for the theoretical setup on off-policy

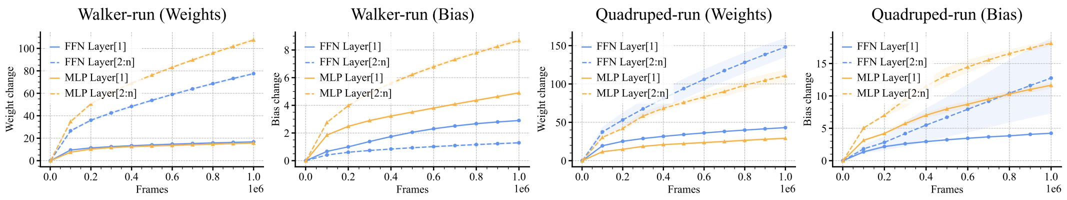


Figure 12: *Weight and bias changes* in FFN and MLP during training, using *SAC*. While FFN's bias parameters undergo less change than MLP's bias parameters, the results are mixed when it comes to weight parameters on the Quadruped environment.

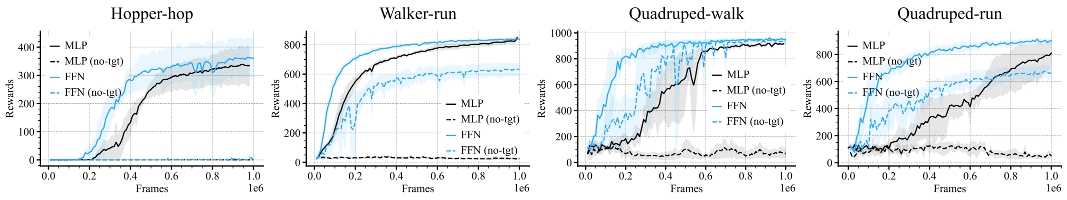


Figure 13: Learning curves showing that FFN improves learning stability to the extent that learning can happen on some domains even without the target network. On the contrary, MLP consistently fails without a target value network. FFN has consistently less variance than the MLP.

divergence, which predates many of the recent analysis on spectral-bias in neural networks, and the (re-)introduction of the random Fourier features as a solution to correct it. Function regularization through gradient conditioning is an alternative to re-parameterizing the network. A recent effort could be found in Piché et al. (2021). A key benefit of reparameterizing the network is speed. We can managed to reduce the walltime by 32% on the challenging Quadruped environment, by reducing the replay ratio to $1/6$.

Fourier features for reinforcement learning dates back as early as Konidaris et al. (2011). While this manuscript was under review, a few similar publication and preprints came out, all developed independently. Li & Pathak (2021) focuses on the smoothing effect that Fourier feature networks have on rejecting excessive noise. Our finding is that vanilla perceptron networks are on the biased-side of the bias-variance trade-off. In other words, the main issue in off-policy learning with neural function approximators is that the network *underfits* real signal, as opposed to being overfit to random noises. The rank of the neural representation describes the portion of the linear space occupied by the largest eigenvalues of the kernel regardless of the spatial frequency of those corresponding eigenfunctions. Therefore lower rank does not correspond to smoother functions. We additionally find that maintaining a Fourier feature to input feature ratio ($D/d > 40$) is critical to the expressiveness of the network, which allowed us to scale up to Quadruped without needing to concatenating the raw input as a crutch. Brellmann et al. (2022) is a concurrent submission to ours that delightfully includes the random tile encoding scheme from Rahimi & Recht (2007), and on-policy algorithm, PPO. Our intuition is that policy gradient needs to be considered a life-long learning problem, and *locality* in the policy model could speed up learning by eliminating the need to repeatedly sample areas the policy has experienced. We are excited about these efforts from the community, and urge the reader to visit them for diverse treatments and broader perspectives.

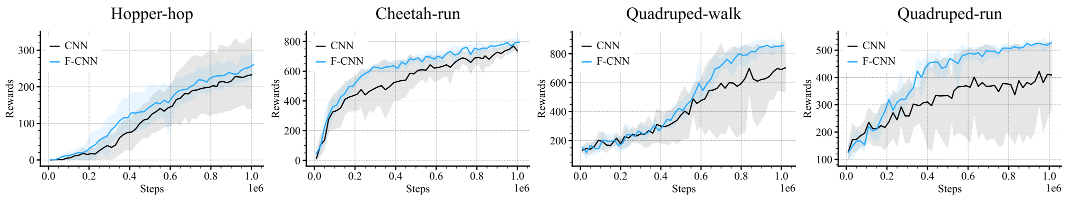


Figure 14: Control from pixels learning curve on the DeepMind control suite using Fourier-CNN (F-CNN) and *DrQv2*. We use a feature-input ratio of $40 : 1$. Performance with the F-CNN has much lower variance and is consistently at the top of the confidence range of the vanilla CNN.

Acknowledgments

The authors want to thank Jim Halverson and Dan Roberts at IAIFI for fruitful discussions over neural tangent kernels, and its connection to off-policy divergence; Aviral Kumar at UC Berkeley for bringing into our attention the expressivity issues with off-policy value approximation; and the reviewer and area chair for their kind feedback.

This work is supported in part by the Army Research Office under Grant W911NF-21-1-0328; the National Science Foundation Institute for Artificial Intelligence and Fundamental Interactions (IAIFI, https://iaifi.org/) under the Cooperative Agreement PHY-2019786; and the MIT-IBM Watson AI Lab and Research Collaboration Agreement No.W1771646. The authors also acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing high performance computing resources. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Reproducibility Statement

We include detailed implementation details in Appendix A.3.

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

S Arora, S S Du, W Hu, Z Li, R Salakhutdinov, and others. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv*, 2019.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pp. 30–37. Elsevier, 1995.

Dimitri P Bertsekas. A counterexample to temporal differences learning. *Neural Comput.*, 7(2): 270–279, March 1995. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1995.7.2.270.

Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. *Adv. Neural Inf. Process. Syst.*, 32, 2019. ISSN 1049-5258.

David Brellmann, Goran Frehse, and David Filliat. Fourier features in reinforcement learning with neural networks, 2022. URL `https://openreview.net/forum?id=VO7bAwdWRjg`.

Abdulkadir Canatar, Blake Bordelon, and Cengiz Pehlevan. Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks. *Nat. Commun.*, 12(1):2914, May 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-23103-1.

Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning. December 2019.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Adv. Neural Inf. Process. Syst.*, 32, 2019. ISSN 1049-5258.

Peter Dayan. The convergence of td(λ) for general λ. *Mach. Learn.*, 8(3-4):341–362, May 1992. ISSN 0885-6125, 1573-0565. doi: 10.1007/bf00992701.

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. Iteratively extending time horizon reinforcement learning. In *Machine Learning: ECML 2003*, pp. 96–107. Springer Berlin Heidelberg, 2003. doi: 10.1007/978-3-540-39857-8_11.

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

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. pp. 1587–1596, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Hado Hasselt. Double q-learning. volume 23, pp. 2613–2621, 2010.

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.

Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

Kenji Kawaguchi and Jiaoyang Huang. Gradient descent finds global minima for generalizable deep neural networks of practical sizes, 2019.

Diederik P Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. July 2021.

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.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

Kimin Lee, Michael Laskin, A. Srinivas, and P. Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *ICML*, 2021.

Alexander Li and Deepak Pathak. Functional regularization for reinforcement learning via learned fourier features. *Adv. Neural Inf. Process. Syst.*, 34, 2021. ISSN 1049-5258.

Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. November 2019.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3):293–321, May 1992. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00992699.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *J. Mach. Learn. Res.*, 8(Oct):2169–2231, 2007. ISSN 1532-4435, 1533-7928.

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.

Marvin Minsky. Steps toward artificial intelligence. *Proc. IRE*, 49(1):8–30, January 1961. ISSN 0096-8390. doi: 10.1109/JRPROC.1961.287775.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

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.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 1996.

Gerhard Neumann, Jan Peters, et al. Fitted q-iteration by advantage weighted regression. In *Advances in Neural Information Processing Systems 21-Proceedings of the 2008 Conference*, pp. 1177–1184, 2009.

Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Mach. Learn.*, 49(2):161–178, November 2002. ISSN 0885-6125, 1573-0565. doi: 10.1023/A:1017928328829.

Alexandre Piché, Joseph Marino, Gian Maria Marconi, Christopher Pal, and Mohammad Emtiyaz Khan. Beyond target networks: Improving deep q-learning with functional regularization. June 2021.

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.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Adv. Neural Inf. Process. Syst.*, 20, 2007. ISSN 1049-5258.

Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. *Adv. Neural Inf. Process. Syst.*, 21, 2008. ISSN 1049-5258.

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, 2005a.

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, 2005b. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

Basri Ronen, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. In H Wallach, H Larochelle, A Beygelzimer, F dAlché-Buc, E Fox, and R Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

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.

William D Smart and Leslie Pack Kaelbling. Practical reinforcement learning in continuous spaces. In *ICML*, pp. 903–910, 2000.

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

Richard S Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1): 9–44, August 1988. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00115009.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, October 2018. ISBN 9780262352703.

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.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm$_control : Software and tasks for continuous control$, 2020.

Gerald Tesauro. Practical issues in temporal difference learning. *Adv. Neural Inf. Process. Syst.*, 4, 1991. ISSN 1049-5258.

Richard H Thaler. Anomalies: The winner's curse. *Journal of economic perspectives*, 2(1):191–202, 1988.

John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Mach. Learn.*, 16(3):185–202, September 1994. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00993306.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Greg Yang. Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes. October 2019.

Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. June 2020.

Greg Yang and Etai Littwin. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11762–11772. PMLR, 2021.

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

Zichao Yang, Andrew Wilson, Alex Smola, and Le Song. A la carte–learning fast kernels. In *Artificial Intelligence and Statistics*, pp. 1098–1106, 2015.

Denis Yarats and Ilya Kostrikov. Soft actor-critic (sac) implementation in pytorch. https://github.com/denisyarats/pytorch_sac, 2020.

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

A Appendix

A.1 Comparing Various Fourier Features

Using 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
$$\mathrm{MLP}(x) = f^t \circ \mathrm{ReLU} \circ \cdots f^2 \circ \mathrm{ReLU} \circ f^1(x)$$
- **Fourier features network (FFN) (Ours)** uses sine activation with random phase-shift, to replace the first layer of the network with learned Fourier features
$$\mathrm{LFF}(x) = \sin(Wx + c), \quad W_{i,j} \sim \mathcal{N}(0, \pi b/d), c_i \sim \mathcal{U}(-\pi, \pi)$$
so that the Fourier features network (FFN) is
$$\mathrm{FFN}(x) = f^t \circ \mathrm{ReLU} \circ \cdots f^2 \circ \mathrm{LFF}(x)$$
- **RFF (Tancik et al., 2020)** that uses sine and cosine pairs concatenated together
$$\mathrm{RFF}_{\text{Tancik}}(x) = [\sin(2\pi Wx), \cos(2\pi Wx)], \quad W_{i,j} \sim \mathcal{N}(0, \sigma^2)$$
- **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
$$\mathrm{SIREN}(x) = \mathit{lff}^t \circ \cdots \mathit{lff}^2 \circ \mathit{lff}^1(x).$$
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.

All the above Fourier feature variants are equivalent and should produce similar results when bandwidth is tuned properly.

A.2 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.3 Results on DMC domains

DMC domains We list the 8 DMC domains along with their observation space dimension and action space dimension in Table 1. We also list the optimal bandwidth b used by FFN for each of these envs.

Results with SAC and DDPG We scale up use of FFN to 8 DMC domains by replacing the MLP in the actor and the critic of a SAC agent with FFN. Figure 15 shows learning curve of FFN and MLP with SAC as the RL algorithm on 8 DMC domains. To show that FFN also works with other RL algorithms, we replace the MLP in the actor and the critic of a DDPG agent with FFN. Figure 16 show learning curve of FFN and MLP with DDPG as the RL algorithm on 8 DMC domains.

Table 1: DMC domains in increasing order of state space and action space dimensionality

Name	Observation space	Action space	Bandwidth b
Acrobot-swingup	Box(6,)	Box(1,)	0.003
Finger-turn-hard	Box(12,)	Box(2,)	0.001
Hopper-hop	Box(15,)	Box(4,)	0.003
Cheetah-run	Box(17,)	Box(6,)	0.001
Walker-run	Box(24,)	Box(6,)	0.001
Humanoid-run	Box(67,)	Box(21,)	0.001
Quadruped-walk	Box(78,)	Box(12,)	0.0003
Quadruped-run	Box(78,)	Box(12,)	0.0001

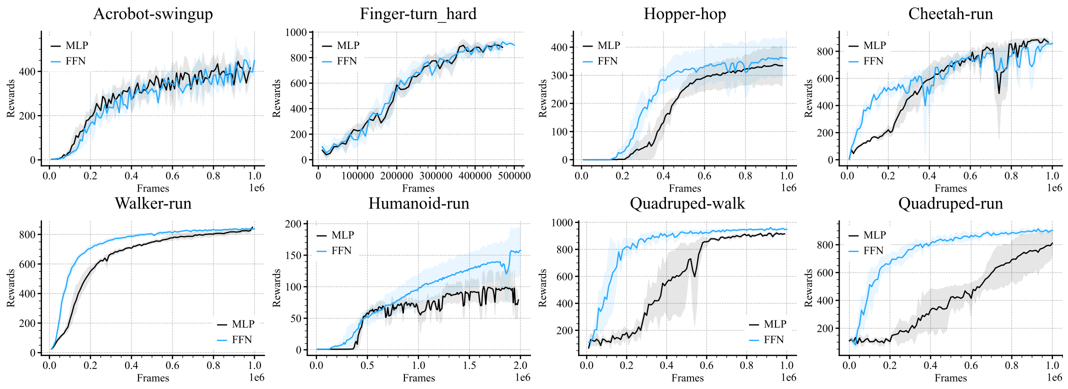


Figure 15: Learning curve with FFN applied to *SAC* on the DeepMind control suite. Domains are ordered by input dimension, showing an overall trend where domains with higher state dimension benefits more from Fourier features. We use a (fourier) feature-input ratio of $40 : 1$.

Value approximation error FFN improves over MLP both in sample efficiency and the overall performance because of its ability to fit value functions with less gradient updates. Figure 17 shows that FFN reduces value approximation error with more sample efficiency than MLP.

Measuring weight and bias change in FFN and MLP We hypothesize that FFN fits value functions with more sample efficiency than MLP because of its better initialization due to sampling from a wide space of frequency and phase. If this is indeed the case, the weights and biases of FFN should undergo less change than that of an MLP during training of an RL agent. Let $\mathcal{W}_t$ refer to the concatenated weight matrices (in vector form) from different layers of critic of an RL agent after training on t environment frames (steps). Similarly, let $\mathcal{B}_t$ refer to the concatenated bias vectors from different layers of critic of an RL agent after training on t environment frames. Figure 18 shows evolution of $\|\mathcal{W}_t - \mathcal{W}_0\|$ for MLP and FFN during training of SAC agents. Similarly, Figure 19 shows evolution of $\|\mathcal{B}_t - \mathcal{B}_0\|$ for MLP and FFN during training of SAC agents.

Removing the target network Since FFN is able to fit value function with less gradient updates, there is less of a need for a target network. Figure 20 shows performance of both MLP and FFN without a target network. While MLP completely fails in all environments, FFN either matches its performance with target networks or suffers small degradation in most environments. However, there are some environments (Hopper-hop, Humanoid-run, Acrobat-swingup) where FFN completely fails without target network.

Architectural Details for state DMC domains For MLP baseline, we parameterize the actor and the critic with 3 layer MLP. For our method, we replace the 3 layer MLP in the actor and the critic with a 3 layer FFN. If d is the input dimension, both MLP and FFN have $[40 \times d, 1024, 1024]$ as hidden dimension for each of their layers. Note that $d =$ observation dimension for actor and $d =$ observation dimension $+$ actor dimension for critic.

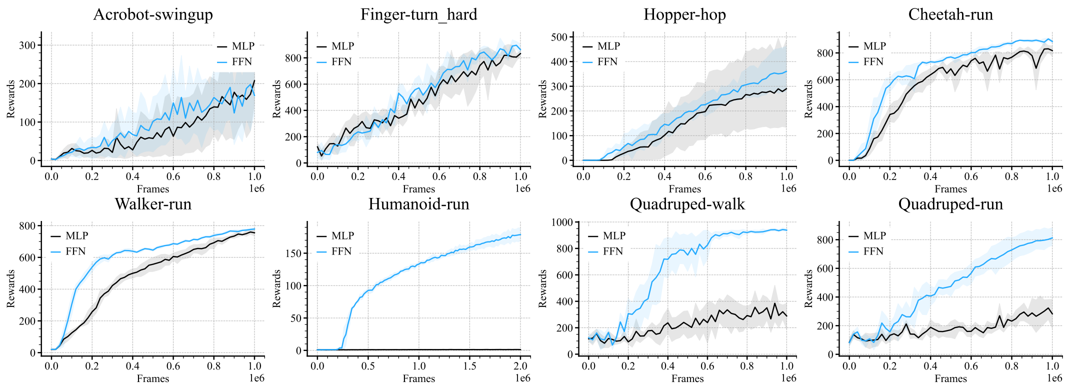


Figure 16: Learning curve with FFN applied to *DDPG* on the DeepMind control suite. Domains are ordered by input dimension. The over trend agrees with results on soft actor-critic, where domains with higher state dimension benefits more from random Fourier features. The same feature-input ratio of 1 : 40 is applied.

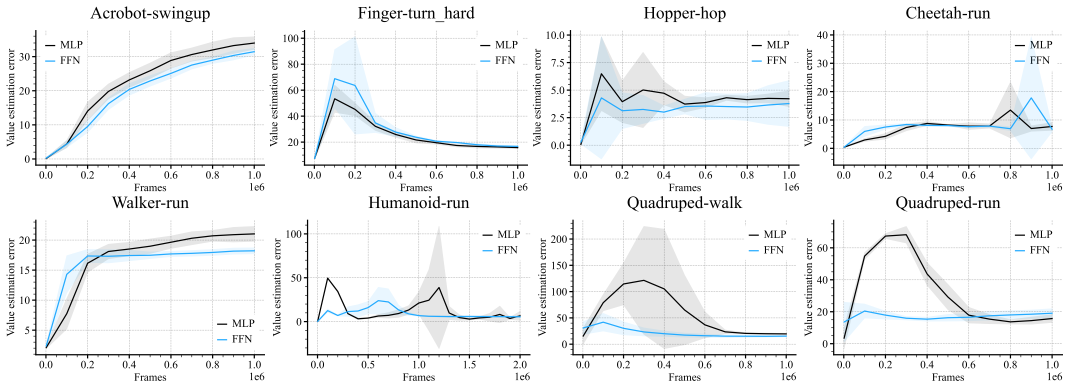


Figure 17: Value estimation error with FFN applied to *SAC* on the DeepMind control suite. FFN is more sample efficient in reducing value estimation error, thereby improving the sample efficiency of *SAC*.

Architectural Details for image DMC domains For CNN baseline, we parameterize the actor and the critic with CNN model borrowed from DrQv2 (Yarats et al., 2021). For our method, we derive conv FFN from the CNN model (taken from DrQv2) by replacing the first convolutional layer with a 1x1 convolutional layer having sin activation and using the initialization scheme described in Equation 14.

A.4 Ablations and Hyperparameter sensitivity analysis

Sensitivity to bandwidth b Since FFN introduces a bandwidth hyperparameter b, it is natural to ask how the choice of b affects FFN's performance. Figure 21 shows that FFN's performance does vary with choice of b. Furthermore, the best performing value of b differs with environment. This difference arises from the fact that the optimal value function for different environments have different spectral bias and hence requires different bandwidth b for FFN.

Sensitivity to (Fourier) feature-input ratio In addition to the bandwidth b, we need to choose the Fourier dimension for FFN. We maintained the feature-input ratio (i.e. $\frac{D}{d}$) to be 40. But can we get away with a lower feature-input ratio? Figure 22 shows that it is important to maintain feature-input ratio to be at least 40 and any reduction in the feature-input ratio hurts the performance of FFN.

FFN on actor vs FFN on critic In our experiments, we used FFN with both actor and critic. We analyze how removing FFN from either actor or critic affects the performance of RL agent. Figure 23

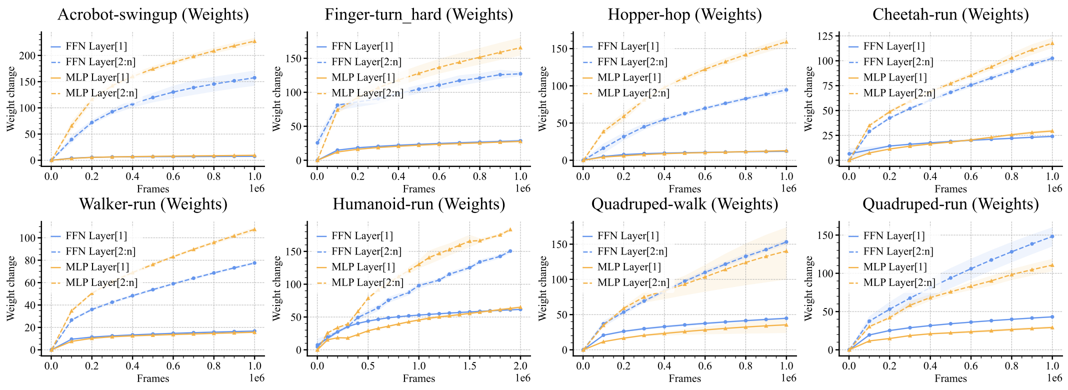


Figure 18: *Weight change* in FFN and MLP during training of RL agents with *SAC* on the DeepMind control suite. The results are mixed and FFN's weight parameters undergo less change than MLP's weight parameters only in some environments.

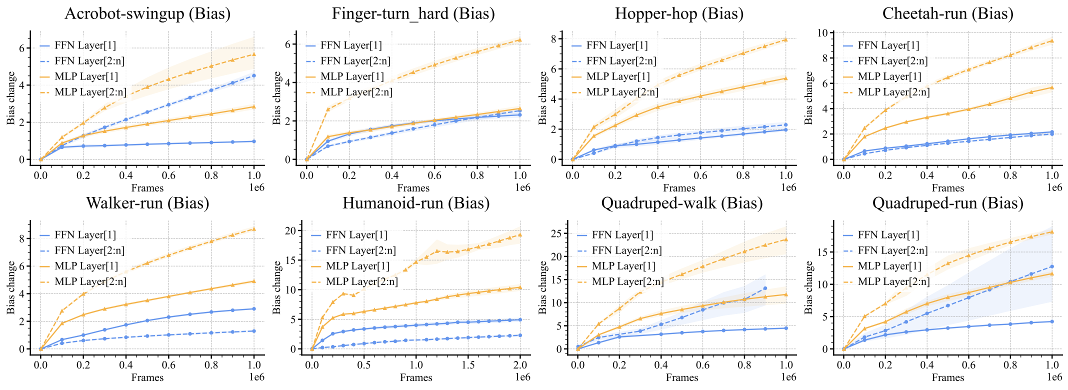


Figure 19: *Bias change* in FFN and MLP during training of RL agents with *SAC* on the DeepMind control suite. Given FFN's bias parameters have better initialization, they undergo less change than MLP's bias parameters.

shows that using FFN only critic doesn't affect the performance of the RL agent. However, using FFN with only actor brings down the performance of the RL agent to that using MLP for both actor and critic.

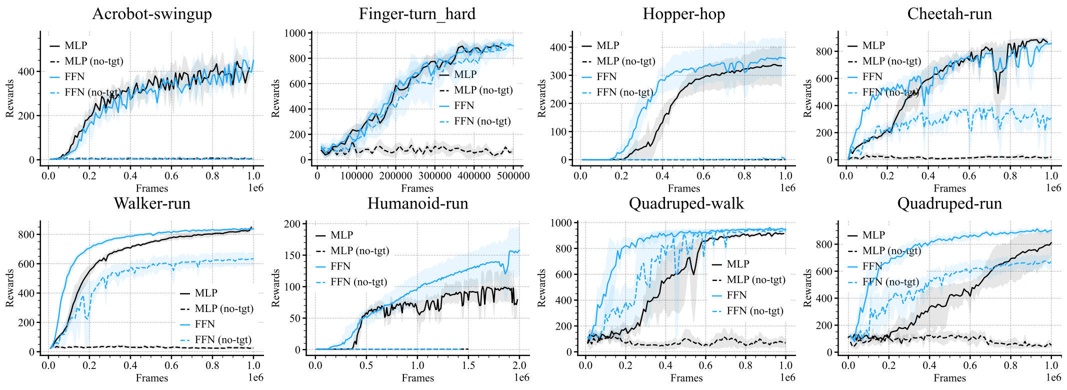


Figure 20: Since FFN require fewer gradients for value function estimation, its performance doesn't degrade as much when target value networks are removed. On the contrary, MLP completely fails when target value networks are removed.

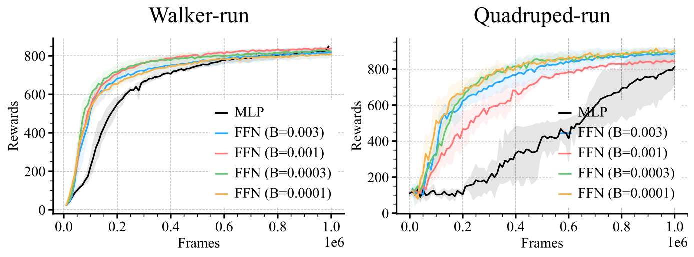


Figure 21: Learning curve with FFN for different values of B on Walker-run and Quadruped-run. We use *SAC* as the RL algorithm. We observe that different B values lead to different performances and hence, we must choose the B value carefully for each environment.

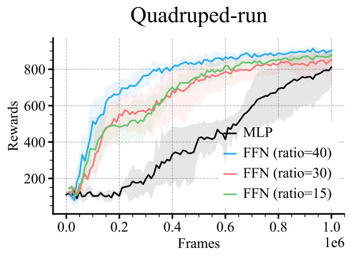


Figure 22: Learning curve with different Fourier dimension ratio on Quadruped-run. Lowering the ratio decreases the performance.

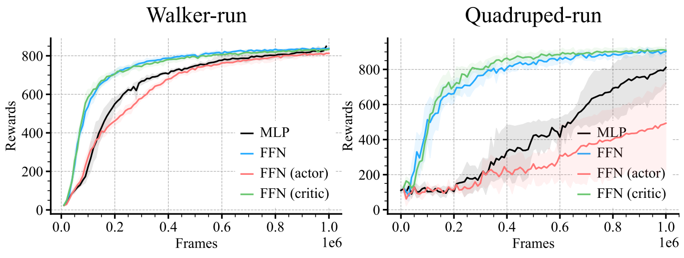


Figure 23: Using FFN for critic is as good as using FFN for both actor and critic. However, using learned FFN only for actor is similar to the MLP baseline. This indicates the gain mainly comes from better value approximation.