Acknowledgments  We thank Shikhar Bahl, Murtaza Dalal, Wenlong Huang, and Aravind Sivakumar for comments on early drafts of this paper. The work was supported in part by NSF IIS-2024594 and GoodAI Research Award. AL is supported by the NSF GRFP. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE1745016 and DGE2140739. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix A
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A

3. If you ran experiments...

...
Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code is provided in the supplemental material.

Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Hyperparameters are listed in Section 5 and Appendix H.

Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]

If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]
A Theoretical Analysis

A.1 Functional Convergence Rate

We provide Lemma 2 and a quick proof sketch as background for readers who are not familiar with the neural tangent kernel literature.

**Lemma 2.** When training an infinite width neural network via gradient flow on the squared error, the training residual at time $t$ is:

$$f_{\theta_t}(x) - y = e^{-\eta K t} (f_{\theta_0}(x) - y)$$

(11)

where $f_{\theta_t}(x)$ is the column vector of model predictions for all $x_i$, $y$ is the column vector of stacked training labels, $\eta$ is the multiplier for gradient flow, and $K$ is the NTK kernel matrix with $K_{ij} = \langle \nabla_{\theta} f_{\theta_0}(x_i), \nabla_{\theta} f_{\theta_0}(x_j) \rangle$.

**Proof.** The squared error $L(\theta, x) = \|f_{\theta}(x) - y\|^2_2$ has gradient:

$$\nabla_{\theta} L(\theta, x) = \nabla_{\theta} f_{\theta}(x)^\top (f_{\theta}(x) - y)$$

(12)

Since we train with gradient flow, the parameters change at rate:

$$\frac{d\theta_t}{dt} = -\eta \nabla_{\theta_t} L(\theta_t, x)$$

(13)

$$= -\eta \nabla_{\theta_t} f_{\theta_t}(x)^\top (f_{\theta_t}(x) - y)$$

(14)

By the chain rule,

$$\frac{df_{\theta_t}(x)}{dt} = \frac{df_{\theta_t}(x)}{d\theta_t} \frac{d\theta_t}{dt}$$

(15)

$$= -\eta \nabla_{\theta_t} f_{\theta_t}(x) \nabla_{\theta_t} f_{\theta_t}(x)^\top (f_{\theta_t}(x) - y)$$

(16)

$$= -\eta K_{\theta_t}(f_{\theta_t}(x) - y)$$

(17)

where $K$ is the NTK kernel matrix at time $t$, with entries $K_{ij} = \langle \nabla_{\theta} f_{\theta_0}(x_i), \nabla_{\theta} f_{\theta_0}(x_j) \rangle$. Since $y$ is a constant, and $K_{\theta_t} \approx K_{\theta_0} \triangleq K$ due to the infinite-width limit, we can write this as:

$$\frac{d(f_{\theta_t}(x) - y)}{dt} = -\eta K (f_{\theta_t}(x) - y)$$

(18)

This is a well-known differential equation, with closed form solution:

$$f_{\theta_t}(x) - y = e^{-\eta K t} (f_{\theta_0}(x) - y)$$

(19)

A.2 Eigenvalues of the NTK matrix and the Discrete Fourier Transform

**Lemma 3.** A circulant matrix $C \in \mathbb{R}^{n \times n}$ with first row $(c_0, \ldots, c_{n-1})$ has eigenvectors $(x^{(k)})_{k=1}^n$ corresponding to the column vectors of the DFT matrix:

$$x^{(k)} = (\omega_n^0, \omega_n^1, \ldots, \omega_n^{(n-1)k})^\top$$

(20)

where $\omega_n = e^{\frac{2\pi i}{n}}$ is the $n$th root of unity. The corresponding eigenvalue is the $k$th DFT value $\lambda^{(k)} = \text{DFT}(c_0, \ldots, c_{n-1})_k$.

**Proof.** This is a well-known property of circulant matrices [3]. Nevertheless, we provide a simple proof here. First, let’s make clear the structure of the circulant matrix $C$:

$$C = \begin{bmatrix}
    c_0 & c_1 & c_2 & \ldots & c_{n-1} \\
    c_{n-1} & c_0 & c_1 & \ldots & c_{n-2} \\
    c_{n-2} & c_{n-1} & c_0 & \ldots & c_{n-3} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    c_1 & c_2 & c_3 & \ldots & c_0 \\
\end{bmatrix}$$

(21)
Again, we want to show that an eigenvector of $C$ is $x^{(k)}$, the $k$th column of the DFT matrix $F$.

$$x^{(k)} = (\omega_n^{0k}, \omega_n^{1k}, \ldots, \omega_n^{(n-1)k})^\top$$

(22)

where $\omega_n = e^{\frac{2\pi i}{n}}$ is the $n$th root of unity. Note that for all positive integers $k$, $\omega_n^k$ is periodic in $j$ with period $n$. This is because:

$$\omega_n^k = e^{\frac{2\pi ik}{n}} = \cos(2\pi k) + i\sin(2\pi k)$$

(23)

$$= 1$$

(24)

(25)

Now, let us show that $x^{(k)}$ is an eigenvector of $C$. Let $y = Cx^{(k)}$. The $i$th element of $y$ is then

$$y_i = \sum_{j=0}^{n-1} c_j \omega_{n}^{jk}$$

(26)

$$= \omega_{n}^{nk} \sum_{j=0}^{n-1} c_j \omega_{n}^{(j-i)k}$$

(27)

The remaining sum does not depend on $i$, since $c_j$ and $\omega_n^{(j-i)k}$ are periodic with period $n$. This means we can rearrange the indices of the sum to get:

$$y_i = \omega_{n}^{nk} \sum_{j=0}^{n-1} c_j \omega_{n}^{jk}$$

(28)

$$= x^{(k)}_i \lambda_k$$

(29)

where $\lambda_k = \sum_{j=0}^{n-1} c_j \omega_n^{jk}$ is exactly the $k$th term in the DFT of the signal $(c_0, c_1, \ldots, c_{n-1})$. Thus, $Cx^{(k)} = \lambda_k x^{(k)}$, so $x^{(k)}$ is an eigenvector of $C$ with corresponding eigenvalue $\lambda^{(k)} = \text{DFT}(c_0, \ldots, c_{n-1})_k$.

A.3 Proof of Lemma 1: NTK for 2 layer Fourier feature model

Proof. The gradient consists of two parts: the gradient with respect to $B$ and the gradient with respect to $W$. We can calculate the NTK for each part respectively and then sum them.

For $W$:

$$\nabla_W f(x) = \sqrt{\frac{2}{m}} \left[ \frac{\sin(Bx)}{\cos(Bx)} \right]$$

(30)

The width-$m$ kernel is then:

$$k_m^W(x, x') = \frac{2}{m} \sum_{i=1}^{m/2} \cos(b_i^\top x) \cos(b_i^\top x') + \sin(b_i^\top x) \sin(b_i^\top x')$$

(31)

Using the angle difference formula, this reduces to:

$$k_m^W(x, x') = \frac{2}{m} \sum_{i=1}^{m/2} \cos(b_i^\top (x - x'))$$

(32)

As $m \to \infty$, this converges to a deterministic kernel $k^W(x, x') = \mathbb{E}_{B_{i,j} \sim \mathcal{N}(0, \Sigma)}[\cos(b_i^\top (x - x'))]$.

Using the fact that $E_{B \sim \mathcal{N}(0, \Sigma)}[\cos(t^\top X)] = \exp\{-\frac{1}{2}t^\top \Sigma t\}$ for fixed vector $t$ and the fact that $\Sigma = \text{diag}(\sigma^2, \sigma^2)$ in our case, this kernel function simplifies to:

$$k^W(x, x') = \exp\left\{-\frac{\sigma^2}{2} ||x - x'||_2^2\right\}$$

(33)
For $B$:

$$\nabla_{b_i} f(x) = \sqrt{\frac{2}{m}} (W_i \cos(b_i^T x) - W_{i+m/2} \sin(b_i^T x)) x$$  \hspace{1cm} (34)

The width-$m$ kernel for $B$ is then:

$$k_m^B(x, x') = \frac{2x^T x'}{m} \sum_{i=1}^{m/2} W_i^2 (\cos(b_i^T x) \cos(b_i^T x')) + W_{i+m/2}^2 (\sin(b_i^T x) \sin(b_i^T x'))$$

$$- W_i W_{i+m/2} (\cos(b_i^T x) \sin(b_i^T x') + \sin(b_i^T x) \cos(b_i^T x'))$$  \hspace{1cm} (35)

As we take $m \to \infty$, recall that $W_j \sim \mathcal{N}(0, 1)$ i.i.d.. Thus, $\mathbb{E}[W_j^2] = 1$ and $\mathbb{E}[W_j W_k] = 0$ for $j \neq k$. The NTK is then

$$k^B(x, x') = x^T x' \mathbb{E} [\cos(b_i^T x) \cos(b_i^T x') + \sin(b_i^T x) \sin(b_i^T x)]$$  \hspace{1cm} (36)

The interior of the expectation is exactly the same as the summand in Equation 31. Following the same steps, we get the simplified kernel function:

$$k^B(x, x') = x^T x' \exp \left\{ -\frac{\sigma^2}{2} \|x - x'\|_2^2 \right\}$$  \hspace{1cm} (37)

Finally, our overall kernel function $k(x, x') = k^W(x, x') + k^B(x, x')$ is:

$$k(x, x') = (1 + x^T x') \exp \left\{ -\frac{\sigma^2}{2} \|x - x'\|_2^2 \right\}$$  \hspace{1cm} (38)

Since $x, x' \in \mathbb{R}^{d-1}$, they have unit norm, with $\|x - x'\|_2^2 = 2(1 - x^T x') = 2(1 - \cos \theta)$. This gives us two equivalent forms of the NTK:

$$k(x, x') = \left(2 - \frac{\|x - x'\|_2^2}{2}\right) \exp \left\{ -\frac{\sigma^2}{2} \|x - x'\|_2^2 \right\}$$  \hspace{1cm} (39)

$$= (1 + \cos \theta) \exp \left\{ \frac{\sigma^2}{2} (\cos \theta - 1) \right\}$$  \hspace{1cm} (40)

\section*{A.4 When is the Bellman Update a Contraction?}

Here, we examine LFF’s stability under Bellman updates by using results from Achiam et al. \cite{Achiam2017}, who used the NTK approximation to prove that the Bellman update is a contraction in finite MDPs if

$$\forall i, \quad \alpha K_{ii} \rho_i < 1$$  \hspace{1cm} (41)

$$\forall i, \quad (1 + \gamma) \sum_{j \neq i} |K_{ij}| \rho_j \leq (1 - \gamma) K_{ii} \rho_i$$  \hspace{1cm} (42)

where $K$ is the NTK of the Q-network and $\rho_i$ is the density of transition $i$ in the replay buffer. Intuitively, $K_{ij}$ measures the amount that a gradient update on transition $j$ affects the function output on transition $i$. In order for the Bellman update to be a contraction, the change at $(s_i, a_i)$ due to gradient contributions from all other transitions should be relatively small. This suggests that LFF, with very large $\sigma$, could fulfill the conditions in Equation 43 and 44. We formalize this in Theorem 1:

\textbf{Theorem 1.} For a finite MDP with the state-action space as a finite, uniform subset of $S^d$, and when we have uniform support for each transition in the replay buffer, the Bellman update on a 2 layer LFF architecture is a contraction for suitably small learning rate $\alpha$.

\textbf{Proof.} We need our kernel to satisfy two conditions \cite{Achiam2017}:

$$\forall i, \quad \alpha K_{ii} \rho_i < 1$$  \hspace{1cm} (43)

$$\forall i, \quad (1 + \gamma) \sum_{j \neq i} |K_{ij}| \rho_j \leq (1 - \gamma) K_{ii} \rho_i$$  \hspace{1cm} (44)
Equation 43 is easy to satisfy, as we assumed that all transitions appear uniformly in our buffer, and we know from Lemma 1 that \( k(x, x) = 2 \). Thus, we only need to make the learning rate \( \alpha \) small enough such that \( \alpha < \frac{1}{\|x\|^2} \).

For Equation 44, we can prove a loose lower bound on the variance \( \sigma^2 \) that is required for the Bellman update to be a contraction. As stated in the main text, we assume that we have \( N + 1 \) datapoints \( x_i \) that are distributed approximately uniformly over \( S^{d-1} \). Here, uniformly simply implies that we have an upper bound \( x_i^T x_j = \cos \theta < 1 - \delta \) for all \( i \neq j \) and fixed positive \( \delta > 0 \).

First, we bound \( |K_{ij}| \) for all \( i \neq j \). Using the expression from Lemma 1 and our upper bound on \( \cos \theta \), we have \( |K_{ij}| \leq (2 - \delta) \exp \{-\delta \sigma^2\}, \forall i \neq j \). Then, plugging this into Equation 44 and cancelling the buffer frequencies \( \rho_i \), which are equal by assumption,

\[
N(1 + \gamma)(2 - \delta) \exp \{-\delta \sigma^2\} \leq 2(1 - \gamma)
\]

Rearranging gives us:

\[
\sigma^2 \geq \frac{1}{\delta \log \frac{N(1 + \gamma)(2 - \delta)}{2(1 - \gamma)}}
\]

As long as \( \alpha \) is small enough and \( \gamma < 1 \), an infinite-width LFF architecture initialized with a suitably large \( \tau \) will enjoy Bellman updates that are always contractions in the sup-norm.

**Note** However, this does not align with what we see in practice, where small \( \sigma \) yields the best performance, and increasing \( \sigma \) leads to worse performance. This is because Achiam et al. [1] makes the assumption that the replay buffer assigns positive probability to every possible transition, which is impossible in our continuous MDPs. We also do stochastic optimization with minibatches, which further deviates from the theory. Finally, guarantee of a contraction does not imply sample efficiency. Indeed, an algorithm that contracts slowly at every step can be much worse than an algorithm that greatly improves in expectation.
B On-policy Results

Figure 11: On-Policy Evaluation: We train PPO on 4 OpenAI Gym environments with our LFF architecture and a vanilla MLP, both of which have roughly the same number of parameters. LFF does not produce a consistent gain in sample efficiency here, which is consistent with our hypothesis that LFF helps only for off-policy RL.

B.1 On-policy LFF Setup

We evaluate proximal policy optimization (PPO, Schulman et al. [36]) on 4 environments from OpenAI gym [7]. These environments range from easy, e.g. HalfCheetah-v2, to difficult, e.g. Humanoid-v2. Just as we did in the off-policy setup, we modify only the architecture, keeping the hyperparameters fixed. We compare MLPs with 3 hidden layers to LFF with our Fourier feature input layer followed by 2 hidden layers. We use $d_{\text{fourier}} = 1024$ and $\sigma = 0.001$ for all environments and also test what happens if we use LFF for only the policy or only the critic.

B.2 LFF architecture for on-policy RL

In Figure 11, we show the results of using LFF for PPO. Unlike LFF on SAC, LFF did not yield consistent gains for PPO. The best setting was to use LFF for only the critic, which does as well as MLPs on 3 environments (HalfCheetah, Hopper, and Humanoid) and better on Walker2d, but this is only a modest improvement. This is not surprising and is likely because policy gradient methods have different optimization challenges than those based on Q-learning. For one, the accuracy of the value function baseline is less important for policy gradient methods. The on-policy cumulative return provides a lot of reward signal, and the value function baseline mainly reduces variance in the gradient update. In addition, generalized advantage estimation [35] further reduces variance. Thus, noise in the bootstrapping process is not as serious of a problem as in off-policy learning.
C Performance Under Added Noise

Figure 12: State-based with Added Noise: We add zero-mean Gaussian noise to the targets. As the standard deviation of the added noise increases, LFF maintains its performance better than MLPs.
D Noise Amplification vs Implicit Underparameterization in Gridworld MDP

![Figure 13: Measuring noise amplification vs implicit underparameterization on GRID16SMOOTHOBS.](image)

We want to examine whether noise amplification is indeed a problem that off-policy deep RL methods suffer from. Furthermore, if it is happening, Kumar et al. [23] hypothesize that it is being caused by underfitting (“implicit underparameterization”), which would contradict our claim that LFF improves learning by regularizing the training dynamics. We test this in Figure 13 by performing Fitted Q-iteration (FQI) on the GRID16SMOOTHOBS environment from Kumar et al. [23]. GRID16SMOOTHOBS is a discrete environment with 256 states and 5 actions, so we can use Q-iteration to calculate the optimal $Q^*$ and compare that with our learned $Q_\theta$. $T$ denotes the number of gradient steps per FQI iteration; increasing the number of gradient steps is empirically more likely to cause divergence for MLP Q-functions. Note that we use a log scale for the y-axis in (a,b,e).

**Noise amplification** (a) shows that MLP-based Q-functions steadily blow up to orders of magnitude above the true $Q^*$, whose average value is around 15 in this environment. In contrast, LFF-based Q-functions either converge stably to the correct magnitude, or resist increasing as much. (b) shows that the MLP target values are in a positive feedback loop with the Q-values. (c) shows that divergence coincides with a drop in the returns. Together, these results indicate that there can be a harmful feedback loop in the bootstrapping process, and that methods like LFF, which reduce fitting to noise, can help stabilize training.

**Implicit Underparameterization** We run Fitted Q-iteration and calculate the effective rank of the Q-network, which we parameterize using either a MLP or LFF network. We follow the procedure from Kumar et al. [23]: sample 2048 states from the replay buffer and calculate the singular values $\sigma_j$ of the aggregated feature matrix $\Phi$. The effective rank is then defined as $\text{srank}_\delta(\Phi) = \min \left\{ k : \frac{1}{\sum_{j=1}^k \sigma_j^2(\Phi)} \geq 1 - \delta \right\}$. (d) shows that the MLP’s effective rank does not actually drop over training. Furthermore, LFF is able to avoid diverging $Q$-values, even though it has significantly lower srank than its MLP counterpart. Thus, noise amplification for MLPs in this setting is likely not related to any underfitting measured by the effective rank. (e) shows that the...
E Sensitivity Analyses

Figure 14: Sensitivity to \( \sigma \). \( \sigma = 0.001 \) is a good default across all of these state-based environments. Results are averaged over 5 seeds, using a Fourier dimension of 1024, and the shaded region denotes 1 standard error.

Figure 15: Sensitivity to the Fourier dimension. A Fourier dimension of 1024 is a good default across all of these state-based environments. Results are averaged over 5 seeds, using \( \sigma = 0.001 \), and the shaded region denotes 1 standard error.
F High Frequency Learning with Learned Fourier Features

Figure 16: Potential Underfitting in RL. Left: gridworld structure. Red squares are lava, green are goals, gray are walls, and white squares are empty. Middle left: ground truth value function $V^\star(s)$. Middle right: we fit $Q^\star(s, a)$ with an MLP, then display $\max_a Q_\theta(s, a)$. The result is blurry and cannot properly distinguish between critical high and low value states. Right: we fit $Q^\star(s, a)$ with our proposed learned Fourier feature architecture, then display $\max_a Q_\rho(s, a)$. It is able to exactly reproduce the ground truth, even with the same number of parameters and gradient steps.

While the main paper focused on using LFF with small $\sigma$ to bias networks towards mainly learning low frequency functions, we can still use larger $\sigma$ to encourage faster high-frequency learning. Prior theoretical work [4, 9, 33, 49] found that ReLU MLPs suffer from spectral bias – they can take an impractically long time to fit the highest frequencies. In the reinforcement learning setting, this can cause underfitting when fitting high frequencies is desirable.

F.1 Gridworld

We demonstrate that underfitting is indeed happening in RL. We create a toy $64 \times 64$ gridworld task [11] in Figure 16, where each square can be one of five types: start state, goal state, empty, wall, or lava. The agent starts at the start state and receives +1 reward for every timestep at a goal state, -1 penalty for every timestep at a lava square, and 0 reward otherwise. It cannot enter cells with a wall. The agent has five actions available: up, down, left, right, or no-op. Whenever the agent takes a step, there is a 20% chance of a random action being taken instead, so it is important for the agent to stay far from lava, lest it accidentally fall in. We use a discount of $\gamma = 0.9$. To create the environment, we randomly initialize it with 25% lava squares and 10% wall squares. We learn the optimal $Q^\star$ using Q-iteration, then attempt to fit various neural network architectures with parameters $\theta$ to the ground truth $Q^\star$ values through supervised learning:

$$\theta^* = \arg \min_\theta \sum_{s \in S, a \in A} (Q_\theta(s, a) - Q^\star(s, a))^2 \quad (47)$$

We then try to fit $Q^\star$ using a standard MLP with 3 layers and 256 hidden units, and using our proposed architecture with an equivalent number of parameters and $\sigma = 3$. Due to the challenge of visualizing $Q(s, a)$ with 5 actions, we instead show $V(s) = \max_a Q(s, a)$ in Figure 16.

Surprisingly, MLPs have extreme difficulty fitting $Q^\star(s, a)$, even when doing supervised learning on this low-dimensional toy example. Even without the challenges of nonstationarity, bootstrapping, and exploration, the deep neural network has trouble learning the optimal Q-function. The learned Q-function blurs together nearby states, making it impossible for the agent to successfully navigate narrow corridors of lava. Prior work has described this problem as state aliasing [27], where an agent conflates separate states in its representation space. We anticipate that this problem is worse in higher dimensional and continuous MDPs and is pervasive throughout reinforcement learning.

In contrast, our LFF embedding with $\sigma = 3$ is able to perfectly learn the ground-truth Q-function. This indicates that LFF with large $\sigma$ can help our Q-networks and policies fit key high-frequency details in certain RL settings. We believe that there are two promising applications for high-frequency learning with LFF: model-based RL and tabular problems. Model-based RL requires modeling the dynamics, which can have sharp changes, such as at contact points. Modeling transition dynamics is also supervised, so there are no bootstrap noise problems exacerbated by accelerating the rate at which high-frequencies are learned. Tabular problems are also suited for high-frequency learning, as they often have sharp changes in dynamics or rewards (e.g. gridworld squares with walls, cliffs, or
lava). Capturing high-frequencies with LFF could improve both the sample-efficiency and asymptotic performance in this setting.

G Fourier Basis Variance After Training

Figure 17: Change in the standard deviation of Fourier basis entries $B_{ij}$. The curves on the left show how the standard deviation changes from $\sigma$ at initialization as state-based SAC training progresses. The confidence interval indicates the standard deviation of the standard deviation, measured across 5 seeds. The histograms in the middle and left show the distribution of entries within $B$ for the policy and Q-networks at the end of training.

In Section 4, we used the Neural Tangent Kernel (NTK) perspective to show that the initialization variance of the Fourier basis $B$ controls the per-frequency learning rate in infinite-width neural networks. However, the NTK’s infinite-width assumption implies that the entries of the Fourier basis do not change over the course of training. Since we train $B$ with finite width, we examine how its variance evolves over training, which affects its per-frequency learning rate at each point in time. Figure 17 and 18 show how the standard deviation of the Fourier basis change for the policy and Q-network. The standard deviation generally increases from $\sigma = 0.001$ or $\sigma = 0.003$ to about 0.1, which should still be more biased towards low frequencies than vanilla MLPs are (see Figure 4). Furthermore, the increase in standard deviation could actually be desirable. Having more data at the end of training could reduce the impact of bootstrap noise, so there may be less need for smoothing with small $\sigma$. Larger $\sigma$ could bias the network towards fitting medium-frequency signals that are important for achieving full asymptotic efficiency. This could explain the image-based results in Figure 19, where the standard deviation rises to about 0.5.
Figure 18: Continuation of Figure 17, which shows how the standard deviation of the Fourier basis changes over training.
Figure 19: **Change in the standard deviation of Fourier basis entries** $B_{ij}$ **for image-based experiments.** For each of the four environments, we show how the standard deviation of the shared encoder’s Fourier basis changes over training. The histogram shows the distribution of the entries of the shared encoder’s $B$ at the end of training.

### Hyperparameters

We list hyperparameters for state-based and image-based SAC experiments in Table 1 and Table 2, respectively. We also show hyperparameters for PPO experiments from Appendix B in Table 3.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Soft Actor Critic [14]</td>
</tr>
<tr>
<td>Starting codebase</td>
<td>Yarats and Kostrikov [51]</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam [19]</td>
</tr>
<tr>
<td>Adam ($\beta_1$, $\beta_2$)</td>
<td>(0.9, 0.999)</td>
</tr>
<tr>
<td>Discount</td>
<td>0.99</td>
</tr>
<tr>
<td>Batch size</td>
<td>1024</td>
</tr>
<tr>
<td>Target smoothing coefficient ($\tau$)</td>
<td>0.005</td>
</tr>
<tr>
<td>Reward scale</td>
<td>Auto-tuned [15]</td>
</tr>
<tr>
<td>Actor learning rate</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Critic learning rate</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Reward scale learning rate</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Number of exploratory warmup steps</td>
<td>5000</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>3 for MLP, 2 for LFF</td>
</tr>
<tr>
<td>Hidden size</td>
<td>1024</td>
</tr>
<tr>
<td>Hidden nonlinearity</td>
<td>ReLU</td>
</tr>
<tr>
<td>Fourier dimension</td>
<td>64 for Cheetah, 1024 otherwise</td>
</tr>
<tr>
<td>Standard deviation $\sigma$ of Fourier basis initialization</td>
<td>0.003 for Cheetah, 0.001 otherwise</td>
</tr>
</tbody>
</table>

Table 1: Hyperparameters used for the state-based SAC experiments.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting codebase</td>
<td>Laskin et al. [24]</td>
</tr>
<tr>
<td>Observation rendering</td>
<td>(100, 100)</td>
</tr>
<tr>
<td>Replay buffer size</td>
<td>100000</td>
</tr>
<tr>
<td>Initial steps</td>
<td>1000</td>
</tr>
<tr>
<td>Stacked frames</td>
<td>3</td>
</tr>
<tr>
<td>Action repeat</td>
<td>4</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam [19]</td>
</tr>
<tr>
<td>Adam ($\beta_1, \beta_2$)</td>
<td>(0.5, 0.999) for entropy, (0.9, 0.999) otherwise</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Batch size</td>
<td>512</td>
</tr>
<tr>
<td>Encoder smoothing coefficient ($\tau$)</td>
<td>0.05</td>
</tr>
<tr>
<td>Q-network smoothing coefficient ($\tau$)</td>
<td>0.01</td>
</tr>
<tr>
<td>Critic target update freq</td>
<td>2</td>
</tr>
<tr>
<td>Convolutional layers (excluding LFF embedding)</td>
<td>4 for LFF, 5 otherwise</td>
</tr>
<tr>
<td>Discount</td>
<td>0.99</td>
</tr>
<tr>
<td>Fourier dimension</td>
<td>128 for Hopper, 64 otherwise</td>
</tr>
<tr>
<td>Initial standard deviation $\sigma$ of Fourier basis</td>
<td>0.1 for Hopper, Cheetah; 0.01 for Finger, Quadruped</td>
</tr>
</tbody>
</table>

Table 2: Hyperparameters used for the image-based SAC experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Proximal Policy Optimization [36]</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$3 \times 10^{-4}$</td>
</tr>
<tr>
<td>Learning rate decay</td>
<td>linear</td>
</tr>
<tr>
<td>Entropy coefficient</td>
<td>0</td>
</tr>
<tr>
<td>Value loss coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>Clip parameter</td>
<td>0.2</td>
</tr>
<tr>
<td>Environment steps per optimization loop</td>
<td>2048</td>
</tr>
<tr>
<td>PPO epochs per optimization loop</td>
<td>10</td>
</tr>
<tr>
<td>Batch size</td>
<td>64</td>
</tr>
<tr>
<td>GAE $\lambda$</td>
<td>0.95</td>
</tr>
<tr>
<td>Discount</td>
<td>0.99</td>
</tr>
<tr>
<td>Total timesteps</td>
<td>$10^6$</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>2</td>
</tr>
<tr>
<td>Hidden size</td>
<td>256</td>
</tr>
<tr>
<td>Hidden nonlinearity</td>
<td>tanh</td>
</tr>
<tr>
<td>Fourier dimension</td>
<td>64</td>
</tr>
<tr>
<td>Variance of Fourier basis initialization $\tau$</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 3: Hyperparameters used for the PPO experiments.