Curvature-Aware Training for Coordinate Networks

Hemanth Saratchandran[*1] Shin-Fang Chng[*1] Sameera Ramasinghe[2]
Lachlan MacDonald[1] Simon Lucey[1]
[1] Australian Institute of Machine Learning, University of Adelaide.
[2] Amazon, Australia.

Abstract

Coordinate networks are widely used in computer vision due to their ability to represent signals as compressed, continuous entities. However, training these networks with first-order optimizers can be slow, hindering their use in real-time applications. Recent works have opted for shallow voxel-based representations to achieve faster training, but this sacrifices memory efficiency. This work proposes a solution that leverages second-order optimization methods to significantly reduce training times for coordinate networks while maintaining their compressibility. Experiments demonstrate the effectiveness of this approach on various signal modalities, such as audio, images, videos, shape and neural radiance fields (NeRF).

1. Introduction

Coordinate networks [39], or implicit neural functions [35], achieve state-of-the-art results in multidimensional signal reconstruction tasks, such as image synthesis [37, 6], geometry [36, 21], and robotics [16, 5]. However, coordinate networks admitting traditional activation functions (e.g., ReLU, sigmoid, and tanh) fail to capture high-frequency details due to spectral bias [29]. To overcome this limitation, positional embedding layers [40] are often added, but they can produce noisy first-order gradients that hinder architectures requiring backpropagation [17, 8]. A recent alternative approach is to use non-traditional activation functions, such as sine [35] or Gaussian [30] activations, which enable high-frequency encoding without positional embedding layers. The major benefit of these activations over positional embedding layers is their well-behaved gradients [35, 30].

Although coordinate networks have shown remarkable performance in signal reconstruction tasks, they are typically trained using first-order optimizers like Adam, leading to slow training times. Consequently, some in the vision community have resorted to using shallow voxel-based representations [10, 44, 3, 38], despite their drawbacks such as high memory usage and lack of implicit architectural bias.

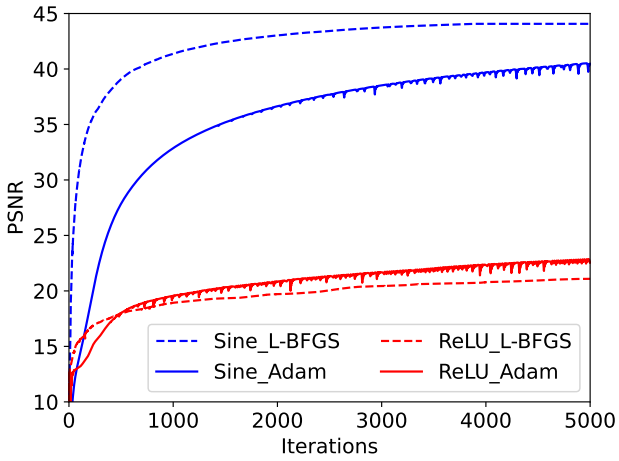


Figure 1: Sine- and ReLU-coordinate networks were compared on an image reconstruction task using L-BFGS and Adam optimizers. The L-BFGS optimizer showed faster convergence for the sine-network, while the ReLU-network converged faster with Adam.

In this paper, we present an intriguing revelation that a new breed of coordinate networks [35, 30], activated by sine and Gaussian functions, can be trained efficiently using second-order optimizers such as L-BFGS [24]. This is because their loss landscapes exhibit favorable gradient and curvature conditioning, which leads to superlinear convergence, in contrast to the linear convergence seen with Adam. Fig. 1 showcases this point by comparing sine- and ReLU-coordinate networks trained with an L-BFGS optimizer [24], a curvature-aware second-order optimizer, and an Adam optimizer. The convergence rate of the sine-network trained with L-BFGS is significantly faster than the one trained with Adam – highlighting the good curvature

[1]*Equal contribution. Correspondence to: Hemanth Saratchandran <hemanth.saratchandran@adelaide.edu.au>, Shin-Fang Chng <shinfang.chng@adelaide.edu.au>. Source code will be available at https://github.com/sfchng/curvature-aware-INRs.git

properties of the loss landscape. In contrast, the ReLU-network trained with Adam has faster convergence rate than the one trained with L-BFGS, a manifestation of the poor curvature properties of its loss landscape.

However, one of the downsides of second-order optimizers is their computational complexity when dealing with a large number of parameters. We explore this issue in the context of coordinate networks and demonstrate that, as the network size grows, Adam may outperform L-BFGS in terms of training time. To address this challenge, we propose a novel strategy of breaking down large-scale datasets into smaller patches and training a single coordinate network with a second-order optimizer on each patch. Our experiments reveal that this approach can lead to training time accelerations of up to $6-14$ times faster than Adam, and serves as an effective remedy for modelling larger size signals.

A summary of our contributions are:-

1. Our paper is the first to examine the training of coordinate networks using L-BFGS and theoretically show that while superlinear convergence is guaranteed for networks activated by sine or Gaussian functions, it is not generally guaranteed for ReLU (with or without positional embedding).

2. We validate this theory empirically by showing that sine-/Gaussian-activated coordinate networks are up to 5 times faster when trained with L-BFGS over Adam. We present results on image, audio, video, shape and neural radiance field reconstruction tasks.

3. We explore a patch-based decomposition strategy to limit the considerable computational cost of L-BFGS as the size of the signal or network grows. Specifically, we demonstrate that a sine-activated patch-based NeRF (i.e. KiloNeRF [32]) trained with L-BFGS is 6 times more efficient than the same network trained with Adam.

2. Related Work

Coordinate Networks [39] also known as implicit neural functions [35], have gained increasing interest in recent years due to the seminal work by Mildenhall et al. [21]. Unlike conventional neural networks that operate on high-dimensional inputs and are primarily used for classification tasks, coordinate networks encode signals as weights using low-dimensional coordinates and aim to preserve smoothness in the outputs [44]. One of the remarkable aspects of Mildenhall et al.'s work is their demonstration of the generalization properties of neural signal representations, which ushered in a huge body of work on the subject in recent years [7, 9, 12, 23, 25, 26, 27, 31, 33, 35, 8, 41, 42, 34, 45, 4]. However, to achieve optimal performance, such networks need to use positional embeddings to encode high-frequency signal content [44]. Sitzmann et al. [35] proposed SIREN, a sine-activated network, that can improve the fidelity of signals without positional embedding layers. However, a disadvantage of SIREN is that it needs a principled initialization scheme [35]. Ramasinghe and Lucey [30] introduced a Gaussian-activated coordinate network that, like SIREN, achieved state-of-the-art performance on signal reconstruction but is robust to random initialization schemes.

Optimizaton of Neural Networks is a complex topic with a rich history. Initially, practitioners used gradient descent due to its ease of use and low memory requirements. However, as more sophisticated models emerged, variants of gradient descent were developed to accommodate larger models. While second-order optimization methods offer superior convergence in theory [24], they are computationally expensive and not easily applicable to stochastic sampling strategies. To overcome these challenges, researchers have developed second-order optimizers that yield superior results compared to standard first-order ones, such as K-FAC [19], variants of L-BFGS [2, 43, 22], Shampoo [13], and GGT [1].

3. Preliminaries

3.1. Coordinate Multi-Layer Perceptrons (MLPs)

Coordinate-MLPs are a new class of neural networks which encode signals as weights using low dimensional coordinates as inputs. A coordinate-MLP with L layers, $f : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$ can be defined as

$$f : x \rightarrow T_L \circ \psi \circ T_{L-1} \circ \cdots, \circ \psi \circ T_1(x), \tag{1}$$

where $T_i : x_i \rightarrow A_i x_i + b_i$ is an affine transformation with trainable parameters $A_i \in \mathbb{R}^{n_{i-1} \times n_i}$, $b_i \in \mathbb{R}^i$, and ψ is a non-linear activation. The layer widths of the network are given by the numbers $\{n_1, n_2, \ldots, n_L\}$.

All our networks will be trained with the Mean Squared Error (MSE) loss function. Given N training samples $\{(x_i, y_i)\}_{i=1}^N$, where x_i and y_i denotes the input data and target data, respectively, we write the MSE loss function as

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} |f(\theta, x_i) - y_i|^2, \tag{2}$$

where f denotes a coordinate-MLP, and θ denotes the parameters of f, i.e. the weights and biases (W, b).

We briefly discuss commonly used coordinate-MLPs.

ReLU-MLPs are popular due to their universal approximation capabilities [14, 18], but they suffer from spec-

tral bias [29]. This bias can cause a preference for low-frequency components, leading to suboptimal signal reconstruction, particularly for signals with high-frequency components.

Positional encoded MLPs (ReLU-PE) avoid the spectral bias of ReLU-MLPs by adding a positional embedding layer (**PE**) to the network. This involves embedding low-dimensional data inputs $\mathbf{x}$ into a higher-dimensional space using an embedding layer $\gamma : \mathbb{R}^d \to \mathbb{R}^{d+D}$. Popular embedding layers include Fourier feature embeddings [40] and Gaussian embeddings [44].

Sine-MLPs are a positional embedding free coordinate network [35] that employ a sine activation function $\mathbf{x} \to \sin(2\pi\omega\mathbf{x})$, where ω is a frequency hyperparameter. A larger ω increases the frequency of the network allowing it to learn high-frequency targets, overcoming spectral bias.

Gaussian-MLPs are another class of embedding-free coordinate networks [30] that employ a Gaussian activation function $\mathbf{x} \to \exp\left(\frac{|\mathbf{x}-\mu|^2}{2\sigma^2}\right)$. The hyperparameter μ denotes the mean and σ the standard deviation of the Gaussian, with a smaller σ leading to a higher frequency network.

3.2. Second-order optimizers

This section introduces three second-order optimizers discussed in the paper. For more information and pseudocode, see Sec. 1 of supp. material.

Newtons method utilizes a quadratic approximation of an objective function f and uses the inverse Hessian matrix to take a gradient step. The update is computed as:

$$\theta_{t+1} = \theta_t - H(\theta_t)^{-1}\nabla f(\theta_t), \tag{3}$$

where $H(\theta_t)$ denotes the Hessian of f at θ_t. Thus we see that the optimizer utilizes curvature information to take updates as the Hessian is a measure of the curvature of the objective function. Convergence to a global minimum is guaranteed for convex functions [24], but the algorithm may not converge for non-convex functions. When the algorithm converges to a minimum, it does so at a *quadratic rate*, which is significantly faster than first-order optimizers such as gradient descent/Adam, which converge at sublinear/linear rate [24]. However, inverting the Hessian has a computational complexity of $\mathcal{O}(p^3)$ for an objective function with p parameters [24], making Newton's method memory-intensive for high parameter objective functions such as overparameterised neural networks.

The BFGS algorithm is a quasi-Newton method that approximates the inverse Hessian matrix with a positive definite matrix M_t iteratively to avoid computing the exact Hessian matrix in Newton's method.

Given a choice of initialisation, M_0, M_{t+1} can be computed using the closed form BFGS update

$$M_{t+1} = \left(I - \frac{y_t s_t^T}{y_t^T s_t}\right)^T M_t \left(I - \frac{y_t s_t^T}{y_t^T s_t}\right) + \frac{s_t s_t^T}{y_t^T s_t}, \tag{4}$$

where for an objective function f

$$y_t = \nabla f(\theta_{t+1}) - \nabla f(\theta_t) \text{ and } s_t = \theta_{t+1} - \theta_t. \tag{5}$$

The parameter update at iteration t is then given by

$$\theta_{t+1} = \theta_t - M_t \nabla f(\theta_t). \tag{6}$$

The fundamental principle of quasi-Newton methods is to avoid computing the inverse Hessian from scratch every iteration. Instead, the BFGS algorithm approximates the inverse Hessian H^{-1} with a positive definite matrix M_{t+1}, via (4), using recent curvature information, via (5), and an existing approximation M_t. This reduces the computational complexity to $\mathcal{O}(p^2)$ for an objective function with p-parameters. The BFGS algorithm converges at a superlinear rate, slower than Newton's method but faster than Gradient descent/Adam, for a twice differentiable objective function with Lipshitz continuous Hessian [24].

The L-BFGS algorithm is a limited memory variant of the BFGS algorithm. Instead of storing the approximate inverse Hessian M_t at each iteration, the algorithm stores a limited number of the vector pairs $\{s_t, y_t\}$, see (5), used in the construction of the approximate Hessian M_t, see (4). This reduces the computational complexity to $\mathcal{O}(p)$ [24]. Its convergence rate is superlinear [24].

4. Theoretical Analysis

4.1. Analyzing the Hessian of a Coordinate Network

This section gives a theoretical analysis of the poor gradient and curvature conditioning of the MSE loss landscape of a ReLU-activated coordinate network. In contrast, the well-conditioned gradient and curvature of the MSE loss landscape of a sine-/Gaussian-coordinte network is highlighted. The predictions made from the theory are then empirically verified. See supp. material sec. 2 for proofs of the theory.

As the weights of a neural network in (1) are trainable we can represent it as a map $f : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}^{n_L}$, where p denotes the parameter dimension and is given by $p = n_0 \times n_1 + n_1 \times n_2 + \ldots + n_{L-1} \times n_L$. Letting $p = (\theta_1, \ldots, \theta_L)$ with $\theta_i \in \mathbb{R}^{n_i \times n_{i-1}}$, we write the map as

$$f(\theta, X) = f_L(\theta_L) \circ \cdots \circ f_1, (\theta_1)(X), \tag{7}$$

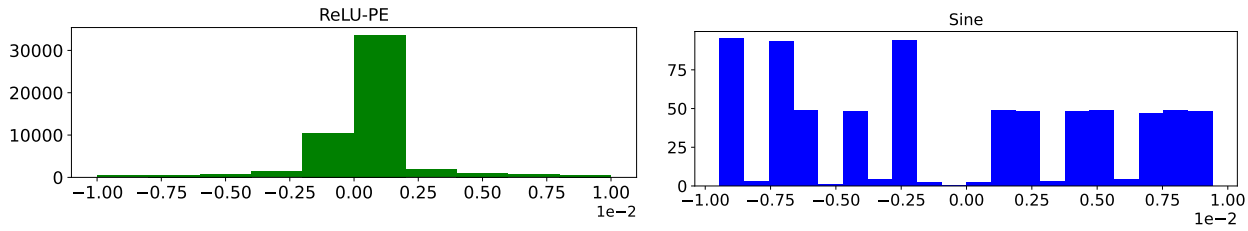


Figure 2: Total eigenvalue distribution of the Hessian of MSE loss for sine- and ReLU-PE-activated networks throughout training. ReLU-PE has 28% of its eigenvalues at 0, while the smallest eigenvalue for the sine-activated network is 5×10^{-4}. This highlights the superior conditioning of the Hessian of a sine-activated network (**no** zero eigenvalues) compared to a ReLU one (**many** zero eigenvalues).

where $f_i(\theta_i) : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ is defined by

$$f_i(\theta_i)(v) = \psi(\theta_i \cdot v). \tag{8}$$

Each of the maps $f_i(\theta_i)$ can be expanded as a map

$$f_i : \mathbb{R}^{n_i \times n_{i-1}} \times \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i} \tag{9}$$

and thus a neural network can equally well be described via a collection of maps $\{f_i : \mathbb{R}^{n_i \times n_{i-1}} \times \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}\}_{i=1}^{n_L}$ satisfying the composition structure (1).

Given an input data set X, we let

$$F_k = f_k(\theta_k) \circ \cdots \circ f_1(\theta_1)(X) \tag{10}$$

denote the k-layer neural output function.

For a fixed set of training data (X, Y), with $X \in \mathbb{R}^d$ the inputs and $Y \in \mathbb{R}^{n_L}$ the targets, the MSE loss function, see (2), is a map $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$. To simplify the statement of the following lemma, we introduce the following notation. Let $\Delta(\psi'(\theta_{L-j}F_{L-j-1})$ denote the diagonal matrix with entries given by $\psi'(\theta_{L-j}F_{L-j-1})$, where ψ' denotes the derivative of the activation function ψ, flattened column wise as a vector and let

$$\mathcal{D}_{L-l-1} = \prod_{j=1}^{L-l-1} (\theta_{L-j}^T \otimes Id)\Delta(\psi'(\theta_{L-j}F_{L-j-1})), \tag{11}$$

where $\otimes$ denotes the Kronecker product of matrices.

The following lemma computes the gradient of the MSE loss (2).

Lemma 4.1. *Let f be a neural network and (X, Y) a training data set, with X inputs and Y targets, defined by the family of maps $\{f_i : \mathbb{R}^{n_i \times n_{i-1}} \times \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}\}_{i=1}^{L}$. Given $\theta_l \in \mathbb{R}^{n_l \times n_{l-1}}$, we have*

$$\nabla_{\theta_l}\mathcal{L} = \big(Id \otimes F_{l+1}\big)\mathcal{D}_{L-l-1}\big(\theta_L^T \otimes Id\big)\big(F_L - y\big), \tag{12}$$

where $\mathcal{L}$ denotes the MSE loss function associated to the network.

The Hessian of the MSE loss of a neural network can be computed using lemma 4.1, the product rule, and the chain rule. For each point $\theta \in \mathbb{R}^p$, the Hessian $Hess(\mathcal{L}(\theta))$ will be a $(n_L \times p) \times p$-matrix. Thus one can see that if p is large, the Hessian will be an extremely large matrix even in the case that $n_L = 1$. Even though the Hessian is an enormous matrix, one can still obtain insight into its structure via (12). Given a parameter point $\theta_k \in \mathbb{R}^{n_k \times n_{k-1}}$, we observe that the second derivative $\nabla_{\theta_k}\nabla_{\theta_l}\mathcal{L}$ will have three main terms given by applying the product rule:

1. $\big(\nabla_{\theta_k}\big(Id \otimes F_{l+1}\big)\big)\mathcal{D}_{L-l-1}\big(\theta_L^T \otimes Id\big)\big(F_L - y\big)$
2. $\big(Id \otimes F_{l+1}\big)\big(\nabla_{\theta_k}\mathcal{D}_{L-l-1}\big)\big(\theta_L^T \otimes Id\big)\big(F_L - y\big)$
3. $\big(Id \otimes F_{l+1}\big)\mathcal{D}_{L-l-1}\big(\nabla_{\theta_k}\big(\big(\theta_L^T \otimes Id\big)\big(F_L - y\big)\big)\big)$.

Terms 1. and 3. will all contain first-order derivatives of the neural network function f. The second term will be the only term that will contain second-order derivatives of the neural network function, see Sec. 2 of supp. material for details. As the derivative of a ReLU activation is a step function, and its second derivative is a Dirac delta distribution, see Sec. 2 of supp. material for a proof, this analysis shows that in the case of a ReLU-activated network (with or without positional embedding), the Hessian of the loss function is discontinuous and hence poorly conditioned.

Proposition 4.2. *Let f be a ReLU-network, with or without positional embedding. Then the hessian of the loss $\mathcal{L}$ contains two types of poorly conditioned terms:*

1. *sums of step functions*
2. *sums of Dirac delta distributions.*

Prop. 4.2 implies the Hessian of the loss function in a ReLU-network, with or without positional embedding, is likely to be rank deficient due to the high probability of encountering many zeros in the Hessian matrix arising from the step function and delta function terms in its expansion. In contrast, the derivatives of sine and Gaussian

functions exhibit smoother behavior and are less prone to rank-deficiency in their Hessians. Additionally, Prop. 4.2 suggests that the curvature of a ReLU-MLP is poorly conditioned compared to one activated by sine/Gaussian. Hence, second-order optimizers that take curvature into account are expected to perform better on sine- or Gaussian-activated coordinate networks as opposed to those activated by ReLU.

We verified our theoretical predictions on an image reconstruction task by training two networks: one with a sine activation and another with a ReLU-PE [21]. Both networks were trained for 50 iterations on a 50×50 image with full sampling and L-BFGS optimizer. We computed the eigenvalues of the Hessian of the MSE loss at each iteration throughout training. Fig. 2 shows the distribution of eigenvalues in the interval $[-0.01, 0.01]$. As predicted by our theory, the sine-activated network has no zero eigenvalues, whereas the ReLU-PE-network has many. For a more comprehensive analysis, including ReLU and Gaussian MLPs with varying width, depth, and initialization schemes, refer to Sec. 3 of supp. material.

4.2. Analyzing L-BFGS on a Coordinate Network

In this section, we provide a theoretical and empirical analysis of the L-BFGS algorithm [24] on coordinate networks activated by ReLU and sine/Gaussian. We theoretically show that for a ReLU/ReLU-PE-activated coordinate network the L-BFGS algorithm is not guaranteed to converge superlinearly, however for a sine- or Gaussian-activated network superlinear convergence is guaranteed. We then verify these theoretical predictions empirically. See supp. material sec. 2 for proofs of the theorems.

The following theorem provides conditions under which the L-BFGS algorithm converges superlinearly. Its proof can be found in [24].

Theorem 4.3. *Let $f(\theta)$ be a twice continuously differentiable objective function. Suppose the iterates θ_t of the L-BFGS algorithm (see Sec. 3.2) converge to a minimiser θ^* of f. Furthermore, assume that the Hessian H of f is Lipshitz continuous locally around θ^*. Then the iterates θ_t converge superlinearly to θ^*.*

Theorem 4.3 shows that in order to guarantee that the L-BFGS algorithm converges superlinearly to a minimum, two conditions must be checked:

1. The objective function f must be twice continuously differentiable.
2. The Hessian H of the objective function must be Lipshitz continuous locally about the minimum point.

We show that ReLU/ReLU-PE-activated coordinate networks can fail to satisfy both conditions. We will first define the notion of a continuously differentiable minimum of a general continuous objective function.

Definition 1. *Let f be a continuous objective function and θ^* a (possibly local) minimum of f. We say θ^* is a continuously differentiable (local) minimum of f if f is differentiable at θ^* and the derivative is continuous at θ^*. Otherwise θ^* is called a non-continuously differentiable (local) minimum.*

Example 1. *$ReLU(x) = \max(x, 0)$, is an example of a function that contains both non-continuously differentiable and continuously differentiable minima. The point 0 is a non-continuously differentiable minimum. This is because the derivative of ReLU is given by the function $\mathcal{H}(x) = 0$ for $x \leq 0$ and $\mathcal{H}(x) = 1$ for $x > 0$. This function is clearly not continuous at 0. However, all negative numbers are continuously differentiable minima.*

Example 2. *The function $f(x) = |x|$ is an example of a function with only non-continuously differentiable minima, given by $x = 0$.*

Example 3. *A sine function has only continuously differentiable minima.*

Example 4. *The MSE loss function of a ReLU/ReLU-PE-coordinate network will have non-continuously differentiable minima [23]. In contrast, by the chain rule the MSE loss function of a sine/Gaussian-coordinate network can only have continuously differentiable minima.*

Ex. 4 highlights a key difference between the loss landscape of a ReLU/ReLU-PE-activated network and a sine-/Gaussian-activated one, trained with MSE loss. Namely, that the former can have non-continuously differentiable minima making the Hessian about such a minimum discontinuous, while the latter will always have well behaved continuously differentiable minima. As the following theorems show, this can affect the rate of convergence of a second-order optimizer on the MSE loss of such networks.

Theorem 4.4. *Let f be a ReLU/ReLU-PE-activated coordinate network. Let $\mathcal{L}(\theta)$ denote the MSE loss associated to f and a training set (X, Y), see Sec. 3.1.*

1. *Then $\mathcal{L}$ is not twice continuously differentiable at every parameter point θ.*
2. *If the L-BFGS algorithm applied to $\mathcal{L}(\theta)$ converges to a (local) minimum θ^* such that θ^* is a non-continuously differentiable (local) minimum of $\mathcal{L}$. Then the convergence is not guaranteed to be superlinear.*

Theorem 4.5. *Let f be an sine- or Gaussian-activated coordinate network. Let $\mathcal{L}(\theta)$ denote the MSE loss associated to f and a training set (X, Y), see Sec. 3.1.*

1. *Then $\mathcal{L}$ is twice continuously differentiable at every parameter point θ.*

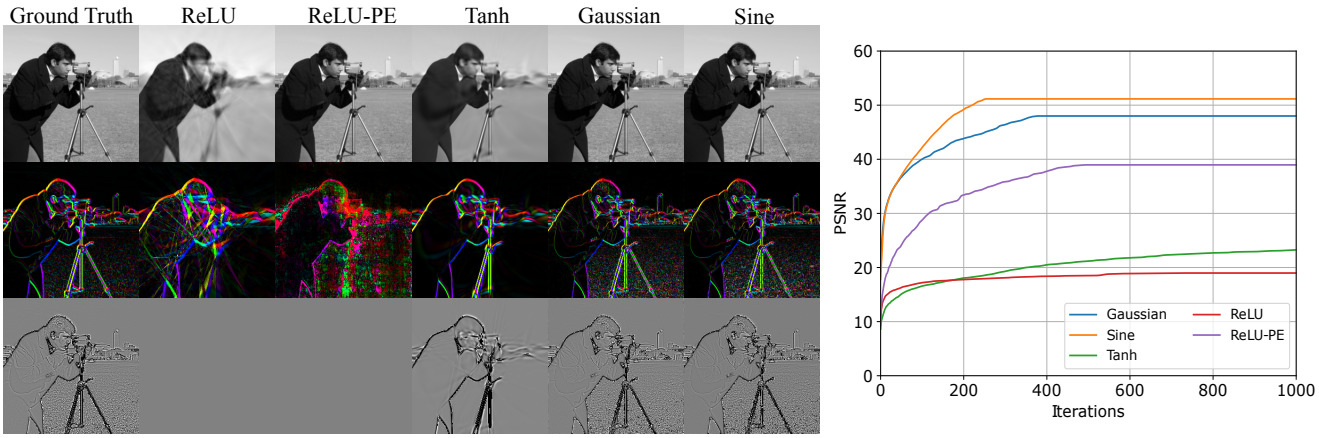


Figure 3: **2D Image Reconstruction.** *Left*: Comparison of various coordinate-MLPs f in fitting the *Cameraman* image (*top left*) using the L-BFGS optimizer. Note that all networks were only trained on the target image. We also show gradient (*second row*) and Laplacian (*third row*) of neural output function. *Right*: Training convergence of each network.

2. *If the L-BFGS algorithm applied to $\mathcal{L}(\theta)$ converges to a (local) minimum θ^* then the convergence is super-linear.*

Thms. 4.4 and 4.5 show that the MSE loss of a sine- or Gaussian-coordinate network has continuous curvature across parameter space, while ReLU-activated networks do not. This makes second-order optimizers effective in accelerating the training of sine-/Gaussian-activated networks, compared to first-order optimizers such as SGD or Adam, which generally have sub-linear/linear rates of convergence [11, 15].

Fig. 1 shows the convergence of a sine and ReLU-activated coordinate network trained with both Adam and L-BFGS on an image reconstruction task. The sine-network trained with L-BFGS has a much faster convergence rate than the sine-activated network trained with Adam, as predicted by Thm. 4.5. However, the ReLU-network trained with Adam converges at a faster rate than L-BFGS, see Thm 4.4.

5. Experiments

In this section, we demonstrate the effectiveness of L-BFGS on various popular tasks: 2D image reconstruction and novel view synthesis using neural radiance fields (NeRF); see Sec. 4 of supp. material for additional results for other modalities such as audio, shape and video reconstruction.

5.1. Images

Given pixel coordinates $\mathbf{x} \in \mathbb{R}^2$, we aim to optimize the network f to regress the associated RGB values $\mathbf{c} \in \mathbb{R}^3$ [35, 30]. In this task, we will first explain why non-traditional activations are well-suited for training with L-BFGS, followed by their comparisons with that of competitive first-order optimizers, e.g. Adam.

Gradient Perspective. In Fig. 3, we compare the performance of various network architectures optimized with L-BFGS on the Cameraman image using a 4-hidden layer MLP with 64 hidden units. As predicted by Prop. 4.2 and Thm. 4.4, ReLU and ReLU-PE activations produce *extremely bad* gradients and Laplacian, while Tanh lacks fine details. In contrast, sine and Gaussian activations produce high-quality reconstructions with *well-behaved derivatives*. Furthermore, non-traditional activations converge *significantly faster* than traditional ones, indicating that traditional activations do not train well with L-BFGS, as predicted by Thm. 4.5.

L-BFGS vs Adam. Fig. 4 shows a reconstruction snapshot of the *pepper* image, trained with both L-BFGS and

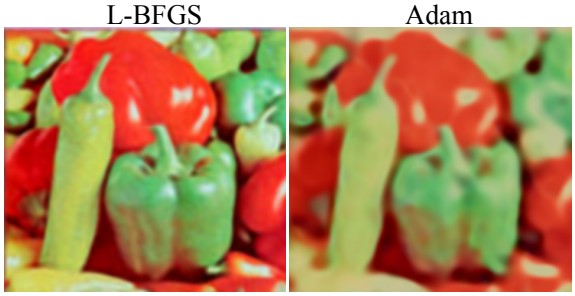


Figure 4: **2D Image Reconstruction.** L-BFGS has achieved a substantially better reconstruction than Adam given the same amount of training time.

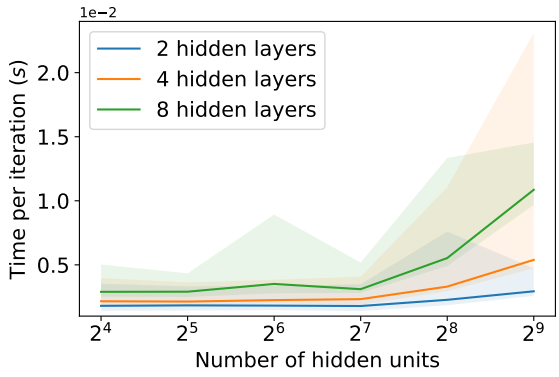


Figure 5: As the size of the network parameters grow, the time per iteration for the L-BFGS optimizer increases due to the added computational complexity. Note that *solid* line denotes mean while *transparency region* denotes variance.

Adam, using a 4-hidden layer, 64 width sine-activated network. Impressively, despite L-BFGS only being trained for $1s$, the reconstruction is remarkably sharp (35.76dB), compared to Adam which achieved a PSNR of 22.93. Overall, L-BFGS achieves convergence $5\times$ faster than Adam.

Computational Bottleneck. We found that when training with a large size neural network, L-BFGS did not offer any significant advantage over Adam. Fig. 5 shows that the computational time of L-BFGS increases when the network's parameter size grows, due to its computational complexity for computations using past curvature vectors, see Sec. 3.2. To mitigate this issue, we propose a patch-based decomposition strategy in Sec. 5.2.

5.2. KiloImage

We introduce KiloImage, a patch-based decomposition strategy for optimizing a gigapixel image using L-BFGS. We uniformly decompose the image into K grids of equal dimension, each represented by a small sine-MLP with 4 hidden layers and 64 neurons. We use frequency 30. Once all the MLPs are optimized, we combine all the resulting outputs to form a global reconstruction. Fig. 6 shows an example of reconstructing a image of resolution 2000×1000 using the patch-based decomposition technique. We used $K = 200$. As depicted in Fig. 6, L-BFGS outperforms Adam, achieving an average ~ 14 times faster convergence, resulting in high-quality reconstructions after just 0.14 seconds of training. We refer the readers to Sec. 4 of the supp. material for additional results on other gigapixel instances.

Patched-MLP trained with L-BFGS vs. non-patched MLP trained with Adam We use a 5-layer sine-MLP with 930 neurons (2603073 parameters) and compared it to a patched-MLP (2573400 parameters). Due to hardware constraint, we used stochastic sampling with the largest minibatch-size option (1 million points). We observe that patched L-BFGS achieves convergence $46\times$ faster (33.80dB) than the non-patched MLP trained with Adam (30.24dB).

5.3. Neural Radiance Fields (NeRF)

NeRF has recently emerged as a compelling strategy for utilizing a MLP to model 3D objects and scenes using multi-view 2D images. This approach shows promise for generating high-fidelity reconstruction in novel view synthesis task [21, 32, 8, 17]. Given 3D points $\mathbf{x} \in \mathbb{R}^3$ and viewing direction, NeRF aims to estimate the radiance field of a 3D scene which maps each input 3D coordinate to its corresponding volume density $\sigma \in \mathbb{R}$ and directional emitted color $\mathbf{c} \in \mathbb{R}^3$ [21, 17, 8]. In this section, we compare L-BFGS against the Adam optimizer on a popular application of a coordinate network in novel view synthesis task, NeRF [21]. For simplicity, we used a minimalist version of a NeRF model that excluded view-dependence and hierarchical ray sampling. We trained a tiny Gaussian-activated MLP with 4 hidden layers and 128 neurons on the real world LLFF forward-facing scenes [21], that were downscaled by a factor of 5. Fig. 7 presents the qualitative result obtained for the *fern* instance. Impressively, with only 130 seconds, TinyNeRF trained with L-BFGS generated a detailed reconstruction in only 400 iterations whereas TinyNeRF trained with Adam produced blurry renderings for 1300 iterations, indicating the superiority of L-BFGS for this instance.

KiloNeRF. As discussed in Sec. 5.1, while L-BFGS can achieve faster convergence compared to Adam, this advantage diminishes as the number of parameters of the neural network increases. This presents a particular challenge when training NeRF, a 5D high-dimensional problem that typically requires a larger network, such as an 8-layer 256 network. In this section, we showcase how we can mitigate the computational bottleneck associated with training a large-scale NeRF with L-BFGS. Building on the recent innovation of KiloNeRF [32], we trained thousands of 2-layer 32 width sine-activated KiloNeRF with L-BFGS and compared its performance to one trained with Adam. Note that we sampled each MLP with 10k points. As presented in Table 1, L-BFGS trained the KiloNeRF $\sim 6\times$ faster than Adam and produced higher reconstruction quality along the way. Fig. 8 compares a qualitative result of a KiloNeRF trained with L-BFGS and Adam – after 600 seconds, the KiloNeRF trained with L-BFGS is already able to produce good-quality reconstructions.

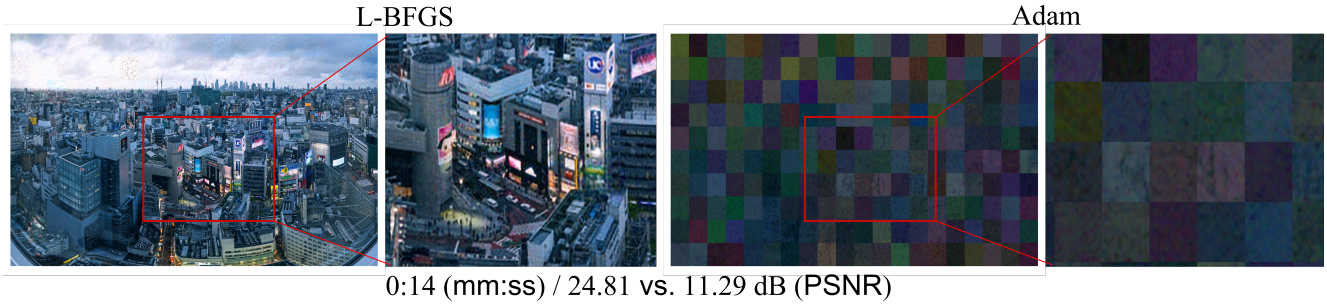


Figure 6: **Gigapixel Image Reconstruction** (1000×2000 **resolution).** Our approach represents a gigapixel image using 200 sine-activated tiny-MLPs. We report a comparison in terms of optimization time and PSNR (L-BFGS vs. Adam). Using L-BFGS, our method achieves a substantially higher-fidelity reconstruction (**24.81** dB) than Adam (11.29 dB) given the same amount of training time. L-BFGS achieves convergence $\sim 14\times$ faster than Adam (33.8 vs. 33.22 dB).

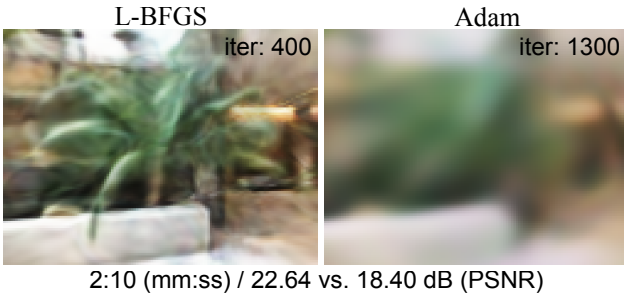


Figure 7: **Novel View Synthesis with NeRF** for a *fern* instance from the LLFF dataset [20]. We report a comparison in terms of optimization time and PSNR (L-BFGS vs. Adam). Using L-BFGS, Tiny-NeRF achieves a good reconstruction (**22.64**dB) compared to Adam (18.40dB) after training for the same amount of time. Overall, L-BFGS achieves convergence (24dB) $\sim 2\times$ faster than Adam.

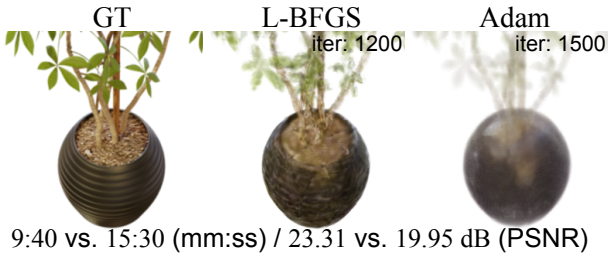


Figure 8: **Novel View Synthesis with KiloNeRF** for a *ficus* test instance from the LLFF dataset [20]. We report a comparison in terms of optimization time and PSNR (L-BFGS vs. Adam). Using L-BFGS, KiloNeRF achieves a good reconstruction (**23.31**dB) compared to Adam (19.95dB).

5.4. Analyzing the Computational Cost

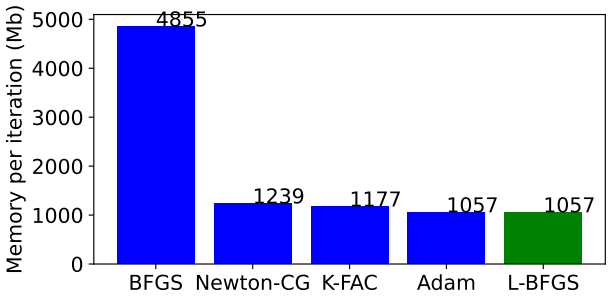
Fig. 9 compares different second-order optimizers in terms of memory usage and optimization time per iteration, see Sec. 1 of supp. material for details on these optimizers. Interestingly, we found that K-FAC [19] struggled to optimize a sine-activated coordinate network. This led us to speculate that the initialization scheme proposed in [35] may not be optimal for K-FAC optimization and warrants further investigation. The results in Fig. 9 were obtained using a 4-layer 64 width Gaussian-activated coordinate-MLP. Our findings indicate that L-BFGS strikes a good balance between fast training and comparable memory usage compared to Adam, offering a "best of both worlds" solution. While Adam has the lowest time and memory per iteration, it is important to note that second-order optimizers aim to converge with fewer iterations. This is demonstrated in the experiments discussed in Sec. 5.

Method	PSNR↑	SSIM↑	LPIPS↓	Num Iters↓	Training Time (s)↓
Adam	20.78	0.85	0.18	6000	5324.48
L-BFGS	**22.01**	**0.86**	**0.15**	**1200**	**889.97**

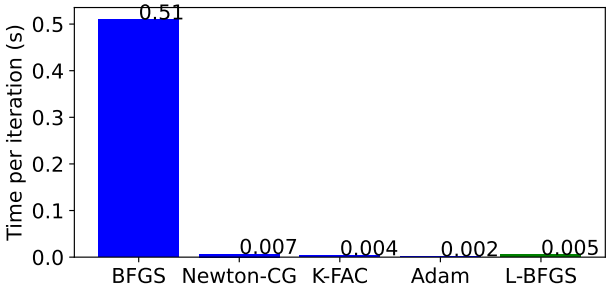
Table 1: Quantitative results of KiloNeRF on all instances from the Blender dataset [21]. On average, L-BFGS trained $6\times$ faster than Adam and was able to achieve competitive quality scores with significantly less iterations.

6. Conclusion

In this paper, we examine the application of second-order optimizers in training coordinate-MLPs. We show that coordinate networks using non-traditional activations, such as sine or Gaussian functions, exhibit better Hessians/curvature conditioning than those using ReLU activations. We validated our theory using an L-BFGS optimizer across various signal reconstruction tasks. Our results demonstrate that using second-order optimizers on small-scale data projects significantly reduces training times compared to using Adam. However, in large-scale applications



(a) Memory per iteration



(b) Training time per iteration

Figure 9: Comparison of computational complexity of various well-known second-order optimizers. L-BFGS strikes a good balance between fast training and comparable memory usage compared to Adam.

with a large number of parameters, the computational complexity of a second-order optimizer hinders its use. To mitigate this challenge, we proposed a patch-based decomposition strategy, breaking down large datasets into smaller patches. By training a second-order optimizer on each patch, we offer a viable approach for efficiently training coordinate networks with second-order optimizers in large-scale scenarios.

7. Limitations

We trained coordinate-MLPs using L-BFGS with full samples. While stochastic L-BFGS exists (e.g., [43, 28, 22]), we found that such stochastic L-BFGS algorithms perform poorly for training coordinate-MLPs compared to Adam with stochastic sampling. This presents a challenge for practitioners who want to use stochastic second-order optimizers in training coordinate-MLPs.

Acknowledgements

This research was funded (partially) by the Australian Government through the Australian Research Council.

References

[1] Naman Agarwal, Brian Bullins, Xinyi Chen, Elad Hazan, Karan Singh, Cyril Zhang, and Yi Zhang. Efficient full-matrix adaptive regularization. In *International Conference on Machine Learning*, pages 102–110. PMLR, 2019. 2

[2] Paul T Boggs and Richard H Byrd. Adaptive, limited-memory bfgs algorithms for unconstrained optimization. *SIAM Journal on Optimization*, 29(2):1282–1299, 2019. 2

[3] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, pages 333–350. Springer, 2022. 1

[4] Anpei Chen, Zexiang Xu, Xinyue Wei, Siyu Tang, Hao Su, and Andreas Geiger. Factor fields: A unified framework for neural fields and beyond. *arXiv preprint arXiv:2302.01226*, 2023. 2

[5] Boyuan Chen, Robert Kwiatkowski, Carl Vondrick, and Hod Lipson. Fully body visual self-modeling of robot morphologies. *Science Robotics*, 7(68):eabn1944, 2022. 1

[6] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8628–8638, 2021. 1

[7] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 2

[8] Shin-Fang Chng, Sameera Ramasinghe, Jamie Sherrah, and Simon Lucey. Gaussian activated neural radiance fields for high fidelity reconstruction and pose estimation. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII*, pages 264–280. Springer, 2022. 1, 2, 7

[9] Boyang Deng, J. P. Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Nasa neural articulated shape approximation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 612–628, Cham, 2020. Springer International Publishing. 2

[10] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 1

[11] Guillaume Garrigos and Robert M Gower. Handbook of convergence theorems for (stochastic) gradient methods. *arXiv preprint arXiv:2301.11235*, 2023. 6

[12] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 2

[13] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018. 2

[14] Changcun Huang. Relu networks are universal approximators via piecewise linear or constant functions. *Neural Computation*, 32(11):2249–2278, 2020. 2

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[16] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022. 1

[17] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5741–5751, 2021. 1, 7

[18] Bo Liu and Yi Liang. Optimal function approximation with relu neural networks. *Neurocomputing*, 435:216–227, 2021. 2

[19] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015. 2, 8

[20] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 8

[21] 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. *Communications of the ACM*, 65(1):99–106, 2021. 1, 2, 5, 7, 8

[22] Philipp Moritz, Robert Nishihara, and Michael Jordan. A linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, pages 249–258. PMLR, 2016. 2, 9

[23] Rotem Mulayoff, Tomer Michaeli, and Daniel Soudry. The implicit bias of minima stability: A view from function space. *Advances in Neural Information Processing Systems*, 34:17749–17761, 2021. 2, 5

[24] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999. 1, 2, 3, 5

[25] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2

[26] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021. 2

[27] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021. 2

[28] Peng Qi, Wei Zhou, and Jizhong Han. A method for stochastic l-bfgs optimization. In *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 156–160. IEEE, 2017. 9

[29] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019. 1, 3

[30] S. Ramasinghe and S. Lucey. Beyond Periodicity: Towards a Unifying Framework for Activations in Coordinate-MLPs. In *ECCV*, 2022. 1, 2, 3, 6

[31] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14153–14161, June 2021. 2

[32] C. Reiser, S. Peng, Y. Liao, and A. Geiger. KiloNeRF: Speeding Up Neural Radiance Fields With Thousands of Tiny MLPs. In *ICCV*, 2021. 2, 7

[33] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 2

[34] Vishwanath Saragadam, Jasper Tan, Guha Balakrishnan, Richard G Baraniuk, and Ashok Veeraraghavan. Miner: Multiscale implicit neural representation. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIII*, pages 318–333. Springer, 2022. 2

[35] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, G., and Wetzstein. Implicit Neural Representations with Periodic Activation Functions. In *NIPS*, 2020. 1, 2, 3, 6, 8

[36] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems*, 32, 2019. 1

[37] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10753–10764, 2021. 1

[38] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 1

[39] Yu Sun, Jiaming Liu, Mingyang Xie, Brendt Wohlberg, and Ulugbek S Kamilov. Coil: Coordinate-based internal learning for imaging inverse problems. *arXiv preprint arXiv:2102.05181*, 2021. 1, 2

[40] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020. 1, 3

[41] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021. 2

[42] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4578–4587, June 2021. 2

[43] Renbo Zhao, William Benjamin Haskell, and Vincent YF Tan. Stochastic l-bfgs: Improved convergence rates and practical acceleration strategies. *IEEE Transactions on Signal Processing*, 66(5):1155–1169, 2017. 2, 9

[44] Jianqiao Zheng, Sameera Ramasinghe, Xueqian Li, and Simon Lucey. Trading positional complexity vs deepness in coordinate networks. In *Computer Vision – ECCV 2022*, pages 144–160, Cham, 2022. Springer Nature Switzerland. 1, 2, 3

[45] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12786–12796, 2022. 2