Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains

Matthew Tancik[1*] **Pratul P. Srinivasan**[1,2*] **Ben Mildenhall**[1*] **Sara Fridovich-Keil**[1]

Nithin Raghavan[1] **Utkarsh Singhal**[1] **Ravi Ramamoorthi**[3] **Jonathan T. Barron**[2] **Ren Ng**[1]

[1]University of California, Berkeley [2]Google Research [3]University of California, San Diego

Abstract

We show that passing input points through a simple Fourier feature mapping enables a multilayer perceptron (MLP) to learn high-frequency functions in low-dimensional problem domains. These results shed light on recent advances in computer vision and graphics that achieve state-of-the-art results by using MLPs to represent complex 3D objects and scenes. Using tools from the neural tangent kernel (NTK) literature, we show that a standard MLP fails to learn high frequencies both in theory and in practice. To overcome this spectral bias, we use a Fourier feature mapping to transform the effective NTK into a stationary kernel with a tunable bandwidth. We suggest an approach for selecting problem-specific Fourier features that greatly improves the performance of MLPs for low-dimensional regression tasks relevant to the computer vision and graphics communities.

1 Introduction

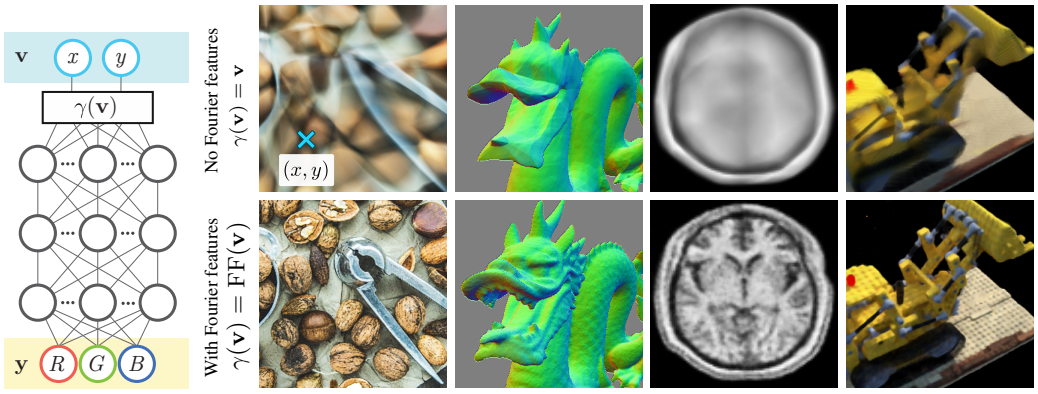
A recent line of research in computer vision and graphics replaces traditional discrete representations of objects, scene geometry, and appearance (*e.g.* meshes and voxel grids) with continuous functions parameterized by deep fully-connected networks (also called multilayer perceptrons or MLPs). These MLPs, which we will call "coordinate-based" MLPs, take low-dimensional coordinates as inputs (typically points in $\mathbb{R}^3$) and are trained to output a representation of shape, density, and/or color at each input location (see Figure 1). This strategy is compelling since coordinate-based MLPs are amenable to gradient-based optimization and machine learning, and can be orders of magnitude more compact than grid-sampled representations. Coordinate-based MLPs have been used to represent images [28, 38] (referred to as "compositional pattern producing networks"), volume density [27], occupancy [24], and signed distance [32], and have achieved state-of-the-art results across a variety of tasks such as shape representation [9, 10, 12, 13, 17, 26, 32], texture synthesis [15, 31], shape inference from images [22, 23], and novel view synthesis [27, 29, 35, 37].

We leverage recent progress in modeling the behavior of deep networks using kernel regression with a neural tangent kernel (NTK) [16] to theoretically and experimentally show that standard MLPs are poorly suited for these low-dimensional coordinate-based vision and graphics tasks. In particular, MLPs have difficulty learning high frequency functions, a phenomenon referred to in the literature as "spectral bias" [3, 33]. NTK theory suggests that this is because standard coordinate-based MLPs correspond to kernels with a rapid frequency falloff, which effectively prevents them from being able to represent the high-frequency content present in natural images and scenes.

A few recent works [27, 44] have experimentally found that a heuristic sinusoidal mapping of input coordinates (called a "positional encoding") allows MLPs to represent higher frequency content.

[*] Authors contributed equally to this work.

Preprint. Under review.



(a) Coordinate-based MLP (b) Image regression $(x,y) \rightarrow$ RGB (c) 3D shape regression $(x,y,z) \rightarrow$ occupancy (d) MRI reconstruction $(x,y,z) \rightarrow$ density (e) Inverse rendering $(x,y,z) \rightarrow$ RGB, density

Figure 1: Fourier features improve the results of coordinate-based MLPs for a variety of high-frequency low-dimensional regression tasks, both with direct (b, c) and indirect (d, e) supervision. We visualize an example MLP (a) for an image regression task (b), where the input to the network is a pixel coordinate and the output is that pixel's color. Passing coordinates directly into the network (top) produces blurry images, whereas preprocessing the input with a Fourier feature mapping (bottom) enables the MLP to represent higher frequency details.

We observe that this is a special case of Fourier features [34]: mapping input coordinates $\mathbf{v}$ to $\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v})\right]^{\mathrm{T}}$ before passing them into an MLP. We show that this mapping transforms the NTK into a stationary (shift-invariant) kernel and enables tuning the NTK's spectrum by modifying the frequency vectors $\mathbf{b}_j$, thereby controlling the range of frequencies that can be learned by the corresponding MLP. We show that the simple strategy of setting $a_j = 1$ and randomly sampling $\mathbf{b}_j$ from an isotropic distribution achieves good performance, and that the scale (standard deviation) of this distribution matters much more than its specific shape. We train MLPs with this Fourier feature input mapping across a range of tasks relevant to the computer vision and graphics communities. As highlighted in Figure 1, our proposed mapping dramatically improves the performance of coordinate-based MLPs. In summary, we make the following contributions:

- We leverage NTK theory and simple experiments to show that a Fourier feature mapping can be used to overcome the spectral bias of coordinate-based MLPs towards low frequencies by allowing them to learn much higher frequencies (Section 4).
- We demonstrate that a random Fourier feature mapping with an appropriately chosen scale can dramatically improve the performance of coordinate-based MLPs across many low-dimensional tasks in computer vision and graphics (Section 5).

2 Related Work

Our work is motivated by the widespread use of coordinate-based MLPs to represent a variety of visual signals, including images [38] and 3D scenes [24, 27, 32]. In particular, our analysis is intended to clarify experimental results demonstrating that an input mapping of coordinates (which they called a "positional encoding") using sinusoids with logarithmically-spaced axis-aligned frequencies improves the performance of coordinate-based MLPs on the tasks of novel view synthesis from 2D images [27] and protein structure modeling from cryo-electron microscopy [44]. We analyze this technique to show that it corresponds to a modification of the MLP's NTK, and we show that other non-axis-aligned frequency distributions can outperform this positional encoding.

Prior works in natural language processing and time series analysis [18, 39, 42] have used a similar positional encoding to represent time or 1D position. In particular, Xu *et al.* [42] use random Fourier features (RFF) [34] to approximate stationary kernels with a sinusoidal input mapping and propose techniques to tune the mapping parameters. Our work extends this by directly explaining such

mappings as a modification of the resulting network's NTK. Additionally, we address the embedding of multidimensional coordinates, which is necessary for vision and graphics tasks.

To analyze the effects of applying a Fourier feature mapping to input coordinates before passing them through an MLP, we rely on recent theoretical work that models neural networks in the limits of infinite width and infinitesimal learning rate as kernel regression using the NTK [2, 5, 11, 16, 20]. In particular, we use the analyses from Lee *et al.* [20] and Arora *et al.* [2], which show that the outputs of a network throughout gradient descent remain close to those of a linear dynamical system whose convergence rate is governed by the eigenvalues of the NTK matrix [2, 3, 5, 20, 43]. Analysis of the NTK's eigendecomposition shows that its eigenvalue spectrum decays rapidly as a function of frequency, which explains the widely-observed "spectral bias" of deep networks towards learning low-frequency functions [3, 4, 33].

We leverage this analysis to consider the implications of adding a Fourier feature mapping before the network, and we show that this mapping has a significant effect on the NTK's eigenvalue spectrum and on the corresponding network's convergence properties in practice.

3 Background and Notation

To lay the foundation for our theoretical analysis, we first review classic kernel regression and its connection to recent results that analyze the training dynamics and generalization behavior of deep fully-connected networks. In later sections, we use these tools to analyze the effects of training coordinate-based MLPs with Fourier feature mappings.

Kernel regression. Kernel regression is a classic nonlinear regression algorithm [40]. Given a training dataset $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i$ are input points and $y_i = f(\mathbf{x}_i)$ are the corresponding scalar output labels, kernel regression constructs an estimate $\hat{f}$ of the underlying function at any point $\mathbf{x}$ as:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \left(\mathbf{K}^{-1}\mathbf{y}\right)_i k(\mathbf{x}_i, \mathbf{x}) \,, \tag{1}$$

where $\mathbf{K}$ is an $n \times n$ kernel (Gram) matrix with entries $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and k is a symmetric positive semidefinite (PSD) kernel function which represents the "similarity" between two input vectors. Intuitively, the kernel regression estimate at any point $\mathbf{x}$ can be thought of as a weighted sum of training labels y_i using the similarity between the corresponding $\mathbf{x}_i$ and $\mathbf{x}$.

Approximating deep networks with kernel regression. Let f be a fully-connected deep network with weights θ initialized from a Gaussian distribution $\mathcal{N}$. Theory proposed by Jacot *et al.* [16] and extended by others [2, 3, 20] shows that when the width of the layers in f tends to infinity and the learning rate for SGD tends to zero, the function $f(\mathbf{x}; \theta)$ converges over the course of training to the kernel regression solution using the *neural tangent kernel* (NTK), defined as:

$$k_{\text{NTK}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}_{\theta \sim \mathcal{N}} \left\langle \frac{\partial f(\mathbf{x}_i; \theta)}{\partial \theta}, \frac{\partial f(\mathbf{x}_j; \theta)}{\partial \theta} \right\rangle . \tag{2}$$

When the inputs are restricted to a hypersphere, the NTK for an MLP can be written as a dot product kernel (a kernel in the form $h_{\text{NTK}}(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)$ for a scalar function $h_{\text{NTK}} : \mathbb{R} \to \mathbb{R}$).

Prior work [2, 3, 16, 20] shows that an NTK linear system model can be used to approximate the dynamics of a deep network during training. We consider a network trained with an L2 loss and a learning rate η, where the network's weights are initialized such that the output of the network at initialization is close to zero. Under asymptotic conditions stated in Lee *et al.* [20], the network's output for any data $\mathbf{X}_{\text{test}}$ after t training iterations can be approximated as:

$$\hat{\mathbf{y}}^{(t)} \approx \mathbf{K}_{\text{test}}\mathbf{K}^{-1}\left(\mathbf{I} - e^{-\eta\mathbf{K}t}\right)\mathbf{y} \,, \tag{3}$$

where $\hat{\mathbf{y}}^{(t)} = f(\mathbf{X}_{\text{test}}; \theta)$ are the network's predictions on input points $\mathbf{X}_{\text{test}}$ at training iteration t, $\mathbf{K}$ is the NTK matrix between all pairs of training points in $\mathbf{X}$, and $\mathbf{K}_{\text{test}}$ is the NTK matrix between all points in $\mathbf{X}_{\text{test}}$ and all points in the training dataset $\mathbf{X}$.

Spectral bias when training neural networks. Let us consider the training error $\hat{\mathbf{y}}_{\text{train}}^{(t)} - \mathbf{y}$, where $\hat{\mathbf{y}}_{\text{train}}^{(t)}$ are the network's predictions on the training dataset at iteration t. Since the NTK matrix $\mathbf{K}$

must be PSD, we can take its eigendecomposition $\mathbf{K} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}}$, where $\mathbf{Q}$ is orthogonal and $\mathbf{\Lambda}$ is a diagonal matrix whose entries are the eigenvalues $\lambda_i \geq 0$ of $\mathbf{K}$. Then, since $e^{-\eta\mathbf{K}t} = \mathbf{Q}e^{-\eta\mathbf{\Lambda}t}\mathbf{Q}^{\mathrm{T}}$:

$$\mathbf{Q}^{\mathrm{T}}(\hat{\mathbf{y}}_{\text{train}}^{(t)} - \mathbf{y}) \approx \mathbf{Q}^{\mathrm{T}}\left(\left(\mathbf{I} - e^{-\eta\mathbf{K}t}\right)\mathbf{y} - \mathbf{y}\right) = -e^{-\eta\mathbf{\Lambda}t}\mathbf{Q}^{\mathrm{T}}\mathbf{y}\,. \tag{4}$$

This means that if we consider training convergence in the eigenbasis of the NTK, the i^{th} component of the absolute error $|\mathbf{Q}^{\mathrm{T}}(\hat{\mathbf{y}}_{\text{train}}^{(t)} - \mathbf{y})|_i$ will decay approximately exponentially at the rate $\eta\lambda_i$. In other words, components of the target function that correspond to kernel eigenvectors with larger eigenvalues will be learned faster. For a conventional MLP, the eigenvalues of the NTK decay rapidly [4, 5, 14]. This results in extremely slow convergence to the high frequency components of the target function, to the point where standard MLPs are effectively unable to learn these components, as visualized in Figure 1. Next, we describe a technique to address this slow convergence by using a Fourier feature mapping of input coordinates before passing them to the MLP.

4 Fourier Features for a Tunable Stationary Neural Tangent Kernel

Machine learning analysis typically addresses the case in which inputs are high dimensional points (*e.g.* the pixels of an image reshaped into a vector) and training examples are sparsely distributed. In contrast, in this work we consider *low-dimensional regression* tasks, wherein inputs are assumed to be dense coordinates in a subset of $\mathbb{R}^d$ for small values of d (*e.g.* pixel coordinates). This setting has two significant implications when viewing deep networks through the lens of kernel regression:

1. We would like the composed NTK to be shift-invariant over the input domain, since the training points are distributed with uniform density. In problems where the inputs are normalized to the surface of a hypersphere (common in machine learning), a dot product kernel (such as the regular NTK) corresponds to spherical convolution. However, inputs in our setting are dense in Euclidean space. A Fourier feature mapping of input coordinates makes the composed NTK stationary (shift-invariant), acting as a convolution kernel over the input domain (see Appendix C for additional discussion on stationary kernels).
2. We would like to control the bandwidth of the NTK to improve training speed and generalization. As we see from Eqn. 4, a "wider" kernel with a slower spectral falloff achieves faster training convergence for high frequency components. However, we know from signal processing that reconstructing a signal using a kernel whose spectrum is *too* wide causes high frequency aliasing artifacts. We show in Section 5 that a Fourier feature input mapping can be tuned to lie between these "underfitting' and "overfitting" extremes, enabling both fast convergence and low test error.

Fourier features and the composed neural tangent kernel. Fourier feature mappings have been used in many applications since their introduction in the seminal work of Rahimi and Recht [34], which used random Fourier features to approximate an arbitrary stationary kernel function by applying Bochner's theorem. Extending this technique, we use a Fourier feature mapping γ to featurize input coordinates before passing them through a coordinate-based MLP, and investigate the theoretical and practical effect this has on convergence speed and generalization. The function γ maps input points $\mathbf{v} \in [0, 1)^d$ to the surface of a higher dimensional hypersphere with a set of sinusoids:

$$\gamma(\mathbf{v}) = \left[a_1\cos(2\pi\mathbf{b}_1^{\mathrm{T}}\mathbf{v}), a_1\sin(2\pi\mathbf{b}_1^{\mathrm{T}}\mathbf{v}), \ldots, a_m\cos(2\pi\mathbf{b}_m^{\mathrm{T}}\mathbf{v}), a_m\sin(2\pi\mathbf{b}_m^{\mathrm{T}}\mathbf{v})\right]^{\mathrm{T}}\,. \tag{5}$$

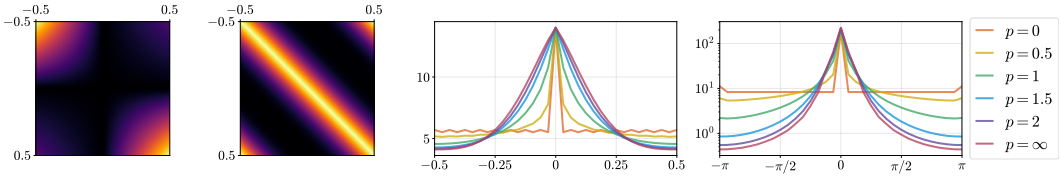
Because $\cos(\alpha - \beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$, the kernel function induced by this mapping is:

$$k_\gamma(\mathbf{v}_1, \mathbf{v}_2) = \gamma(\mathbf{v}_1)^{\mathrm{T}}\gamma(\mathbf{v}_2) = \sum_{j=1}^{m} a_j^2\cos\left(2\pi\mathbf{b}_j^{\mathrm{T}}\left(\mathbf{v}_1 - \mathbf{v}_2\right)\right) = h_\gamma(\mathbf{v}_1 - \mathbf{v}_2)\,, \tag{6}$$

$$\text{where } h_\gamma(\mathbf{v}_\Delta) \triangleq \sum_{j=1}^{m} a_j^2\cos(2\pi\mathbf{b}_j^{\mathrm{T}}\mathbf{v}_\Delta)\,. \tag{7}$$

Note that this kernel is stationary (a function of only the difference between points). We can think of the mapping as a Fourier approximation of a kernel function: $\mathbf{b}_j$ are the Fourier basis frequencies used to approximate the kernel, and a_j^2 are the corresponding Fourier series coefficients.

After computing the Fourier features for our input points, we pass them through an MLP to get $f(\gamma(\mathbf{v}); \theta)$. As discussed previously, the result of training a network can be approximated by kernel



(a) No mapping NTK (b) Basic mapping NTK (c) NTK spatial (d) NTK Fourier spectrum

Figure 2: Adding a Fourier feature mapping can improve the poor conditioning of a coordinate-based MLP's neural tangent kernel (NTK). (a) We visualize the NTK function $k_{\text{NTK}}(x_i, x_j)$ (Eqn. 2) for a 4-layer ReLU MLP with one scalar input. This kernel is not shift-invariant and does not have a strong diagonal, making it poorly suited for kernel regression in low-dimensional problems. (b) A basic input mapping $\gamma(v) = [\cos 2\pi v, \sin 2\pi v]^{\mathrm{T}}$ makes the composed NTK $k_{\text{NTK}}(\gamma(v_i), \gamma(v_j))$ shift-invariant (stationary). (c) A Fourier feature input mapping (Eqn. 5) can be used to tune the composed kernel's width, where we set $a_j = 1/j^p$ and $b_j = j$ for $j = 1, \ldots, n/2$. (d) Higher frequency mappings (lower p) result in composed kernels with wider spectra, which enables faster convergence for high-frequency components (see Figure 3).

regression using the kernel $h_{\text{NTK}}(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)$. In our case, $\mathbf{x}_i = \gamma(\mathbf{v}_i)$ so the composed kernel becomes:

$$h_{\text{NTK}}(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j) = h_{\text{NTK}}\left(\gamma\left(\mathbf{v}_i\right)^{\mathrm{T}}\gamma\left(\mathbf{v}_j\right)\right) = h_{\text{NTK}}\left(h_\gamma\left(\mathbf{v}_i - \mathbf{v}_j\right)\right). \tag{8}$$

Thus, training a network on these embedded input points corresponds to kernel regression with the *stationary* composed NTK function $h_{\text{NTK}} \circ h_\gamma$. The MLP function approximates a convolution of the composed NTK with a weighted Dirac delta at each input training point $\mathbf{v}_i$:

$$\hat{f} = (h_{\text{NTK}} \circ h_\gamma) * \sum_{i=1}^{n} w_i \delta_{\mathbf{v}_i} \tag{9}$$

where $\mathbf{w} = \mathbf{K}^{-1}\mathbf{y}$ (from Eqn. 1). This allows us to draw analogies to signal processing, where the composed NTK acts similarly to a reconstruction filter. In the next section, we show that the frequency decay of the composed NTK determines the behavior of the reconstructed signal.

5 Manipulating the Fourier Feature Mapping

Preprocessing the inputs to a coordinate-based MLP with a Fourier feature mapping creates a composed NTK that is not only stationary but also *tunable*. By manipulating the settings of the a_j and $\mathbf{b}_j$ parameters in Eqn. 5, it is possible to dramatically change both the rate of convergence and the generalization behavior of the resulting network. In this section, we investigate the effects of the Fourier feature mapping in the setting of 1D function regression.

We train MLPs to learn signals f defined on the interval $[0, 1)$. We sample cn linearly spaced points on the interval, using every c^{th} point as the training set and the remaining points as the test set. Since our composed kernel function is stationary, evaluating it at linearly spaced points on a periodic domain makes the resulting kernel matrix circulant: it represents a convolution and is diagonalizable by the Fourier transform. Thus, we can compute the eigenvalues of the composed NTK matrix by simply taking the Fourier transform of a single row. All experiments are implemented in JAX [8] and the NTK functions are calculated automatically using the Neural Tangents library [30].

Visualizing the composed NTK. We first visualize how modifying the Fourier feature mapping changes the composed NTK. We set $b_j = j$ (full Fourier basis in 1D) and $a_j = 1/j^p$ for $j = 1, \ldots, n/2$. We use $p = \infty$ to denote the mapping $\gamma(v) = [\cos 2\pi v, \sin 2\pi v]^{\mathrm{T}}$ that simply wraps $[0, 1)$ around the unit circle (this is referred to as the "basic" mapping in later experiments). Figure 2 demonstrates the effect of varying p on the composed NTK. By construction, lower p values result in a slower falloff in the frequency domain and a correspondingly narrower kernel in the spatial domain.

Effects of Fourier features on network convergence. We generate ground truth 1D functions by sampling cn values from a family with parameter α as follows: we sample a standard i.i.d. Gaussian vector of length cn, scale its i^{th} entry by $1/i^\alpha$, then return the real component of its inverse Fourier transform. We will refer to this as a "$1/f^\alpha$ noise" signal.

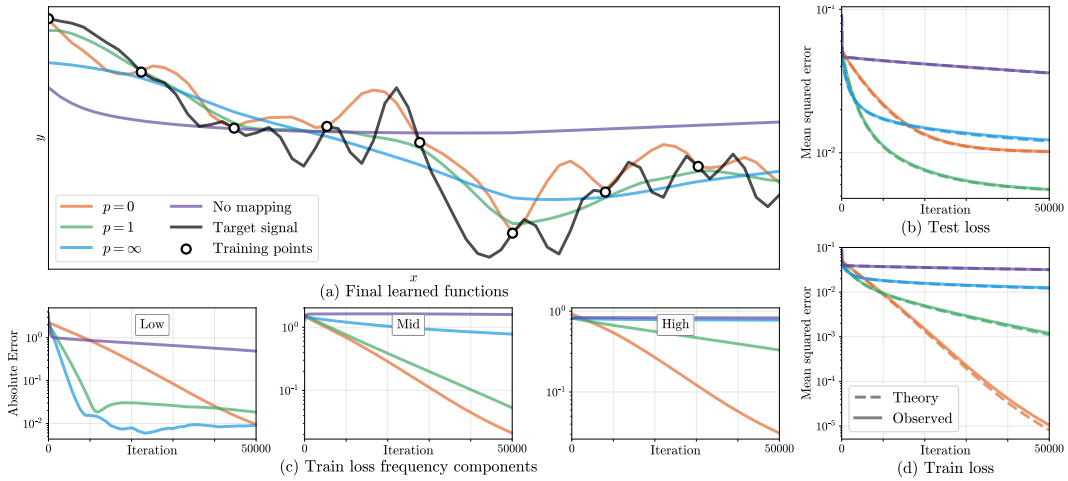


Figure 3: Combining a network with a Fourier feature mapping has dramatic effects on convergence and generalization. Here we train a network on 32 sampled points from a 1D function (a) using mappings shown in Fig. 2. A mapping with a smaller p value yields a composed NTK with more power in higher frequencies, enabling the corresponding network to learn a higher frequency function. The theoretical and experimental training loss improves monotonically with higher frequency kernels (d), but the test-set loss is lowest at $p = 1$ and falls as the network starts to overfit (b). As predicted by Eqn. 4, we see roughly log-linear convergence of the training loss frequency components (c). Higher frequency kernels result in faster convergence for high frequency loss components, thereby overcoming the "spectral bias" observed when training networks with no input mapping.

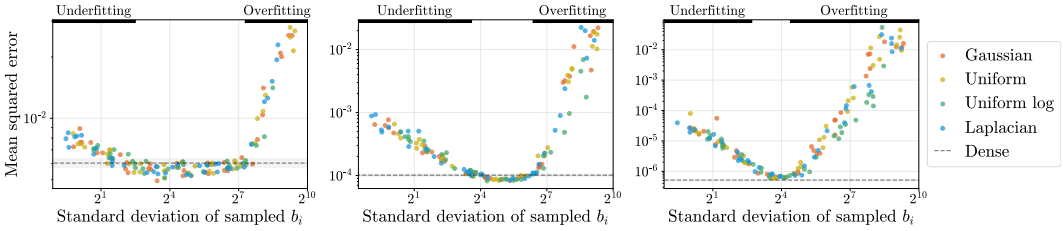
In Figure 3, we train MLPs (4 layers, 1024 channels, ReLU activations) to fit a bandlimited $1/f^1$ noise signal ($c = 8, n = 32$) using Fourier feature mappings with different p values. Figures 3b and 3d show that the NTK linear dynamics model accurately predict the effects of modifying the Fourier feature mapping parameters. Separating different frequency components of the training error in Figure 3c reveals that networks with narrower NTK spectra converge faster for low frequency components but essentially never converge for high frequency components, whereas networks with wider NTK spectra successfully converge across all components. The Fourier feature mapping $p = 1$ has adequate power across frequencies present in the target signal (so the network converges rapidly during training) but limited power in higher frequencies (preventing overfitting or aliasing).

Tuning Fourier features in practice. Eqn. 3 allows us to estimate a trained network's theoretical loss on a validation set using the composed kernel. For small 1D problems, we can minimize this loss with gradient-based optimization to choose mapping parameters a_j (given a dense sampling of b_j). In this carefully controlled setting (1D signals, small training dataset, gradient descent with small learning rate, very wide networks), we find that this optimized mapping also achieves the best performance when training networks. Please refer to Appendix A.1 for details and experiments.

In real-world problems, especially in multiple dimensions, it is not feasible to use a feature mapping that densely samples Fourier basis functions; the number of Fourier basis functions scales with the number of training data points, which grows exponentially with dimension. Instead, we sample a set of random Fourier features [34] from a parametric distribution. We find that the exact sampling distribution family is much less important than the distribution's scale (standard deviation).

Figure 4 demonstrates this point using hyperparameter sweeps for a variety of sampling distributions. In each subfigure, we draw 1D target signals ($c = 2, n = 1024$) from a fixed $1/f^\alpha$ distribution and train networks to learn them. We use random Fourier feature mappings (of length 16) sampled from different distribution families (Gaussian, uniform, uniform in log space, and Laplacian) and sweep over each distribution's scale. Perhaps surprisingly, the standard deviation of the sampled frequencies alone is enough to predict test set performance, regardless of the underlying distribution's shape. We show that this holds for higher-dimensional tasks in Appendix A.4. We also observe that passing this sparse sampling of Fourier features through an MLP matches the performance of using a dense set of Fourier features with the same MLP, suggesting a strategy for scaling to higher dimensions. We

scales with $(n+1)^{\text{dimension}}$



(a) Data sampled from $1/f^{0.5}$ (b) Data sampled from $1/f^{1.0}$ (c) Data sampled from $1/f^{1.5}$

Figure 4: We find that a sparse random sampling of Fourier features can perform as well as a dense set of features and that the width of the distribution matters more than the shape. Here, we generate random 1D signals from $1/f^{\alpha}$ noise and report the test-set accuracy of different trained models that use a sparse set (16 out of 1024) of random Fourier features sampled from different distributions. Each subplot represents a different family of 1D signals. Each dot represents a trained network, where the color indicates which Fourier feature sampling distribution is used. We plot the test error of each model versus the empirical standard deviation of its sampled frequencies. The best models using sparsely sampled features are able to match the performance of a model trained with dense Fourier features (dashed lines with error bars). All sampling distributions trace out the same curve, exhibiting underfitting (slow convergence) when the standard deviation of sampled frequencies is too low and overfitting when it is too high. This implies that the precise shape of the distribution used to sample frequencies does not have a significant impact on performance.

proceed with a Gaussian distribution for our higher-dimensional experiments in Section 6 and treat the scale as a hyperparameter to tune on a validation dataset.

6 Experiments

We validate the benefits of using Fourier feature mappings for coordinate-based MLPs with experiments on a variety of regression tasks relevant to the computer vision and graphics communities.

6.1 Compared mappings

In Table 1, we compare the performance of coordinate-based MLPs with no input mapping and with the following Fourier feature mappings ($\cos, \sin$ are applied elementwise):

Basic: $\gamma(\mathbf{v}) = [\cos(2\pi \mathbf{v} v), \sin(2\pi \mathbf{v})]^{\mathrm{T}}$. Simply wraps input coordinates around the circle.

Positional encoding: $\gamma(\mathbf{v}) = \left[\ldots, \cos(2\pi \sigma^{j/m} \mathbf{v}), \sin(2\pi \sigma^{j/m} \mathbf{v}), \ldots\right]^{\mathrm{T}}$ for $j = 0, \ldots, m-1$. Uses log-linear spaced frequencies for each dimension, where the scale σ is chosen for each task and dataset by a hyperparameter sweep. This is a generalization of the "positional encoding" used by prior work [27, 39, 44]. Note that this mapping is deterministic and only contains on-axis frequencies, making it naturally biased towards data that has more frequency content along the axes.

Gaussian: $\gamma(\mathbf{v}) = [\cos(2\pi \mathbf{B}\mathbf{v}), \sin(2\pi \mathbf{B}\mathbf{v})]^{\mathrm{T}}$, where each entry in $\mathbf{B} \in \mathbb{R}^{m \times d}$ is sampled from $\mathcal{N}(0, \sigma^2)$, and σ is chosen for each task and dataset with a hyperparameter sweep. In the absence of any strong prior on the frequency spectrum of the signal, we use an isotropic Gaussian distribution.

Our experiments show that all of the Fourier feature mappings improve the performance of coordinate-based MLPs over using no mapping and that the Gaussian RFF mapping performs best.

6.2 Tasks

We conduct experiments with direct regression, where supervision labels are in the same space as the network outputs, as well as indirect regression, where the network outputs are passed through a forward model to produce observations in the same space as the supervision labels (Appendix D contains a theoretical analysis of indirect regression through a linear forward model). For each task and dataset, we tune Fourier feature scales on a held-out set of signals. For each target signal, we train an MLP on a training subset of the signal and compute error over the remaining test subset. All tasks

	Direct supervision			Indirect supervision			
	2D image		3D shape [24]	2D CT		3D MRI	3D NeRF [27]
	Natural	Text		Shepp	ATLAS	ATLAS	
No mapping	19.32	18.40	0.864	16.75	15.44	26.14	22.41
Basic	21.71	20.48	0.892	23.31	16.95	28.58	23.16
Positional enc.	24.95	27.57	0.960	26.89	19.55	32.23	25.28
Gaussian	**25.57**	**30.47**	**0.973**	**28.33**	**19.88**	**34.51**	**25.48**

Table 1: We compare four different input mappings on a variety of low-dimensional regression tasks. All results are reported in PSNR except *3D shape*, which uses IoU (higher is better for all). *No mapping* represents using a standard MLP with no feature mapping. *Basic*, *Positional encoding*, and *Gaussian* are different variants of Fourier feature maps. For the *Direct supervision* tasks, the network is supervised using ground truth labels for each input coordinate. For the *Indirect supervision* tasks, the network outputs are passed through a forward model before the loss is applied (integral projection for CT, the Fourier transform for MRI, and nonlinear volume rendering for NeRF). Fourier feature mappings improve results across all tasks, with random Gaussian features performing best.

(except 3D shape regression) use L2 loss and a ReLU MLP with 4 layers and 256 channels. The 3D shape regression task uses cross-entropy loss and a ReLU MLP with 8 layers and 256 channels. We apply a sigmoid activation to the output for each task (except the view synthesis density prediction). We use 256 frequencies for the feature mapping in all experiments (see Appendix A.2 for experiments that investigate the effects of network depth and feature mapping sparsity). Appendix E provides additional details on each task and our implementations, and Appendix F shows more result figures.

2D image regression. In this task, we train an MLP to regress from a 2D input pixel coordinate to the corresponding RGB value of an image. For each test image, we train an MLP on a regularly-spaced grid containing $1/4$ of the pixels and report test error on the remaining pixels. We compare input mappings over a dataset of natural images and a dataset of text images.

3D shape regression. Occupancy Networks [24] implicitly represent a 3D shape as the "decision boundary" of an MLP, which is trained to output 0 for points outside the shape and 1 for points inside the shape. Each batch of training data is generated by sampling points uniformly at random from the bounding box of the shape and calculating their labels using the ground truth mesh. Test error is calculated using intersection-over-union versus ground truth on a set of points randomly sampled near the mesh surface to better highlight the different mappings' abilities to resolve fine details.

2D computed tomography (CT). In CT, we observe integral projections of a density field instead of direct measurements. In our 2D CT experiments, we train an MLP that takes in a 2D pixel coordinate and predicts the corresponding volume density at that location. The network is indirectly supervised by the loss between a sparse set of ground-truth integral projections and integral projections computed from the network's output. We conduct experiments using two datasets: procedurally-generated Shepp-Logan phantoms [36] and 2D brain images from the ATLAS dataset [21].

3D magnetic resonance imaging (MRI). In MRI, we observe Fourier transform coefficients of atomic response to radio waves under a magnetic field. In our 3D MRI experiments, we train an MLP that takes in a 3D voxel coordinate and predicts the corresponding response at that location. The network is indirectly supervised by the loss between a sparse set of ground-truth Fourier transform coefficients and Fourier transform coefficients computed from discretely querying the MLP on a voxel grid. We conduct experiments using the ATLAS dataset [21].

3D inverse rendering for view synthesis. In view synthesis, we observe 2D photographs of a 3D scene, reconstruct a representation of that scene, then render images from new viewpoints. To perform this task, we train a coordinate-based MLP that takes in a 3D location and outputs a color and volume density. This MLP is indirectly supervised by the loss between the set of 2D image observations and the same viewpoints re-rendered from the predicted scene representation. We use a simplified version of the method described in NeRF [27], where we remove hierarchical sampling and view dependence and replace the original positional encoding with our compared input mappings.

7 Conclusion

We leverage NTK theory to show that a Fourier feature mapping can make coordinate-based MLPs better suited for modeling functions in low dimensions, thereby overcoming the spectral bias inherent in coordinate-based MLPs. We experimentally show that tuning the Fourier feature parameters offers control over the frequency falloff of the combined NTK and significantly improves performance across a range of graphics and imaging tasks. These findings shed light on the burgeoning technique of using coordinate-based MLPs to represent 3D shapes in computer vision and graphics pipelines, and provide a simple strategy for practitioners to improve results in these domains.

Acknowledgements

We thank Ben Recht for advice, and Cecilia Zhang and Tim Brooks for their comments on the text. BM is funded by a Hertz Foundation Fellowship and acknowledges support from the Google BAIR Commons program. MT, PS and SFK are funded by NSF Graduate Fellowships. RR was supported in part by ONR grants N000141712687 and N000142012529 and the Ronald L. Graham Chair. RN was supported in part by an FHL Vive Center Seed Grant. Google University Relations provided a generous donation of compute credits.

References

[1] Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. *CVPR Workshops*, 2017.

[2] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *ICML*, 2019.

[3] Ronen Basri, Meirav Galun, Amnon Geifman, David Jacobs, Yoni Kasten, and Shira Kritchman. Frequency bias in neural networks for input of non-uniform density. *arXiv preprint arXiv:2003.04560*, 2020.

[4] Ronen Basri, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *NeurIPS*, 2019.

[5] Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. *NeurIPS*, 2019.

[6] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. *arXiv preprint arXiv:2002.02561*, 2020.

[7] R. N. Bracewell. Strip integration in radio astronomy. *Australian Journal of Physics*, 1956.

[8] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. `http://github.com/google/jax`.

[9] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *CVPR*, 2019.

[10] Boyang Deng, JP Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Neural articulated shape approximation. *arXiv preprint arXiv:1912.03207*, 2019.

[11] Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *ICLR*, 2019.

[12] Kyle Genova, Forrester Cole, Aaron Sarna Daniel Vlasic, William T. Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. *ICCV*, 2019.

[13] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3D shape. *CVPR*, 2020.

[14] Reinhard Heckel and Mahdi Soltanolkotabi. Compressive sensing with un-trained neural networks: Gradient descent finds the smoothest approximation. *arXiv preprint arXiv:2005.03991*, 2020.

[15] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. *CVPR*, 2020.

[16] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural Tangent Kernel: Convergence and generalization in neural networks. *NeurIPS*, 2018.

[17] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3D scenes. *CVPR*, 2020.

[18] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[20] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *NeurIPS*, 2019.

[21] Sook-Lei Liew, Julia M. Anglin, Nick W. Banks, Matt Sondag, Kaori L. Ito, Kim, et al. A large, open source dataset of stroke anatomical brain images and manual lesion segmentations. *Scientific Data*, 2018.

[22] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. *CVPR*, 2020.

[23] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3D supervision. *NeurIPS*, 2019.

[24] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. *CVPR*, 2019.

[25] Michael Dawson-Haggerty et al. trimesh, 2019. `https://trimsh.org/`.

[26] Mateusz Michalkiewicz, Jhony K Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. *ICCV*, 2019.

[27] 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. *arXiv preprint arXiv:2003.08934*, 2020.

[28] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *CVPR*, 2015.

[29] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. *CVPR*, 2020.

[30] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in Python. *ICLR*, 2020.

[31] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. *ICCV*, 2019.

[32] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. *CVPR*, 2019.

[33] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. *ICML*, 2019.

[34] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *NeurIPS*, 2007.

[35] Shunsuke Saito, , Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. *ICCV*, 2019.

[36] Lawrence A. Shepp and Benjamin F. Logan. The Fourier reconstruction of a head section. *IEEE Transactions on nuclear science*, 1974.

[37] Vincent Sitzmann, Michael Zollhoefer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. *NeurIPS*, 2019.

[38] Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 2007.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[40] Martin J. Wainwright. *Reproducing Kernel Hilbert Spaces*, page 383–415. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019.

[41] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. Embree: a kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics (TOG)*, 2014.

[42] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Self-attention with functional time representation learning. *NeurIPS*, 2019.

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

[44] Ellen D. Zhong, Tristan Bepler, Joseph H. Davis, and Bonnie Berger. Reconstructing continuous distributions of 3D protein structure from cryo-EM images. *ICLR*, 2020.

A Further experiments

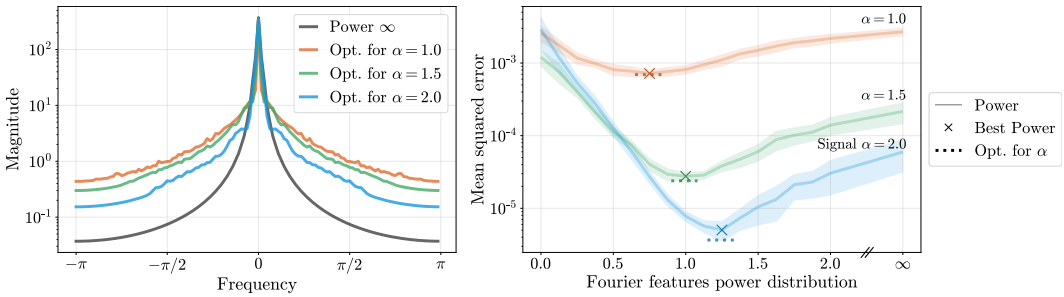
A.1 Optimizing validation error through the NTK linear dynamics

Using Eqn. 3 in the main paper, we can predict what error a trained network will achieve on a set of testing points. Since this equation depends on the composed NTK, we can directly relate predicted test set loss to the Fourier feature mapping parameters a and b for a validation set of signals $\mathbf{y}_{val}$:

$$\mathcal{L}_{\text{opt}} = \left\| \mathbf{u}^{(t)} - \mathbf{y}_{\text{val}} \right\|_2^2 \approx \left\| \mathbf{K}_{\text{val}} \mathbf{K}^{-1} \left(\mathbf{I} - e^{-\eta \mathbf{K} t} \right) \mathbf{y} - \mathbf{y}_{\text{val}} \right\|_2^2, \tag{10}$$

where $\mathbf{K}_{\text{val}}$ is the composed NTK evaluated between points in a validation dataset $\mathbf{X}_{\text{val}}$ and training dataset $\mathbf{X}$, and η and t are the learning rate and number of iterations that will be used when training the actual network.

In Figure 5, we show the results of minimizing Eqn. 10 by gradient descent on a_j values (with fixed corresponding "densely sampled" $b_j = j$) for validation sets sampled from three different $1/f^{\alpha}$ noise families. Note that gradient descent on this theoretical loss approximation produces a_j values which are able to perform as well as the best "power law" a_j values for each respective signal class (compared dashed lines versus $\times$ markers in Figure 5b). As mentioned in the main text, we find that this optimization strategy is only viable for small 1D regression problems. In our multidimensional tasks, using densely sampled $\mathbf{b}_j$ values is not tractable due to memory constraints. In addition, the theoretical approximation only holds when training the network using SGD, and in practice we train using the Adam optimizer [19].



(a) NTK Fourier spectrum (b) Fourier features mapping performances

Figure 5: The Fourier feature mappings can be optimized for better performance on a class of target signals by using the linearized network approximation. Here we consider target signals sampled from three different power law distributions. In (a) we show the spectrum for composed kernels corresponding to different optimized feature mappings, where the feature mappings are initialized to match the "Power ∞" distribution. In (b) we take an alternative approach where we sweep over "power law" settings for our Fourier features. We find that tuning this simple parameterization is able to perform on par with the optimized feature maps.

A.2 Feature sparsity and network depth

In our experiments, we observe that deeper networks need fewer Fourier features than shallow networks. As the depth of the MLP increases, we observe that a sparser set of frequencies can achieve similar performance; Figure 6 illustrates this effect in the context of 2D image regression.

Again drawing on NTK theory, we understand this tradeoff as an effect of frequency "spreading," as illustrated in Figure 7. A Fourier featurization consists of only discrete frequencies, but when composed with the NTK, the influence of each discrete frequency "spreads" over its local neighborhood in the final spectrum. We find that the "spread" around each frequency feature increases for deeper networks. For an MLP to learn all of the frequency components in the target signal, its corresponding composed NTK must contain adequate power across the frequency support of the target signal. This is accomplished either by including more frequencies in the Fourier features or by spreading those frequencies through sufficient NTK depth.

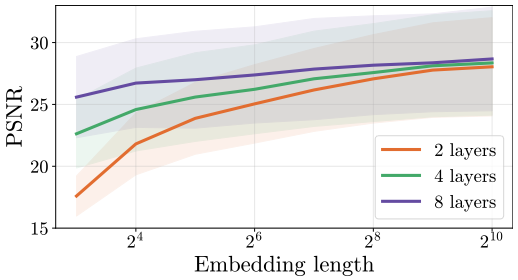
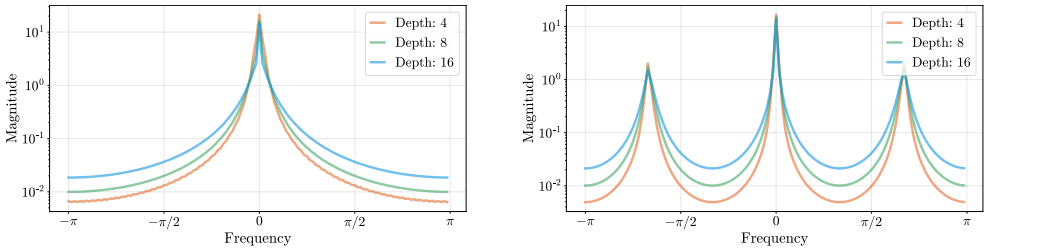


Figure 6: In a 2D image regression task (explained in Section E.1) we find that shallower networks require more Fourier features than deeper networks. This is explained by the frequency spreading effect shown in Figure 7. In this experiment we use the *Natural* image dataset and a Gaussian mapping. All of the network layers have 256 channels, and the networks are trained using an Adam [19] optimizer with a learning rate of 10^{-3}.

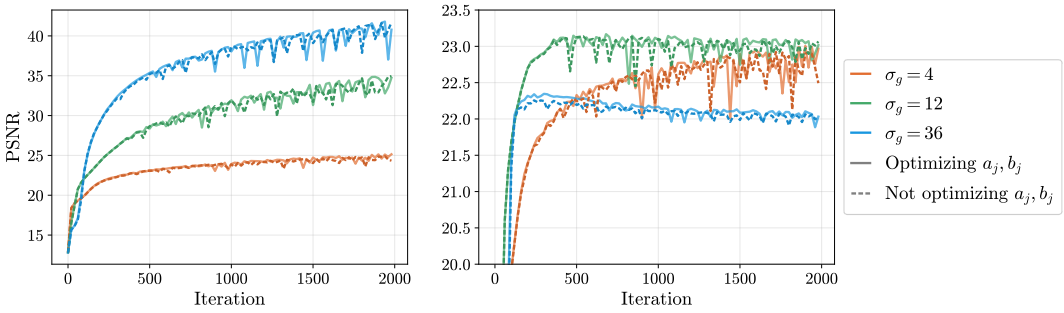


(a) NTK Fourier spectrum with basic mapping (b) NTK Fourier spectrum with basic mapping and an additional frequency

Figure 7: Each frequency included in a Fourier embedding is "spread" by the NTK, with deeper NTKs causing more frequency spreading. We posit that this frequency spreading is what enables an MLP with a sparse set of Fourier features to faithfully reconstruct a complex signal, which would be poorly reconstructed by either sparse Fourier feature regression or a plain coordinate-based MLP.

A.3 Gradient descent does not optimize Fourier features

One may wonder if the Fourier feature mapping parameters a_j and $\mathbf{b}_j$ can be optimized alongside network weights using gradient descent, which may circumvent the need for careful initialization. We performed an experiment in which the $a_j, \mathbf{b}_j$ values are treated as trainable variables (along with the weights of the network) and optimize all variables with Adam to minimize training loss. Figure 8 shows that jointly optimizing these parameters does not improve performance compared to leaving them fixed.

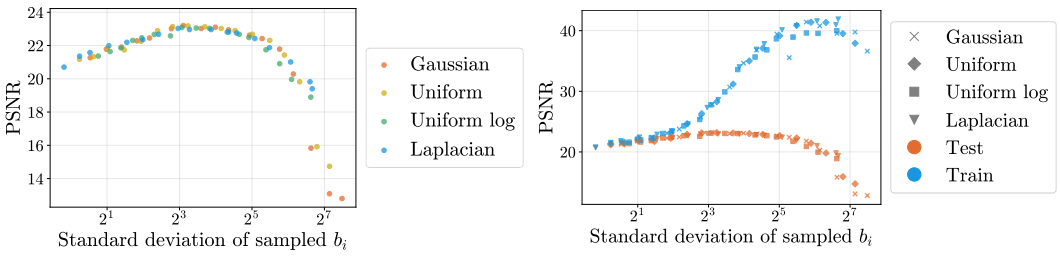


(a) Train (b) Test

Figure 8: "Training" the Fourier feature mapping parameters a_j and $\mathbf{b}_j$ along with the network weights using Adam does not improve performance, as the $\mathbf{b}_j$ values do not deviate significantly from their initial values. We show that this holds when $\mathbf{b}_j$ are initialized at three different scales of Gaussian Fourier features in the case of the 2D image task (a_j are always initialized as 1).

A.4 Visualizing underfitting and overfitting in 2D

Figure 4 in the main text shows (in a 1D setting) that as the scale of the Fourier feature sampling distribution increases, the trained network's error traces out a curve that starts in an underfitting regime (only low frequencies are learned) and ends in an overfitting regime (the learned function includes high-frequency detail not present in the training data). In Figure 9, we show analogous behavior for 2D image regression, demonstrating that the same phenomenon holds in a multidimensional problem. In Figure 10, we show how changing the scale for Gaussian Fourier features qualitatively affects the final result in the 2D image regression task.



(a) Test error for 2D image task

(b) Train and test error for 2D image task

Figure 9: An alternate version of Figure 4 from the main text where the underlying signal is a 2D image (see 2D image task details in Section E.1) instead of 1D signal. This multi-dimensional case exhibits the same behavior as was seen in the 1D case: we see the same underfitting/overfitting pattern for four different isotropic Fourier feature distributions, and the distribution shape matters less than the scale of sampled b_i values.

$\sigma = 1$ $\sigma = 2$ $\sigma = 10$ $\sigma = 32$ $\sigma = 64$

Figure 10: A visualization of the 2D image regression task with different Gaussian scales (corresponding to points along the curve shown in Figure 9). Low values of σ underfit, resulting in oversmoothed interpolation, and large values of σ overfit, resulting in noisy interpolation. We find that $\sigma = 10$ performs best for our *Natural* image dataset.

A.5 Failures of positional encoding (axis-aligned bias)

Here we present a simple experiment to directly showcase the benefits of using an isotropic frequency distribution, such as Gaussian RFF, compared to the axis-aligned "positional encoding" used in prior work [27, 44]. As discussed in the main paper, the positional encoding mapping only uses on-axis frequencies. This approach is well-suited to data that has more frequency content along the coordinate axes, but is not as effective for more natural signals.

In Figure 11, we conduct a simple 2D image experiment where we train a coordinate-based MLP (2 layers, 256 channels) to fit target 2D sinusoid images (512×512 resolution). We sample 64 such 2D sinusoid images (regularly-sampled in polar coordinates, with 16 angles and 4 radii) and train a 2D coordinate-based MLP to fit each, using the same setup as the 2D image experiments described in Section E.1. The isotropic Gaussian RFF mapping performs well across all angles, while the positional encoding mapping performs worse for frequencies that are not axis-aligned.

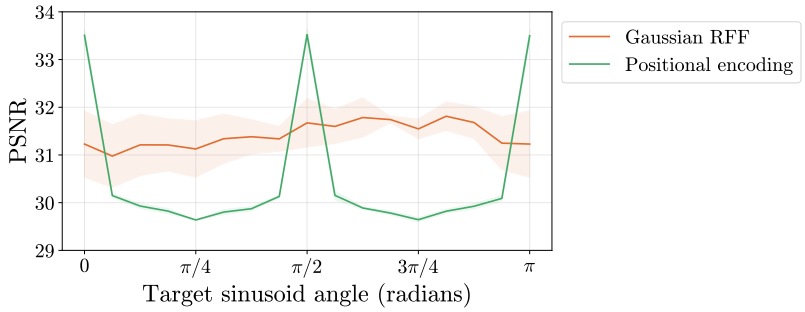


Figure 11: We train a coordinate-based MLP to fit target 2D images consisting of simple sinusoids at different frequencies and angles. The positional encoding mapping performs well at on-axis angles and performs worse on off-axis angles, while the Gaussian RFF mapping performs similarly well across all angles (results are averaged over radii). Error bars are plotted over runs with different randomly-sampled frequencies for the Gaussian RFF mapping, while positional encoding is deterministic.

B Additional details for main text figures

B.1 Main text Figure 3 (effect of feature mapping on convergence speed)

In Figure 12, we present an alternate version of Figure 3 from the main text showing a denser sampling of p values to better visualize the effect of changing Fourier feature falloff on the resulting trained network. Again, the feature mapping used here is $a_j = 1/j^p, b_j = j$ for $j = 1, \ldots, n/2$.

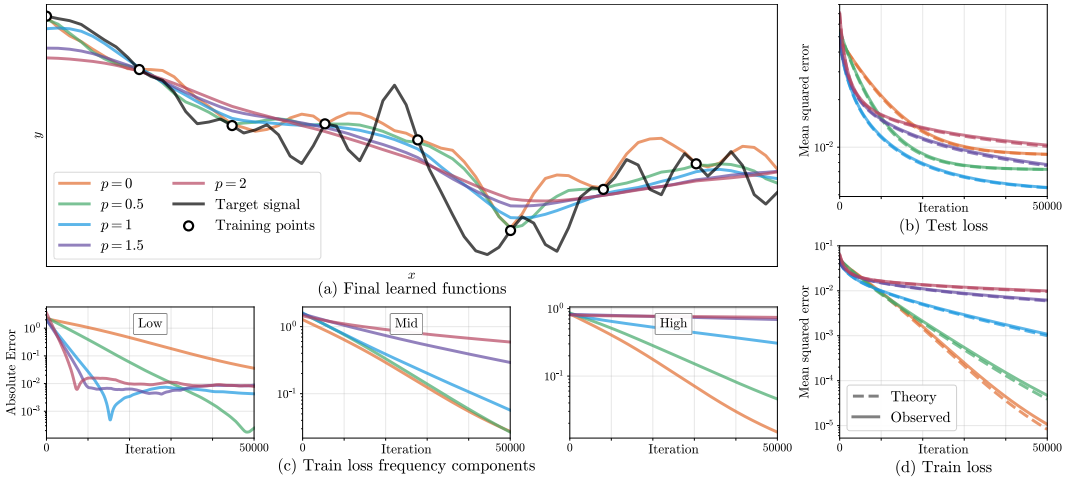


Figure 12: An extension of Figure 3 from the main paper, showing more values of p. In (c) we see that mappings with more gradual frequency falloff (lower p) converge significantly faster in mid and high frequencies, resulting in faster overall training convergence (d). In (b) we see that $p = 1$ achieves a lower test error than the other mappings.

B.2 Main text Figure 4 (different random feature distributions in 1D)

Exact details for the sampling distributions used to generate b_j values for Figure 4 in the main text are shown in Table 2. In Figure 13, we present an alternate version showing both train and test performance, emphasizing the underfitting/overfitting regimes created by manipulating the scale of the Fourier features.

Uniform log distribution We include the *Uniform log* distribution because it is the random equivalent of the "positional encoding" sometimes used in prior work. One observation is that the sampling

for uniform-log variables ($X' = \sigma_{ul}^X$ where $X \sim \mathcal{U}[0, 1)$) corresponds to the following CDF:

$$P(X' \leq x) = \frac{\log x}{\log \sigma_{ul}}, \quad \text{for } x \in [1, \sigma_{ul}) \,, \tag{11}$$

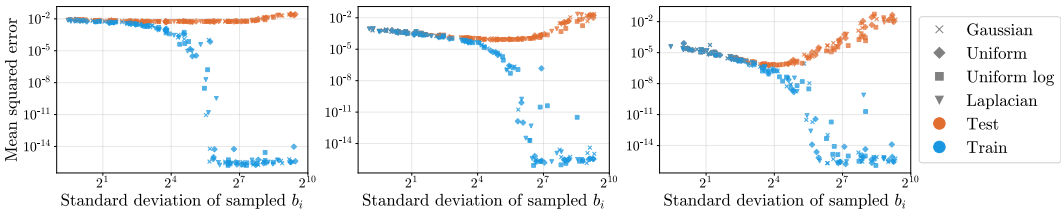
which has the following PDF:

$$p(x) = \frac{d}{dx} P(X' \leq x) = \frac{1}{x \log \sigma_{ul}} \,. \tag{12}$$

This shows that the randomized equivalent of positional encoding is sampling from a distribution proportional to a $1/f$ falloff power law.

Name	Sampled b_j values
Gaussian	$\sigma_g X$ for $X \sim \mathcal{N}(0, 1)$
Uniform	$\sigma_u X$ for $X \sim \mathcal{U}[0, 1)$
Uniform log	σ_{ul}^X for $X \sim \mathcal{U}[0, 1)$
Laplacian	$\sigma_l X$ for $X \sim \text{Laplace}(0, 1)$
Positional Enc.	$2^{\sigma_p X}$ for $X \in \text{linspace}(0, 1)$ (deterministic)

Table 2: Different distributions used for sampling frequencies, where σ is each distribution's "scale".



(a) Data sampled from $\alpha = 0.5$ (b) Data sampled from $\alpha = 1.0$ (c) Data sampled from $\alpha = 1.5$

Figure 13: An alternate version of Figure 4 from the main text showing both training error and test error for a variety of different Fourier feature sampling distributions. Adding training error to the plot clearly distinguishes between the underfitting regime with low frequency b_i (where train and test error are similar) versus the overfitting regime with high frequency b_i (where the test error increases but training error approaches machine precision).

C Stationary kernels

One of the primary benefits of our Fourier feature mapping is that it results in a *stationary* composed NTK function. In this section, we offer some intuition for why stationarity is desirable for our low-dimensional graphics and imaging problems.

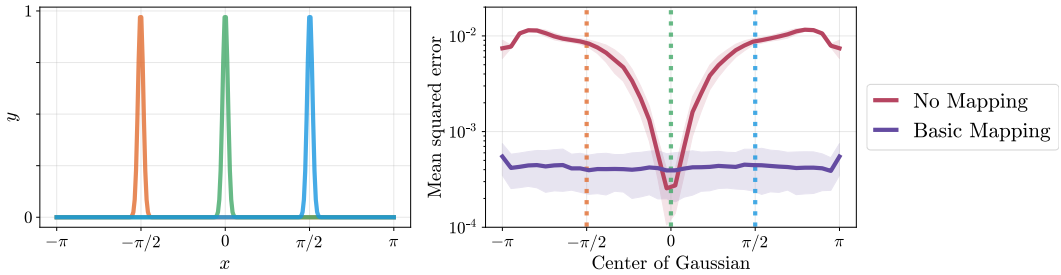
First, let us consider the implications of using an MLP applied directly to a low-dimensional input (without any Fourier feature mapping). In this setting, the NTK is a function of the dot product between its inputs and of their norms [3, 5, 6, 16]. This makes the NTK *rotation*-invariant, but not *translation*-invariant. For our graphics and imaging applications, we want to be able to model an object or scene equally well regardless of its location, so translation-invariance or *stationarity* is a crucial property. We can then add approximate rotation invariance back by using an isotropic frequency sampling distribution.

This aligns with standard practice in signal processing, in which $k(\mathbf{u}, \mathbf{v}) = \tilde{h}(\mathbf{u} - \mathbf{v}) = \tilde{h}(\mathbf{v} - \mathbf{u})$ (*e.g.* the Gaussian or radial basis function kernel, or the sinc reconstruction filter kernel). This Euclidean notion of similarity based on difference vectors is better suited to the low-dimensional regime, in which we expect (and can afford) dense and nearly uniform sampling. Regression with a stationary kernel corresponds to reconstruction with a convolution filter: new predictions are sums of training points, weighted by a function of Euclidean distance.

One of the most important features of our sinusoidal input mapping is that it translates between these two regimes. If $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ for small d, γ is our Fourier feature embedding function, and k is a dot

product kernel function, then $k(\gamma(\mathbf{u}), \gamma(\mathbf{v})) = h(\gamma(\mathbf{u})^{\mathrm{T}}\gamma(\mathbf{v})) = \tilde{h}(\mathbf{u} - \mathbf{v})$. In words, our sinusoidal input mapping transforms a dot product kernel into a stationary one, making it better suited to the low-dimensional regime.

This effect is illustrated in a simple 1D example in Figure 14, which shows that the benefits of a stationary composed NTK indeed appear in the MLP setting with a basic Fourier featurization (using a single frequency). We train MLPs with and without this basic Fourier embedding to learn a set of shifted 1D Gaussian probability density functions. The plain MLP successfully fits a zero-centered function but struggles to fit shifted functions, while the MLP with basic Fourier embedding exhibits stationary behavior, with good performance regardless of shifts.



(a) Example target signals (b) Reconstruction accuracy

Figure 14: A plain coordinate-based MLP can learn a centered function (in this case a Gaussian density) but struggles to model shifts of the same function. Adding a basic Fourier embedding (with a single frequency) enables the MLP to fit the target function equally well regardless of shifts. The NTK corresponding to the plain MLP is based on dot products between inputs, whereas the NTK corresponding to the NTK with Fourier embedding is based on Euclidean distances between inputs, making it shift-invariant. In this experiment we train an MLP (4 layers, 256 channels, ReLU activation) for 500 iterations using the Adam [19] optimizer with a learning rate of 10^{-4}. We report mean and standard deviation performance over 20 random network initializations.

D Indirect supervision through a linear map

In some of the tasks we explore in this work, such as image regression or 3D shape regression, optimization is performed by minimizing a loss between the output of a network and a directly observed quantity, such as the color of a pixel or the occupancy of a voxel. But in many graphics and imaging applications of interest, measurements are *indirect*, and the loss must be computed on the output of a network after it has been processed by some physical forward model. In NeRF [27], measurements are taken by sampling and compositing along rays in each viewing direction. In MRI, measurements are taken along various curves through the frequency domain. In CT, measurements are integral projections of the subject at various angles, which correspond to measuring lines through the origin in the frequency domain. Although the measurement transformation for NeRF is nonlinear (in density, although it is linear in color), those for both CT and MRI are linear. In this section, we extend the linearized training dynamics of Lee *et al.* [20] to the setting of training through a linear operator denoted by a matrix $\mathbf{A}$. This allows us to modify Eqn. 3 to incorporate $\mathbf{A}$, thereby demonstrating that the conclusions drawn in this work for the "direct" regression case also apply to the "indirect" case.

Our derivation closely follows Lee *et al.* [20], and begins by replacing the neural network f with its linearization around the initial parameters θ_0:

$$f_t^{\mathrm{lin}}(\mathbf{x}) \triangleq f_0(\mathbf{x}) + \nabla_\theta f_0(\mathbf{x})|_{\theta=\theta_0}\omega_t\,, \tag{13}$$

where $\omega_t \triangleq \theta_t - \theta_0$ denotes the change in network parameters since initialization and t denotes time in continuous-time gradient flow dynamics. Then [20] describes the dynamics of gradient flow:

$$\dot{f}_t^{\mathrm{lin}}(\mathbf{x}) = -\eta\hat{\Theta}_0(\mathbf{x}, \mathbf{X})\nabla_{f_t^{\mathrm{lin}}(\mathbf{X})}\mathcal{L}\,, \tag{14}$$

where $\hat{\Theta}_t(\cdot,\cdot) = \nabla_\theta f_t(\cdot)\nabla_\theta f_t(\cdot)^{\mathrm{T}}$ is the NTK matrix at time t ($\hat{\Theta}_t$ is shorthand for $\hat{\Theta}_t(\mathbf{X}, \mathbf{X})$) and $\mathcal{L}$ is the training loss. At this point, we depart slightly from the analysis of [20]: instead of

$\mathcal{L} = \sum_{(\mathbf{x},y)\in\mathcal{D}} \ell(f_t^{\text{lin}}(\mathbf{x}), y)$ we have $\mathcal{L} = \frac{1}{2}\left\|\mathbf{A}(f_t^{\text{lin}}(\mathbf{X}) - \mathbf{y})\right\|_2^2$, where $\mathbf{y}$ denotes the vector of training labels. The gradient of the loss is then

$$\nabla_{f_t^{\text{lin}}(\mathbf{X})}\mathcal{L} = \nabla_{f_t^{\text{lin}}(\mathbf{X})}\frac{1}{2}\left\|\mathbf{A}\left(f_t^{\text{lin}}(\mathbf{X}) - \mathbf{y}\right)\right\|_2^2 \tag{15}$$

$$= \mathbf{A}^{\mathrm{T}}\mathbf{A}\left(f_t^{\text{lin}}(\mathbf{X}) - \mathbf{y}\right) . \tag{16}$$

Substituting this into the gradient flow dynamics of Eqn. 14 gives us:

$$\dot{f}_t^{\text{lin}}(\mathbf{x}) = -\eta\hat{\Theta}_0(\mathbf{x},\mathbf{X})\mathbf{A}^{\mathrm{T}}\mathbf{A}\left(f_t^{\text{lin}}(\mathbf{X}) - \mathbf{y}\right) , \tag{17}$$

with corresponding solution:

$$f_t^{\text{lin}}(\mathbf{X}) = \left(\mathbf{I} - e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)\mathbf{y} + e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}f_0(\mathbf{X}) . \tag{18}$$

Finally, again following [20], we can decompose $f_t^{\text{lin}}(\mathbf{x}) = \mu_t(\mathbf{x}) + \gamma_t(\mathbf{x})$ at any test point $\mathbf{x}$, where

$$\mu_t(\mathbf{x}) = \hat{\Theta}_0(\mathbf{x},\mathbf{X})\hat{\Theta}_0^{-1}\left(\mathbf{I} - e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)\mathbf{y} , \tag{19}$$

$$\gamma_t(\mathbf{x}) = f_0(\mathbf{x}) - \hat{\Theta}_0(\mathbf{x},\mathbf{X})\hat{\Theta}_0^{-1}\left(\mathbf{I} - e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)f_0(\mathbf{X}) . \tag{20}$$

Assuming our initialization is small, *i.e.*, $f_0(\mathbf{x}) \approx 0\ \forall\mathbf{x}$, we can write our approximate linearized network output as:

$$f_t^{\text{lin}}(\mathbf{x}) \approx \hat{\Theta}_0(\mathbf{x},\mathbf{X})\hat{\Theta}_0^{-1}\left(\mathbf{I} - e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)\mathbf{y} . \tag{21}$$

In our previous analysis, we work instead with the expected or infinite-width NTK matrix $\mathbf{K}$, which is fixed throughout training. Using this notation, we have

$$\hat{\mathbf{y}}^{(t)} \approx f_t^{\text{lin}}(\mathbf{X}_{\text{test}}) \approx \mathbf{K}_{\text{test}}\mathbf{K}^{-1}\left(\mathbf{I} - e^{-\eta\mathbf{K}\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)\mathbf{y} . \tag{22}$$

This is nearly identical to Eqn. 3in the main paper, except that the convergence is governed by the spectrum of $\mathbf{K}\mathbf{A}^{\mathrm{T}}\mathbf{A}$ rather than $\mathbf{K}$ alone. If $\mathbf{A}$ is unitary, such as the Fourier transform matrix used in (densely sampled) MRI, then training should behave exactly as if we were training on direct measurements. However, if $\mathbf{A}$ is not full rank, then training will only affect the components with nonzero eigenvalues in $\mathbf{K}\mathbf{A}^{\mathrm{T}}\mathbf{A}$. In this more common scenario, we want to design a kernel that will provide large eigenvalues in the components that $\mathbf{A}$ can represent, so that the learnable components will converge quickly, and provide reasonable priors for the components we cannot learn.

In our two tasks that supervise through a linear map, CT and MRI, the $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ has a structure that illuminates how the linear map interacts with the composed NTK. The $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ matrices for both these tasks are diagonalizable by the DFT matrix, where the diagonal entries are simply the number of times the corresponding frequency is measured by the MRI or CT sampling patterns. This follows from the fact that CT and MRI measurements can both be formulated as Fourier space sampling: CT samples rotated slices in Fourier space through the origin [7] and MRI samples operator-chosen Fourier trajectories. This means that frequencies not observed by the MRI or CT sampling patterns will never be supervised during training. Therefore, it is crucial to choose a Fourier feature mapping that results in a composed NTK with a good prior on these frequencies.

E Task details

We present additional details for each task from Section 6 in the main text, including training parameters, forward models, datasets, etc. All experiments are implemented using JAX [8] and trained on a single K80 or RTX2080Ti GPU. Training a single MLP took between 10 seconds (for the 2D image task) and 30 minutes (for the inverse rendering task).

E.1 2D image

The 2D image regression tasks presented in the main text all use 512×512 resolution images. A subsampled grid of 256×256 pixels is used as training data, and an offset grid of 256×256 pixels

is used for testing. We use two image datasets: *Natural* and *Text*, each consisting of 32 images. The *Natural* images are generated by taking center crops of randomly sampled images from the Div2K dataset [1]. The *Text* images are generated by placing random strings of text with random sizes and colors on a white background (examples can be seen in Figure 15). For each dataset we perform a hyperparameter sweep over feature mapping scales on 16 images. We find that scales $\sigma_g = 10$ and $\sigma_p = 6$ work best for the *Natural* dataset and $\sigma_g = 14$ and $\sigma_p = 5$ work best for the *Text* dataset (see Table 2 for mapping definitions). In Table 3, we report model performance using the optimal mapping scale on the remaining 16 images.

	Natural	Text
No mapping	19.32 ± 2.48	18.40 ± 2.23
Basic	21.71 ± 2.71	20.48 ± 1.96
Positional enc.	24.95 ± 3.72	27.57 ± 3.07
Gaussian	$\mathbf{25.57 \pm 4.19}$	$\mathbf{30.47 \pm 2.11}$

Table 3: 2D image results (mean $\pm$ standard deviation of PSNR)

Each model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid output) is trained for 2000 iterations using the Adam [19] optimizer with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). Learning rates are manually tuned for each dataset and method. For *Natural* images a learning rate of 10^{-3} is used for the Gaussian RFF and the positional encoding, and a learning rate of 10^{-2} is used for the basic mapping and "no mapping" methods. For the *Text* images a learning rate of 10^{-3} is used for all methods.

E.2 3D shape

We evaluate the 3D shape regression task (similar to Occupancy Networks [24]) on four complex triangle meshes commonly used in computer graphics applications (*Dragon*, *Armadillo*, *Buddha*, and *Lucy*, shown in Figure 16), each containing hundreds of thousands of vertices. We train one coordinate-based MLP network to represent a single mesh rather than trying to generalize one network to encode multiple objects, since our goal is to demonstrate that a network with no mapping or the low frequency "basic" mapping cannot accurately represent even a *single* shape, let alone a whole class of objects.

We use a network with 8 layers of 256 channels each and a ReLU nonlinearity between each layer. We apply a sigmoid activation to the output. Our batch size is 32^3 points, and we use the Adam optimizer [19] with a learning rate starting at 5×10^{-4} and exponentially decaying by a factor of 0.01 over the course of 10000 total training iterations. At each training iteration, we sample a batch of 3D points uniformly at random from the bounding box of the mesh, and then calculate ground truth labels (using the point-in-mesh method implemented in the Trimesh library [25], which relies on the Embree kernel for acceleration [41]). We use cross-entropy loss to train the network to match these classification labels (0 for points outside the mesh, 1 for points inside).

The meshes are scaled to fit inside the unit cube $[0, 1]^3$ such that the centroid of the mesh is $(0.5, 0.5, 0.5)$. We use the *Lucy* statue mesh as a validation object to find optimal scale values for the positional encoding and Gaussian feature mapping. As described in the caption for Table 4, we calculate error on both a uniformly random test set and a test set that is close to the mesh surface (randomly chosen mesh vertices that have been perturbed by a random Gaussian vector with standard deviation 0.01) in order to illustrate that Fourier feature mappings provide a large benefit in resolving fine surface details. Both test sets have 64^3 points.

In Figure 16, we visualize additional results on all four meshes mentioned above (including the validation mesh *Lucy*). We render normal maps, which are computed by taking the cross product of the numerical horizontal and vertical derivatives of the depth map. The original depth map is generated by intersecting camera rays with the first 0.5 isosurface of the network. We select the Fourier feature scales for (d) and (e) by doing a hyperparameter search based on validation loss for the *Lucy* mesh in the last row and report test loss over the other three meshes (Table 4). Note that the weights for each trained MLP are only 2MB, while the triangle mesh files for the objects shown are 61MB, 7MB, 79MB, and 32MB respectively.

	Uniform points	Boundary points
No mapping	0.959 ± 0.006	0.864 ± 0.014
Basic	0.966 ± 0.007	0.892 ± 0.017
Positional enc.	0.987 ± 0.005	0.960 ± 0.011
Gaussian	$\mathbf{0.988 \pm 0.007}$	$\mathbf{0.973 \pm 0.010}$

Table 4: 3D shape results (mean $\pm$ standard deviation of intersection-over-union). *Uniform points* is an "easy" test set where points are sampled uniformly at random from the bounding box of the ground truth mesh, while *Boundary points* is a "hard" test set where points are sampled near the boundary of the ground truth mesh.

E.3 2D CT

In computed tomography (CT), we observe measurements that are integral projections (integrals along parallel lines) of a density field. We construct a 2D CT task by using ground truth 512×512 resolution images, and computing 20 synthetic integral projections at evenly-spaced angles. For each of these images, the supervision data is the set of integral projections, and the test PSNR is evaluated over the original image.

We use two datasets for our 2D CT task: randomized Shepp-Logan phantoms [36], and the ATLAS brain dataset [21]. For each dataset, we perform a hyperparameter sweep over mapping scales on 8 examples. We found that scales $\sigma_g = 4$ and $\sigma_p = 3$ work best for the *Shepp* dataset and $\sigma_g = 5$ and $\sigma_p = 5$ work best for the *ATLAS* dataset. In Table 5, we report model performance using the optimal mapping scale on a distinct set of 8 images.

	Shepp	ATLAS
No mapping	16.75 ± 3.64	15.44 ± 1.28
Basic	23.31 ± 4.66	16.95 ± 0.72
Positional enc.	26.89 ± 1.46	19.55 ± 1.09
Gaussian	$\mathbf{28.33 \pm 1.15}$	$\mathbf{19.88 \pm 1.23}$

Table 5: 2D CT results (mean $\pm$ standard deviation of PSNR).

Each model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid output) is trained for 1000 iterations using the Adam [19] optimizer with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). The learning rate is manually tuned for each method. Gaussian RFF and positional encoding use a learning rate of 10^{-3}, and the basic and "no mapping" method use a learning rate of 10^{-2}.

E.4 3D MRI

In magnetic resonance imaging (MRI), we observe measurements that are Fourier coefficients of the atomic response to radio waves under a magnetic field. We construct a toy 3D MRI task by using ground truth $96 \times 96 \times 96$ resolution volumes and randomly sampling $\sim 13\%$ of the Fourier coefficients for each volume from an isotropic Gaussian. For each of these volumes, the supervision data is the set of sampled Fourier coefficients, and the test PSNR is evaluated over the original volume.

We use the ATLAS brain dataset [21] for our 3D MRI experiments. We perform a hyperparameter sweep over mapping scales on 6 examples. We find that scales $\sigma_g = 5$ and $\sigma_p = 4$ perform best. In Table 6, we report model performance using the optimal mapping scale on a distinct set of 6 images. Each model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid output) is trained for 1000 iterations using the Adam [19] optimizer with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). We use a manually-tuned learning rate of 2×10^{-3} for each method. Results are visualized in Figure 18.

E.5 3D inverse rendering for view synthesis

In this task we use the "tiny NeRF" simplified version of the view synthesis method NeRF [27] where hierarchical sampling and view dependence have been removed. The model is trained to predict the color and volume density at an input 3D point. Volumetric rendering is used to render novel

	ATLAS
No mapping	26.14 ± 1.45
Basic	28.58 ± 2.45
Positional enc.	32.23 ± 3.08
Gaussian	$\mathbf{34.51 \pm 2.72}$

Table 6: 3D MRI results (mean $\pm$ standard deviation of PSNR).

viewpoints of the object. The loss is calculated between the rendered views and ground truth renders. In our experiments we use the NeRF *Lego* dataset of 120 images downsampled to 400×400 pixel resolution. The dataset is split into 100 training images, 7 validation images, and 13 test images. The reconstruction quality on the validation images is used to determine the best mapping scale; for this scene we find $\sigma_g = 6.05$ and $\sigma_p = 1.27$ perform best.

The model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid on RGB output) is trained for 5×10^5 iterations using the Adam [19] optimizer with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). The learning rate is manually tuned for each mapping: 10^{-2} for no mapping, 5×10^{-3} for basic, 5×10^{-4} for positional encoding, and 5×10^{-4} for Gaussian. During training we use batches of 1024 rays.

The original NeRF method [27] uses an input mapping similar to the *Positional encoding* we compare against. The original NeRF mapping is smaller than our mappings (8 vs. 256 frequencies). We include metrics for this mapping in Table 7 under *Original pos. enc*. The positional encoding mappings only contain frequencies on the axes, and are therefore biased towards signals with on-axis frequency content (as demonstrated in Section A.5). In our experiments we rotate the *Lego* scene, which was manually axis-aligned in the original dataset, for a more equitable comparison. Table 7 also reports metrics for positional encodings on the original axis-aligned scene. Results are visualized in Figure 19.

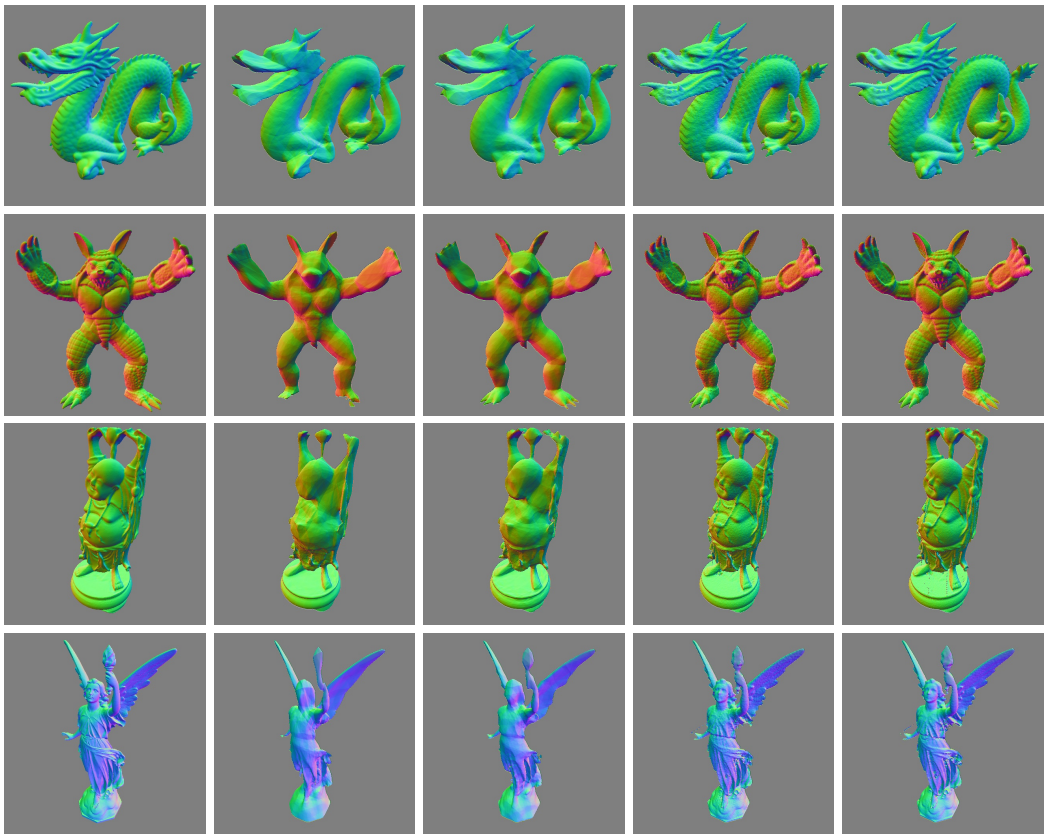
	3D NeRF
No mapping	22.41 ± 0.92
Basic	23.16 ± 0.90
Original pos. enc.	24.81 ± 0.88
Positional enc.	25.28 ± 0.83
Gaussian	$\mathbf{25.48 \pm 0.89}$
Original pos. enc. (axis-aligned)	25.60 ± 0.76
Positional enc. (axis-aligned)	26.27 ± 0.91

Table 7: 3D NeRF results (mean and standard deviation of PSNR). Error is calculated based on held-out images of the scene since the ground truth radiance field is not known.

F Additional results figures

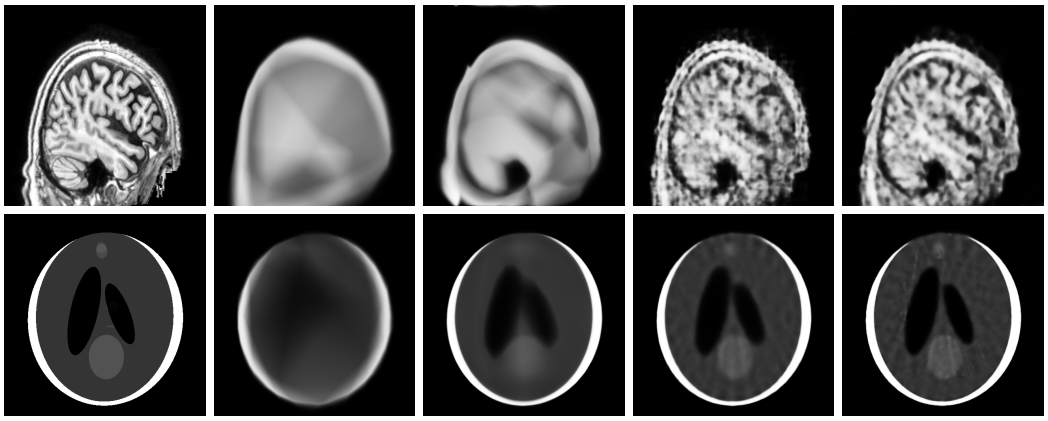
(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 15: Additional results for the 2D image regression task, for three images from our *Natural* dataset (top) and two images from our *Text* dataset (bottom).



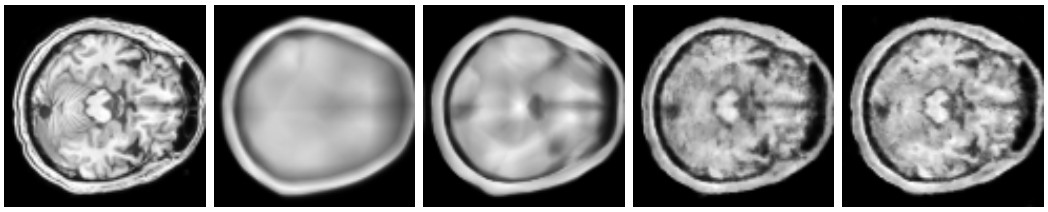
(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 16: Additional results for the 3D shape occupancy task [24].



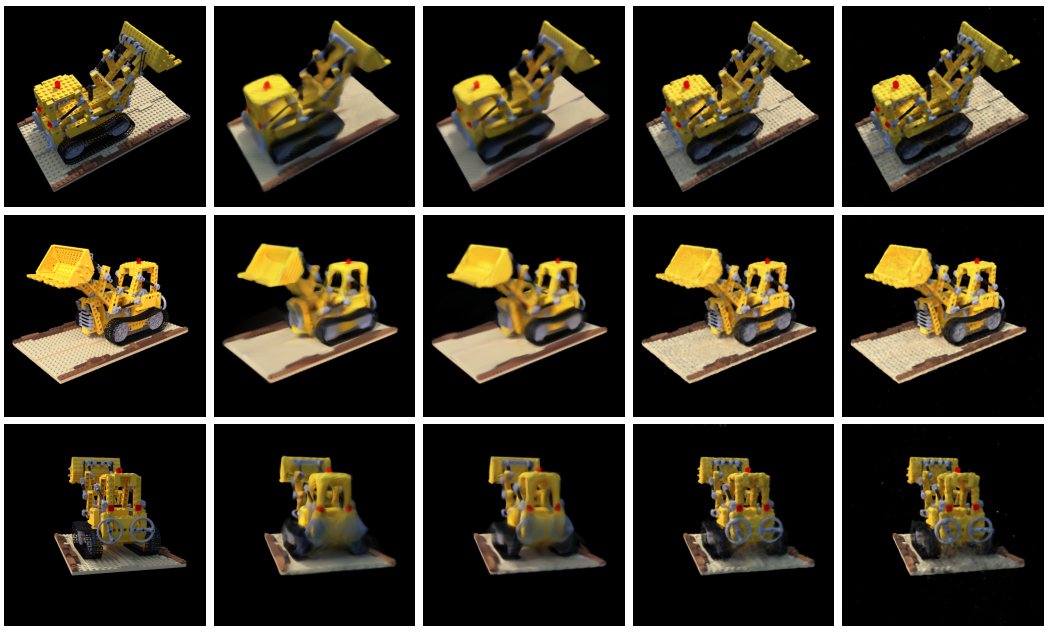
(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 17: Results for the 2D CT task.



(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 18: Additional results for the 3D MRI task.



(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 19: Additional results for the inverse rendering task [27].