Reverse Engineering the Neural Tangent Kernel

James B. Simon [1 2]   **Sajant Anand** [1]   **Michael R. DeWeese** [1 2]

Abstract

The development of methods to guide the design of neural networks is an important open challenge for deep learning theory. As a paradigm for principled neural architecture design, we propose the translation of high-performing *kernels*, which are better-understood and amenable to first-principles design, into equivalent *network architectures*, which have superior efficiency, flexibility, and feature learning. To this end, we constructively prove that, with just an appropriate choice of activation function, any positive-semidefinite dot-product kernel can be realized as either the NNGP or neural tangent kernel of a fully-connected neural network *with only one hidden layer*. We verify our construction numerically and demonstrate its utility as a design tool for finite fully-connected networks in several experiments.

1. Introduction

The field of deep learning theory has recently been transformed by the discovery that, as network widths approach infinity, models take simple analytical forms amenable to theoretical analysis. These limiting forms are described by kernel functions, the most important of which is the *neural tangent kernel* (NTK), which describes the training of an infinite-width model via gradient descent (Jacot et al., 2018; Lee et al., 2019). Limiting kernels are now known for fully-connected networks (Daniely et al., 2016; Lee et al., 2018; Matthews et al., 2018; Jacot et al., 2018; Lee et al., 2019), convolutional networks (Novak et al., 2019a; Arora et al.,

---

[1]Department of Physics, University of California, Berkeley, Berkeley, CA 94720 [2]Redwood Center for Theoretical Neuroscience and Helen Wills Neuroscience Institute, University of California, Berkeley, Berkeley, CA 94720. Correspondence to: James B. Simon <james.simon@berkeley.edu>.

Code to reproduce experiments is available at https://github.com/james-simon/reverse-engineering.

2019b), transformers (Hron et al., 2020), and more (Yang, 2019), giving unprecedented insight into commonly-used network architectures.

However, deep learning theory must ultimately aim not merely to understand existing architectures but to *design better architectures*. Neural network design is famously unscientific, proceeding via an artful combination of intuition, trial-and-error, and brute-force search that yields little understanding of why certain architectures perform well or how we might do better. We argue that theorists should endeavor to use the many insights gained from the study of wide networks to develop well-understood methods and principles to illuminate this process.

We propose that one potential avenue for principled architecture design is *reverse engineering the network-kernel correspondence*: first design a kernel well-suited to the task at hand, then choose network hyperparameters to (approximately) realize that kernel as a finite network. This paradigm promises the best of both model types: kernels are analytically simple and thus far more amenable to first-principles design, while neural networks have the advantages of cheaper training/inference, flexibility in training schemes and regularization, and the ability to learn useful features and transfer to new tasks.

Despite clear implications for architecture design as well as fundamental interest, very little work has studied the problem of achieving a desired kernel with a network architecture. *Here we fully solve this problem for fully-connected networks (FCNs) on normalized data:* we specify the full set of achievable FCN kernels and provide a construction to realize any achievable NNGP kernel or NTK with a shallow (i.e., single-hidden-layer) FCN. We then confirm our results experimentally and describe two cases in which, by reverse-engineering a desired NTK, we can design high-performing FCNs in a principled way.

Our main contributions are as follows:

- We constructively prove that, with just an appropriate choice of activation function, any positive-semidefinite dot-product kernel can be realized as the NNGP kernel or NTK of an FCN with no biases and just one hidden layer.

- As a surprising corollary, we prove that shallow FCNs can realize a strictly *larger* set of dot-product kernels than FCNs with multiple nonlinear layers can.

- We experimentally verify our construction and demonstrate our reverse engineering paradigm for the design of FCNs in two experiments: (a) we engineer a single-hidden-layer, finite-width FCN that mimics the training and generalization behavior of a deep ReLU FCN over a range of network widths, and (b) we design an FCN that significantly outperforms ReLU FCNs on a synthetic parity problem.

1.1. Related Work

The connection between kernels and infinitely wide networks was first revealed by Neal (1996), who showed that randomly-initialized single-hidden-layer FCNs approach Gaussian processes with deterministic kernels as their width increases. Much later, this work was extended to randomly-initialized deep FCNs (Lee et al., 2018; Matthews et al., 2018), and the kernels were given the name "neural network Gaussian process" (NNGP) or "conjugate" kernels. The related NTK, describing the training of deep FCNs, was discovered soon thereafter (Jacot et al., 2018). These findings and their extensions to other architectures have enabled many insights into neural network training (Zou et al., 2018; Du et al., 2019; Allen-Zhu et al., 2019) and generalization (Arora et al., 2019a; Bordelon et al., 2020; Simon et al., 2021). Many studies have analyzed the mapping from network architectures to corresponding kernels, but very few have considered the inverse mapping, which is the focus of this work.

Anticipating the NNGP correspondence, Daniely et al. (2016) considered the kernels represented by wide randomly-initialized networks of various architectures. These deep kernels are given by the composition of many layerwise kernels, and for FCNs, they derive an invertible mapping between activation function and layerwise kernel. Many of our results for the NNGP kernel follow from their work. We note that that study did not consider training or the NTK, did not discuss achieving a desired kernel as a design principle, and included no experiments.

Our work has close connections to random-feature models (Rahimi et al., 2007; Rahimi & Recht, 2008), which use random feature embeddings to approximate analytical kernels. Shallow networks with frozen first layers are random feature models approximating NNGP kernels (though here we chiefly consider the NTK and train all layers in our experiments). Unlike with neural networks, there is a substantial literature on engineering random features to yield desired kernels, with feature maps known for arbitrary dot-product kernels (Kar & Karnick, 2012; Pennington et al., 2015).

In a concurrent study in a spirit similar to ours, Shi et al. (2021) also explored NTK performance as a principle for activation function design. Like us, they decompose activation functions in a Hermite basis, but optimize the coefficients to directly minimize NTK test MSE in a black-box fashion, whereas in our comparable parity problem experiment, we transparently choose a suitable kernel matching the symmetry of the problem.

2. Theoretical Background

2.1. Preliminaries and Notation

We study fully connected architectures in this work. An FCN with L hidden layers of widths $\{n_\ell\}_{\ell=1}^L$ is defined by the recurrence relations

$$\mathbf{z}^{(\ell)} = W^{(\ell)}\mathbf{x}^{(\ell-1)} + \mathbf{b}^{(\ell)}, \qquad \mathbf{x}^{(\ell)} = \phi(\mathbf{z}^{(\ell)}), \quad (1)$$

starting with the input vector $\mathbf{x}^{(0)} \in \mathbb{R}^{d_{\text{in}}}$ and ending with the output vector $\mathbf{z}^{(L+1)} \in \mathbb{R}^{d_{\text{out}}}$. The weight matrices and bias vectors are defined and initialized as

$$W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, \quad W_{ij}^{(\ell)} = \frac{\sigma_w}{\sqrt{n_\ell}}\omega_{ij}^{(\ell)}, \quad \omega_{ij}^{(\ell)} \sim \mathcal{N}(0,1), \quad (2)$$

$$\mathbf{b}^{(\ell)} \in \mathbb{R}^{n_\ell}, \qquad \mathbf{b}_i^{(\ell)} = \sigma_b\beta_i^{(\ell)}, \qquad \beta_i^{(\ell)} \sim \mathcal{N}(0,1), \quad (3)$$

where σ_w and σ_b define the initialization scales of the weights and biases, $\omega_{ij}^{(\ell)}$ and $\beta_i^{(\ell)}$ are the *trainable parameters*, and $n_0 = d_{\text{in}}$ and $n_{L+1} = d_{\text{out}}$. It is the trainable parameters, not the weights and biases themselves, that will evolve according to gradient descent. This formulation, commonly called the "NTK parameterization," effectively decreases the weight updates by a factor of $\mathcal{O}(n_\ell^{-1})$ and allows us to take a sensible infinite-width limit of the network dynamics. The initialization scales σ_w, σ_b can in principle vary between layers, but we will generally take them to be layer-independent. We note that setting $\sigma_b = 0$ corresponds to a network with no biases.

Throughout our analysis, unless stated otherwise, we will make the following normalization assumption on the data:

Assumption 2.1. Each input $\mathbf{x}_i$ satisfies $|x_i| = \sqrt{n^0}$.

Though this assumption is rarely satisfied by natural data, enforcing it costs only one degree of freedom, and it can be enforced invertibly by appending one dummy index to each input before normalization so the number of degrees of freedom is preserved. This assumption will greatly simplify our theoretical analysis of FCN kernels. We will later explore the consequences of lifting this assumption empirically.

We will use $n!!$ to denote the double factorial, $\mathbb{S}^n$ to represent the n-sphere, and δ_{ij} to mean $\mathbb{1}_{i=j}$. We will write the first derivative of a function $f(z)$ as either $f'(z)$ or $\partial_z f(z)$.

We will also abbreviate the bivariate centered normal distribution with diagonal σ^2 and correlation coefficient c as

$$\mathcal{N}_c^{\sigma^2} \equiv \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma^2 & c\,\sigma^2 \\ c\,\sigma^2 & \sigma^2 \end{bmatrix}\right). \tag{4}$$

2.2. Neural Network Kernels

As a neural network's width approaches infinity, its behavior and properties become simple and tractable. In particular, its limiting behavior is described by two kernels in two different senses.[1]

The NNGP. A network with all its parameters randomly initialized will represent a random function, with the prior over parameters determining the prior over functions. In the infinite-width limit, this random function is sampled from a Gaussian process over the input space with mean zero and covariance given by an NNGP kernel $K^{(\mathrm{NNGP})}(\cdot, \cdot)$ (Matthews et al., 2018; Lee et al., 2018). Conditioning this functional prior on training data corresponds to kernel (ridge) regression with the NNGP kernel.

The NTK. When performing gradient flow on a training sample $\mathbf{x}_1$, the network output f on another sample $\mathbf{x}_2$ changes at a rate proportional to $K^{(\mathrm{NTK})}(\mathbf{x}_1, \mathbf{x}_2) \equiv \nabla_\theta f(\mathbf{x}_1) \cdot \nabla_\theta f(\mathbf{x}_2)$. Remarkably, knowledge of the model's NTK at all times is sufficient to describe training and generalization dynamics, with no knowledge of the model's internal structure required. Even more remarkably, for an infinite-width network, the NTK is both initialization-independent and fixed throughout training, and in many cases this limiting NTK can be computed in closed form (Jacot et al., 2018). An ensemble of such wide networks trained for infinite time is equivalent to kernel regression with the NTK (Jacot et al., 2018; Lee et al., 2019).

When referring to the kernels of FCN architectures, we will generally mean their deterministic kernels at infinite width. We will explicitly note when, in later experiments, we examine the empirical NTKs of finite networks.

Dot-product kernels. For FCNs, both the limiting NNGP and limiting NTK kernels are *rotation-invariant*: $K(\mathbf{x}_1, \mathbf{x}_2) = K(|\mathbf{x}_1|, |\mathbf{x}_2|, c)$, with $c \equiv \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{|\mathbf{x}_1||\mathbf{x}_2|} \in [-1, 1]$. Fixing $|\mathbf{x}_1|$ and $|\mathbf{x}_2|$ as per Assumption 2.1, only the third argument can vary, and the functional form reduces to merely $K(c)$. We call kernels of this form *dot-product kernels*. We note that an FCN's kernels are independent of d_{in}.

We will make use of the following classic result governing dot-product kernels. The nonnegativity constraints result from the requirement that the kernel be positive-

semidefinite.

Theorem 2.2 (Schoenberg (1942)). *Any dot product kernel over $\mathbb{S}^\infty \times \mathbb{S}^\infty$ must take the form $K(c) = \sum_{i=0}^\infty a_i c^i$, with $a_i \geq 0$ and $\sum_{i=0}^\infty a_i < \infty$.*

We will say that a polynomial $K : [-1, 1] \to \mathbb{R}$ is positive-semidefinite (PSD) if it obeys the constraints of Theorem 2.2. We note that the set of allowable dot-product kernels over $\mathbb{S}^{d_{\mathrm{in}}} \times \mathbb{S}^{d_{\mathrm{in}}}$ with finite d_{in} is somewhat larger, with the conditions on K becoming more restrictive and approaching those of Theorem 2.2 as $d_{\mathrm{in}} \to \infty$. However, since an FCN's kernels are independent of input dimension, they must obey Theorem 2.2, and we neglect kernels valid only at finite d_{in}.

Kernels of 1HL networks. Here we explicitly write the two kernels of single-hidden-layer (1HL) networks with activation function ϕ and initialization scales σ_w, σ_b (see, e.g., Appendix E of Lee et al. (2019)). We make use of the τ-transform, a common functional in the study of infinite-width networks, defined as

$$\tau_\phi(c; \sigma^2) \equiv \mathbb{E}_{z_1, z_2 \sim \mathcal{N}_c^{\sigma^2}}[\phi(z_1)\phi(z_2)]. \tag{5}$$

When the second argument to τ is 1, we will simply omit it and write $\tau(c)$. We note the following useful identity, which we prove in Appendix D:

$$\tau_{\phi'}(c; \sigma^2) = \frac{\partial_c}{\sigma^2}\tau_\phi(c; \sigma^2). \tag{6}$$

In terms of the τ-transform, the NNGP and NTK kernels of a 1HL network are

$$K^{(\mathrm{NNGP})}(c) = \sigma_w^2 \tau_\phi\left(\frac{\sigma_w^2 c + \sigma_b^2}{\sigma_w^2 + \sigma_b^2}; \sigma_w^2 + \sigma_b^2\right) + \sigma_b^2, \tag{7}$$

$$K^{(\mathrm{NTK})}(c) = K^{(\mathrm{NNGP})}(c) \tag{8}$$
$$+ (\sigma_w^2 c + \sigma_b^2)\tau_{\phi'}\left(\frac{\sigma_w^2 c + \sigma_b^2}{\sigma_w^2 + \sigma_b^2}; \sigma_w^2 + \sigma_b^2\right).$$

2.3. Hermite Polynomials

The *Hermite polynomials* $h_0, h_1, h_2, \ldots$ are an orthonormal basis of polynomials obtained by applying the Gram-Schmidt process to $1, z, z^2, \ldots$ with respect to the inner product $\langle f, g \rangle = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^\infty e^{-z^2/2} f(z)g(z)dz$. As the Hermite polynomials form a complete basis, any function ϕ that is square-integrable w.r.t. the Gaussian measure can be uniquely decomposed as $\phi(z) = \sum_{k=0}^\infty b_k h_k(z)$. We defer a more detailed introduction to the Hermite polynomials to Appendix C and here just mention two properties that will be of particular use (see e.g. O'Donnell (2014) and Wikipedia):

$$\mathbb{E}_{z_1, z_2 \sim \mathcal{N}_c^1}[h_k(z_1)h_\ell(z_2)] = c^k \delta_{k\ell} \quad \forall c \in [-1, 1], \tag{9}$$

$$h_k'(z) = \sqrt{k}\,h_{k-1}(z) \quad \forall k \geq 1. \tag{10}$$

---

[1] A *kernel* is essentially a bounded, symmetric, positive-semidefinite scalar function of two variables. We refer unfamiliar readers to Shawe-Taylor et al. (2004) for an introduction.

3. Theoretical Results

We now present our main theoretical results. We note that the results for NNGP kernels follow from Lemma 11 of Daniely et al. (2016); we include them for completeness, but the main new results are those for the NTK.

Theorem 3.1 (Kernel reverse engineering). *Any desired dot product kernel $K(c) = \sum_{k=0}^{\infty} a_k c^k$ can be achieved as*

- *the NNGP kernel of a single-hidden-layer FCN with $\sigma_w = 1, \sigma_b = 0$ and $\phi(z) = \sum_{k=0}^{\infty} \pm a_k^{1/2} h_k(z)$,*

- *the NTK of a single-hidden-layer FCN with $\sigma_w = 1, \sigma_b = 0$ and $\phi(z) = \sum_{k=0}^{\infty} \pm \left(\frac{a_k}{1+k}\right)^{1/2} h_k(z)$,*

where we use $\pm$ to indicate that the sign of each term is arbitrary. These are the complete set of activation functions that give this kernel in a single-hidden-layer FCN with $\sigma_w = 1, \sigma_b = 0$.

Corollary 3.2. *Let $\phi(z) \longleftrightarrow K(c)$ signify that K is the NNGP kernel or NTK of a single-hidden-layer FCN with activation function ϕ. If $\sigma_b = 0$ and $\phi(z) \longleftrightarrow K(c)$, then*

(a) $\phi'(z) \longleftrightarrow \sigma_w^{-2} K'(c)$.

(b) $\alpha\phi(z) \longleftrightarrow \alpha^2 K(c)$ for all $\alpha \in \mathbb{R}$.

(c) $\phi(-z) \longleftrightarrow K(c)$.

Corollary 3.3. *There are dot-product kernels that can be the NNGP kernel or NTK of a FCN only if it has exactly one nonlinear hidden layer.*

The proof of Theorem 3.1 is remarkably simple, and we provide it here. We defer the proofs of Corollaries 3.2 and 3.3 to Appendix D.

Proof of Theorem 3.1. First, we observe from Equations 6, 7 and 8 that a 1HL FCN with $\sigma_w = 1, \sigma_b = 0$ has kernels

$$K^{(\text{NNGP})}(c) = \tau_\phi(c), \tag{11}$$

$$K^{(\text{NTK})}(c) = \tau_\phi(c) + c\tau_{\phi'}(c) = (1 + c\partial_c)\tau_\phi(c). \tag{12}$$

Next we observe that, for an activation function $\phi(z) = \sum_{k=0}^{\infty} b_k h_k(z)$, Equation 9 lets us evaluate the τ-transforms, yielding

$$K^{(\text{NNGP})}(c) = \tau_\phi(c) = \sum_{k=0}^{\infty} b_k^2 c^k, \tag{13}$$

$$K^{(\text{NTK})}(c) = (1 + c\partial_c)\tau_\phi(c) = \sum_{k=0}^{\infty} b_k^2 (1 + k)c^k. \tag{14}$$

Equating these kernels with the desired $K(c) = \sum_{k=0}^{\infty} a_k c^k$ and solving for b_k completes the proof. $\square$

FCNs have maximal kernel expressivity. The traditional notion of *expressivity* denotes the set of functions achievable by varying a model's parameters. Our results can be understood in terms of a new notion of *kernel expressivity*, denoting the set of kernels achievable by varying an architecture's hyperparameters. Theorem 3.1 states that, for both NNGP and NTK kernels, FCNs have the greatest kernel expressivity we might have hoped for even with just one hidden layer: they can achieve all dot product kernels on $\mathbb{S}^\infty \times \mathbb{S}^\infty$ (i.e. those obeying the condition of Theorem 2.2).

Depth is inessential. We emphasize the surprising fact that, as all achievable dot-product kernels are achievable with a single hidden layer as per Theorem 3.1, *additional hidden layers do not increase kernel expressivity.* In fact, Corollary 3.3 states that additional nonlinear layers in fact *decrease* kernel expressivity.[2] These observations contradict the widespread belief that deeper networks are fundamentally more expressive and flexible (e.g. LeCun et al. (2015); Poggio et al. (2017); Poole et al. (2016); Schoenholz et al. (2017); Telgarsky (2016); Raghu et al. (2017); Mhaskar et al. (2017); Lin et al. (2017); Rolnick & Tegmark (2018)) and suggest that, at least for wide FCNs, one hidden layer is all you need.

Biases are unnecessary. It is also surprising that the full space of achievable dot product kernels can be achieved with only weights and no biases (i.e. $\sigma_b = 0$). This suggests that biases are unimportant when the activation function is chosen well and are merely neuroscience-inspired holdouts from the early days of deep learning.

NTKs for selected shallow networks. Prior works have computed $\tau_\phi(c)$ for several choices of ϕ; we reproduce these expressions in Appendix C. Applying Equation 12, we can then easily compute the 1HL NTKs for these choices of ϕ:

$$K^{(\text{NTK})}_{e^{az}} = e^{a^2(1+c)}(1 + a^2 c), \tag{15}$$

$$K^{(\text{NTK})}_{\sin(az)} = e^{-a^2}\left(\sinh(a^2 c) + a^2 c \cosh(a^2 c)\right), \tag{16}$$

$$K^{(\text{NTK})}_{\cos(az)} = e^{-a^2}\left(\cosh(a^2 c) + a^2 c \sinh(a^2 c)\right), \tag{17}$$

$$K^{(\text{NTK})}_{\text{ReLU}(z)} = \frac{1}{2\pi}\left(\left(1 - c^2\right)^{1/2} + 2c\left(\pi - \cos^{-1}(c)\right)\right). \tag{18}$$

We will revisit the sinusoidal kernels experimentally in Section 4.3. We note that the ReLU kernel is bounded but has divergent slope at $c \to 1$, a hallmark of ReLU NTKs apparent at all depths.

---

[2]We emphasize that Corollary 3.3 is true even permitting deep networks to use different nonlinearities at different layers. The proof of this result relies on the fact that certain PSD polynomials cannot be written as the composition of two other nonlinear PSD polynomials. We note that additional layers with *linear* activations do not decrease kernel expressivity.
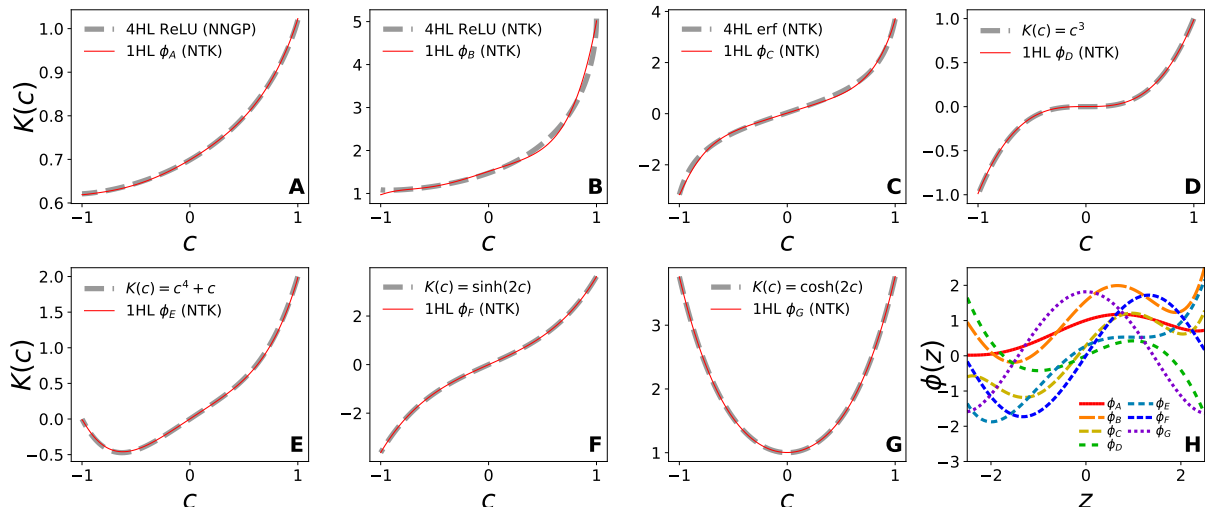
Figure 1. **Any dot-product kernel can be realized as the NTK of a single-hidden-layer FCN. (A-G)** Various desired kernels (grey dashed curves) and empirical NTKs of single-hidden-layer networks (solid red curves). **(H)** The engineered activation functions used in (A-G), which are all polynomials of degree at most 5.

4. Experimental results for finite networks

Our theory shows that arbitrary dot product NNGP kernels or NTKs can be achieved in infinitely-wide 1HL networks with a suitable choice of activation function. Here we explore the experimental implications for finite networks.

All experiments use `JAX` (Bradbury et al., 2018) and `neural_tangents` (Novak et al., 2019b) for network training and kernel computation. Unless otherwise stated, all datasets are normalized to satisfy Assumption 2.1. We report full hyperparameters and provide additional commentary for each experiment in Appendix B.

4.1. Arbitrary NTKs with a Single Hidden Layer

To begin, we illustrate our main result by exhibiting 1HL networks that approximate desired NTKs. For each of seven desired kernels $K(c)$, including three kernels of deep networks, we compute a 5th-order polynomial fit (i.e. approximate $K(c) \approx \sum_{k=0}^{5} a_k c^k$) and use Theorem 3.1 to construct a suitable polynomial activation function ϕ. We then randomly initialize a 1HL network with $\sigma_w = 1, \sigma_b = 0$ and width 2^{18} and compute its empirical NTK as a function of c with toy data $\left\{ \left(\sqrt{2}\cos(\theta), \sqrt{2}\sin(\theta) \right) \right\}_{\theta=0}^{\pi}$.[3] Using the first point in the sequence as one kernel argument, $c = \cos(\theta)$.

The results are shown in Figure 1. In each case, including when mimicking deep neural networks kernels, we find that the 1HL network's empirical NTK is remarkably close to the target kernel. The seven polynomial activation functions

are plotted in Figure 1H.

Discrepancies between target and empirical kernels in Figure 1 are due to a combination of polynomial fitting error and finite width fluctuations. We found that these trade off: using a higher-order polynomial fit for $K(c)$ reduces fitting error but then generally requires higher width to achieve the same attenuation of finite-width fluctuations. This is a consequence of the fact that approximating $\mathbb{E}_{z \sim \mathcal{N}(0,1)} \left[|z^k| \right]$ by random sampling requires exponentially more samples as k increases. We also note that, as shown in Figure 1B, the 4HL ReLU kernel has divergent slope at $c = 1$, making low-order polynomial approximations inadequate. In the next section, we provide a solution for these problems, designing a trainable shallow network that mimics a deep ReLU kernel more faithfully and at narrower width.

4.2. Achieving 4HL Behavior with a 1HL Network

When choosing an FCN architecture, it is common practice to include different depths in the hyperparameter search. It is of course often the case that the best performance comes from networks with multiple hidden layers. However, our main result suggests that, to the extent that these deep networks operate in the kernel regime, it should be possible to design a 1HL FCN that both trains and generalizes like a desired deep FCN, perhaps using far fewer parameters.

Here we demonstrate this capability by designing a 1HL network that mimics the behavior of a 4HL ReLU network. This task presents two main difficulties for our Hermite polynomial activations: (1) ReLU NTKs have divergent slope at $c = 1$ and are poorly fit by low-order polynomials,

---

[3]The factors of $\sqrt{2}$ serve to enforce Assumption 2.1.
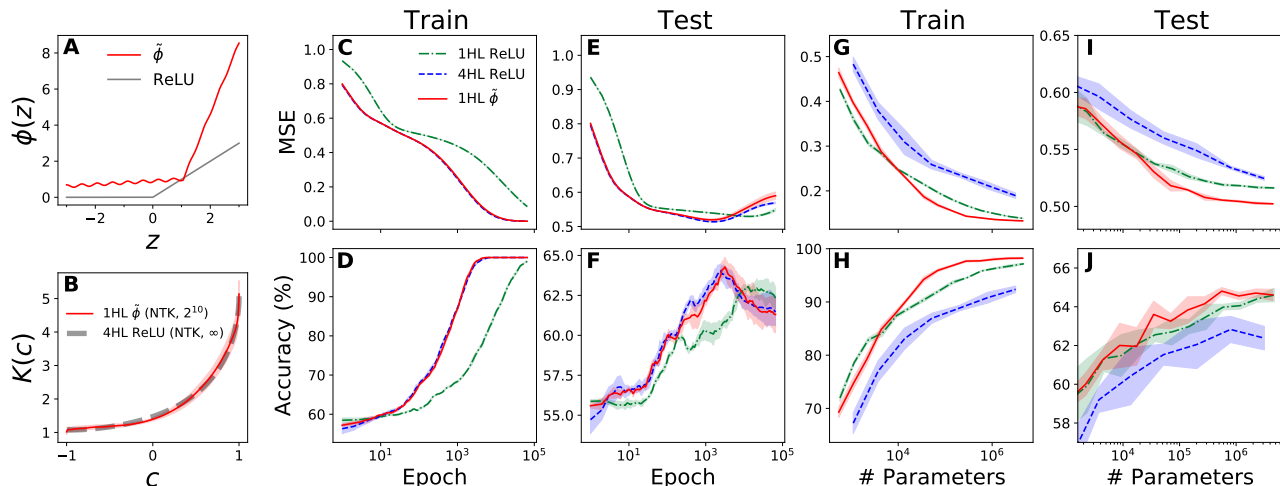
Figure 2. **A 1HL network with activation function designed to mimic a 4HL ReLU NTK achieves the deep architecture's training and generalization behavior with far fewer parameters. (A)** The optimized activation function $\tilde{\phi}$ designed to mimic the infinite-width NTK of a 4HL ReLU network, with ReLU for comparison. **(B)** The empirical NTK of a width-2^{10} 1HL network with activation function $\tilde{\phi}$, with the target deep NTK for comparison. In all subplots, shaded regions show 1σ variation due to random initialization. **(C-F)** Train and test MSE and accuracy for the `wine-quality-red` task throughout training for finite 1HL ReLU, 4HL ReLU, and 1HL $\tilde{\phi}$ networks. Metrics for the shallow $\tilde{\phi}$ network closely match those of the deep ReLU network. **(G-J)**. Final train and test MSE and accuracy for the three architectures with various widths, plotted as a function of the total number of parameters.

and (2) we find that networks with polynomial activations generally diverge upon training. We circumvent both problems by instead using an activation function with a different parameterization: we begin with an activation function of the variational ansatz

$$\tilde{\phi}(z) = \alpha \operatorname{ReLU}(z - \beta) + \gamma \cos(\delta z + \epsilon) + \zeta z + \eta, \quad (19)$$

then numerically optimize $\{\alpha, ..., \eta\}$ such that the NTK of the 1HL network closely matches that of the 4HL ReLU network. We choose this ansatz because of the observation that ReLU activations give sharply peaked kernels as we require here; in the optimized version, the ReLU term comprises the dominant contribution to $\tilde{\phi}(z)$. The optimized activation function is shown in Figure 2A, and its NTK is shown Figure 2B. In this procedure, Theorem 3.1 serves as an existence proof of many activation functions achieving the desired kernel, and our numerical procedure allows us to find one with desirable properties.

We next confirm that shallow networks with activation function $\tilde{\phi}$ indeed mimic the training and generalization behavior of deep ReLU networks. We train width 4096 1HL ReLU, 4HL ReLU, and 1HL $\tilde{\phi}$ networks on the UCI `wine-quality-red` task (800 train samples, 799 test samples, 11 features, 6 classes) with full-batch gradient descent, mean-squared-error (MSE) loss, and step size 0.1. As shown in Figures 2(C-F), we find that, at all times during training, the train MSE, test MSE, train accuracy, and test accuracy for the engineered shallow net closely track those for the 4HL ReLU network. Like the 4HL ReLU network,

the 1HL $\tilde{\phi}$ network reaches low training error many times faster than the 1HL ReLU network.

Even if their asymptotic performance is the same, shallow FCNs in practice often have the advantage of using far fewer parameters, with the total parameter count scaling as $\mathcal{O}(\text{width})$ for shallow models but $\mathcal{O}(\text{width}^2)$ for deep models. It is therefore plausible that our 1HL $\tilde{\phi}$ network, while matching a 4HL ReLU network in generalization and training speed when both are wide, is in fact superior to a 4HL ReLU network when both have the same total number of parameters. To test this hypothesis, we train models of many different widths, with appropriate stopping times for each architecture class chosen via cross-validation to maximize validation accuracy. Note that the 1HL ReLU network needed to be trained 10x longer than the other networks for optimal performance.

As shown in Figures 2(G,H), at a fixed parameter budget, our engineered shallow network achieves significantly lower training MSE and higher training accuracy than either alternative, suggesting it makes much more efficient use of its parameters in memorizing the training set. More importantly, as shown in Figures 2(I,J), our engineered shallow network consistently outperforms the 4HL ReLU architecture at a fixed parameter budget. Furthermore, with $> 3 \times 10^4$ parameters, it outperforms the 1HL ReLU network as well.

We repeat this experiment using the `balance-scale` dataset (313 train samples, 312 test samples, 4 features, 3 classes), `breast-cancer-wisc-diag` dataset (285

train samples, 284 test samples, 30 features, 2 classes), and report results in Figures A.4 and A.5. In both cases, we find that the 1HL $\tilde{\phi}$ network closely mimics the training behavior of the 4HL ReLU net and learned the training data better than the alternatives at fixed parameter budget. As with the `wine-quality-red` dataset, on these datasets, the 1HL $\tilde{\phi}$ network consistently outperforms the 4HL ReLU network at fixed parameter budget and typically outperforms the 1HL ReLU architecture as well. We also test our procedure on a subset of `CIFAR-10` (Krizhevsky, 2009) (1000 train samples, 1000 test samples, 3072 features, 10 classes) and report results in Figure A.6. For this high-dimensional dataset, we still see good agreement between the 4HL ReLU and 1HL mimic net, but, due to the dimensionality of the first weight matrix, the 1HL mimic net has lots its advantage in terms of parameter count.

All experiments thus far have normalized train and test data in accordance with Assumption 2.1. While this assumption was required by our theory, one might wonder if it is necessary to respect in practice. To test this, we repeat the above experiments with the `wine-quality-red` and `balance-scale` datasets with only normalizing input vectors on average before use. As reported in Figures A.7 and A.8, the results vary somewhat: for the `wine-quality-red` dataset, the match is still almost perfect in all metrics except test MSE at late times, while for the `balance-scale` dataset, agreement can be poor depending upon which metric is observed. We leave the elucidation of what factors affect the quality of agreement for future work, though from NTK theory, one would expect that a larger dataset size would induce greater kernel evolution and thus cause greater divergence.

We stress three important takeaways from these experiments:

1. Shallow FCNs can indeed train and generalize like deep FCNs, even at finite width.

2. Our mimic network was the best-performing architecture for many datasets and parameter budgets, suggesting that our approach (and even our specific $\tilde{\phi}$) may be practically useful in use cases where parameter efficiency is important.

3. Most significantly, our kernel engineering paradigm enabled us to design a network architecture in a *principled way*: we began with a clear design goal — to realize a particular target NTK — and our final network closely followed expected behavior. We contrast this with the typical black-box trial-and-error method of neural architecture search.[4]

---

[4]For example, Ramachandran et al. (2018) also find a high-performing activation function (i.e. Swish), but do so via an ex-

It is also noteworthy that, in almost all cases, FCN performance improves essentially monotonically with width (as also seen by Lee et al. (2019)), suggesting that the limiting kernel regime is optimal and justifying the network kernel as the sole consideration in the design of $\tilde{\phi}$.

4.3. Case Study: the Parity Problem

As a second test of our kernel engineering paradigm, we consider the classic *parity problem*. The domain of the parity problem is the boolean cube $\{-1, +1\}^{n_{\text{bits}}}$, and the target function is

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \text{ contains an odd number of +1s,} \\ -1 & \text{else.} \end{cases} \tag{20}$$

We take $n_{\text{bits}} = 11$ and randomly select half the domain as train data and the other half as test data. Despite its simplicity, the parity problem is notoriously difficult for both FCNs and kernel machines, and despite the fact that a deep ReLU network can easily *represent* the correct solution (Hohil et al., 1999; Bengio et al., 2007), in practice it is not learned during training (Bengio et al., 2006; Nye & Saxe, 2018).

The difficulty of the parity problem lies in the fact that points are anticorrelated with their neighbors: flipping any single bit changes the function. By contrast, most standard FCN kernels have high value when evaluated on nearby points (i.e. $K(c)$ is large when c is nearly but not exactly 1), so they expect neighboring points to be *correlated*, not anticorrelated. This can be understood through the lens of the kernel's eigensystem, revealing that, for many FCN kernels, the parity problem is in fact the hardest-to-learn function on the boolean cube (Yang & Salman, 2019; Simon et al., 2021).

However, we can hope to achieve better performance by engineering a kernel that is better-suited to the problem. We desire a kernel with a rapid decay as c decreases from 1.[5] Further noting that the target function has odd symmetry with $n_{\text{bits}} = 11$, we also want the kernel to be odd. We would expect that inference with such a kernel predicts correctly on the half of test points that are opposite a training point and predicts close to zero on other test points, giving accuracy of 75% and MSE of 0.5.

We can achieve such a sharp, odd kernel with a sinusoidal activation function. Expanding the kernel of Equation 16 in

---

pensive exhaustive search that yields little insight into the reasons for high performance.

[5]Though it might seem we should design a kernel that not only decays but quickly becomes negative as c decreases from 1, this is in fact impossible for an FCN kernel as it violates the PSD constraint of Theorem 2.2.
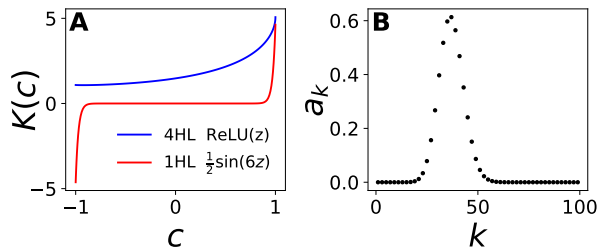
Figure 3. **(A)** NTKs of two networks evaluated on the parity problem. The NTK of the 1HL network with $\frac{1}{2}\sin(6z)$ activations has a rapid decay and odd symmetry, making it well-suited to the parity problem. The NTK of the 1HL network with $10\sin(6z)$ activations (not shown) is identical but $400\times$ greater in magnitude. **(B)** Odd power series coefficients of the NTK of the 1HL $\frac{1}{2}\sin(6z)$ network. Even coefficients (not shown) are zero.

a power series, we find that

$$K_{\sin(az)}^{(\text{NTK})}(c) = e^{-a^2} \sum_{\substack{k \geq 1 \\ k \text{ odd}}} \frac{a^{2k}}{k!}(1+k)c^k, \qquad (21)$$

which has only odd coefficients that peak at index $k \approx a^2$, which, if a is moderately large, implies the kernel is mostly high-order and thus rapidly decays to zero away from $c = \pm 1$. We choose $a = 6$ and test both $\phi(z) = 10\sin(6z)$ and $\phi(z) = \frac{1}{2}\sin(6z)$. We contrast ReLU and sine kernels in Figure 3.

We trained 4HL ReLU networks and 1HL sine networks via gradient descent with MSE loss. The results are shown in Table 1. The 4HL ReLU network performs significantly worse than chance[6] in terms of both test MSE and test accuracy. The first 1HL sine network roughly achieved the "ideal odd kernel" performance we anticipated, which is why we include it in the table. The second 1HL sine network, however, *genuinely learns the target function*, achieving near-zero MSE and perfect accuracy in every run. This success surprised us; we hypothesize that it is related to the fact that the parity function can be written as $f(\mathbf{x}) = \sin(\frac{\pi}{2}\sum_i \mathbf{x}_i)$.[78]

As with the prior experiment, we stress that, though our method does dramatically outperform the FCN baseline on the parity problem, the most important takeaway is not the

[6]We define the naive "chance" predictor as one that always predicts 0 and chooses class labels randomly.

[7]The superiority of this shallow architecture is especially noteworthy given that the parity problem was historically used to illustrate the need for deep architectures (Bengio et al., 2007).

[8]The constants in our ultimate sinusoidal activation functions — 10, $\frac{1}{2}$, and 6 — were chosen after experimentation to best illustrate the two different behaviors highlighted in Table 1. Similar choices gave similar results. We leave as an open problem an explanation of why only the smaller prefactor led to perfect generalization. The answer will require ideas beyond the kernel regime.

depth	$\phi(z)$	test MSE	test accuracy (%)
4	ReLU(z)	2.819 ± 0.109	29.326 ± 1.482
1	$10\sin(6z)$	0.668 ± 0.332	82.048 ± 8.550
1	$\frac{1}{2}\sin(6z)$	$\mathbf{0.021 \pm 0.004}$	$\mathbf{100 \pm 0}$
chance		1	50
ideal odd kernel		.5	75

Table 1. **A kernel-informed choice of activation function dramatically improves performance on the parity problem.** Test MSE and accuracy are shown for a deep ReLU network and two single-hidden-layer networks with kernel-informed activation functions. Metrics for a naive random predictor and an ideal odd kernel are provided for comparison.

high performance itself but rather the fact that we achieved it by designing a neural network in a principled way by selecting and reverse-engineering a kernel suited to the task at hand. Unlike traditional approaches, our method required very little trial-and-error or hyperparameter optimization.

5. Conclusions

Motivated by the need for principled neural architecture design, we have derived a constructive method for achieving any desired positive semidefinite dot-product kernel as an FCN with one hidden layer. In a series of experiments, we verified our construction and showed that this reverse engineering paradigm can indeed enable the design of shallow FCNs with surprising and desirable properties including improved generalization, better parameter efficiency, and deep-FCN behavior with only one hidden layer.

The fact that any FCN kernel can be achieved with just one hidden layer is a surprise with potentially broad implications for deep learning theory. As with most facets of deep learning, the role and value of depth are not well understood. Our results suggest that depth may in fact not be important for FCNs. If this is true, it means that researchers aiming to understand the value of depth should focus on convolutional and other more structured architectures.

Our results open many avenues for future work. On the theoretical side, one might aim to lift the normalization condition on the data or achieve both desired NNGP and NTK kernels simultaneously, likely in a deep network. We also note that the activation functions given by our construction generally have a large number of arbitrary signs, and though all sign choices yield the same limiting kernels, some will outperform others at finite width. Deriving a principle to choose these signs is a potential avenue for future work, and one tool for doing so might be finite-width corrections to the NTK (e.g. Dyer & Gur-Ari (2019); Zavatone-Veth & Pehlevan (2021)). On the experimental side, there is ample room to apply our method to other tabular data tasks, explore its

interactions with regularization and feature learning, and use the large body of work on kernel selection to choose the kernel to reverse engineer (Cristianini et al., 2006; Cortes et al., 2012; Jacot et al., 2020).

Though our experiments focus on the NTK, our results also allow the design of a network's NNGP kernel and thus its Bayesian priors. The ability to choose task-appropriate priors instead of defaulting to those of standard architectures would be a valuable tool for the study and practice of Bayesian neural networks (e.g. Wilson & Izmailov (2020)), another promising direction for future investigation.

Perhaps the most interesting extension of our work would be the derivation of similar results for convolutional architectures, characterizing their kernel expressivity and finding an analog to our reverse engineering construction. Recent results have shed much light on convolutional kernels' eigenspectra and dependence on hyperparameters (Xiao, 2021; Misiakiewicz & Mei, 2021) so this lofty goal may indeed be within reach. If achieved, we hope the extension of our work to convolutional nets might in time enable the first-principles design of state-of-the-art architectures for the first time.

Acknowledgements

References

Allen-Zhu, Z., Li, Y., and Song, Z. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pp. 242–252. PMLR, 2019.

Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pp. 322–332. PMLR, 2019a.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., and Wang, R. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8139–8148, 2019b.

Bengio, Y., Delalleau, O., and Le Roux, N. The curse of highly variable functions for local kernel machines. *Advances in neural information processing systems*, 18: 107, 2006.

Bengio, Y., LeCun, Y., et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007. shows there's a small deep net solution to the parity problem.

Bordelon, B., Canatar, A., and Pehlevan, C. Spectrum dependent learning curves in kernel regression and wide neural networks. In *International Conference on Machine Learning*, pp. 1024–1034. PMLR, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X., and Gu, Q. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019.

Cho, Y. and Saul, L. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2009.

Cortes, C., Mohri, M., and Rostamizadeh, A. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012.

Cristianini, N., Kandola, J., Elisseeff, A., and Shawe-Taylor, J. On kernel target alignment. In *Innovations in machine learning*, pp. 205–256. Springer, 2006.

Daniely, A., Frostig, R., and Singer, Y. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2253–2261, 2016.

Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pp. 1675–1685. PMLR, 2019.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Dyer, E. and Gur-Ari, G. Asymptotics of wide networks from feynman diagrams. *arXiv preprint arXiv:1909.11304*, 2019.

Hohil, M. E., Liu, D., and Smith, S. H. Solving the n-bit parity problem using neural networks. *Neural Networks*, 12(9):1321–1323, 1999.

Hron, J., Bahri, Y., Sohl-Dickstein, J., and Novak, R. Infinite attention: NNGP and NTK for deep attention networks. In *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4376–4386. PMLR, 2020.

Jacot, A., Hongler, C., and Gabriel, F. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8580–8589, 2018.

Jacot, A., Şimşek, B., Spadaro, F., Hongler, C., and Gabriel, F. Kernel alignment risk estimator: risk prediction from training data. *arXiv preprint arXiv:2006.09796*, 2020.

Kar, P. and Karnick, H. Random feature maps for dot product kernels. In *Artificial intelligence and statistics*, pp. 583–591. PMLR, 2012.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. Deep neural networks as gaussian processes. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8570–8581, 2019.

Lee, J., Schoenholz, S. S., Pennington, J., Adlam, B., Xiao, L., Novak, R., and Sohl-Dickstein, J. Finite versus infinite neural networks: an empirical study. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Lin, H. W., Tegmark, M., and Rolnick, D. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, pp. 1223–1247, 2017.

Matthews, A. G. d. G., Rowland, M., Hron, J., Turner, R. E., and Ghahramani, Z. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

Mhaskar, H., Liao, Q., and Poggio, T. A. When and why are deep networks better than shallow ones? In Singh, S. P. and Markovitch, S. (eds.), *AAAI Conference on Artificial Intelligence*, pp. 2343–2349. AAAI Press, 2017.

Misiakiewicz, T. and Mei, S. Learning with convolution and pooling operations in kernel methods. *arXiv preprint arXiv:2111.08308*, 2021.

Neal, R. M. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pp. 29–53. Springer, 1996.

Novak, R., Xiao, L., Bahri, Y., Lee, J., Yang, G., Hron, J., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2019a.

Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J., and Schoenholz, S. S. Neural tangents: Fast and easy infinite neural networks in python. *CoRR*, abs/1912.02803, 2019b.

Nye, M. and Saxe, A. Are efficient deep representations learnable? *arXiv preprint arXiv:1807.06399*, 2018. shows the basins for the parity problem are small.

O'Donnell, R. *Analysis of boolean functions*. Cambridge University Press, 2014.

Pennington, J., Felix, X. Y., and Kumar, S. Spherical random features for polynomial kernels. *Advances in neural information processing systems*, 2015.

Poggio, T. A., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *Int. J. Autom. Comput.*, 14(5):503–519, 2017. doi: 10.1007/s11633-017-1054-2.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. Exponential expressivity in deep neural networks through transient chaos. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3360–3368. Curran Associates, Inc., 2016.

Raghu, M., Poole, B., Kleinberg, J. M., Ganguli, S., and Sohl-Dickstein, J. On the expressive power of deep neural networks. In Precup, D. and Teh, Y. W. (eds.), *International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2847–2854. PMLR, 2017.

Rahimi, A. and Recht, B. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1313–1320. Curran Associates, Inc., 2008.

Rahimi, A., Recht, B., et al. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 3, pp. 5, 2007.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

Rolnick, D. and Tegmark, M. The power of deeper networks for expressing natural functions. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

Schoenberg, I. J. Positive definite functions on spheres. *Duke Mathematical Journal*, 9(1):96 – 108, 1942. doi: 10.1215/S0012-7094-42-00908-6.

Schoenholz, S. S., Gilmer, J., Ganguli, S., and Sohl-Dickstein, J. Deep information propagation. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017.

Shawe-Taylor, J., Cristianini, N., et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

Shi, X., Chen, J., and Wang, L. Heuristic search for activation functions of neural networks based on gaussian processes. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Simon, J. B., Dickens, M., and DeWeese, M. R. Neural tangent kernel eigenvalues accurately predict generalization. *arXiv preprint arXiv:2110.03922*, 2021.

Telgarsky, M. Benefits of depth in neural networks. In *Conference on Learning Theory (COLT)*, volume 49 of *JMLR Workshop and Conference Proceedings*, pp. 1517–1539. JMLR.org, 2016.

Wilson, A. G. and Izmailov, P. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.

Xiao, L. Eigenspace restructuring: a principle of space and frequency in neural networks. *arXiv preprint arXiv:2112.05611*, 2021.

Yang, G. Tensor programs I: wide feedforward or recurrent neural networks of any architecture are gaussian processes. *CoRR*, abs/1910.12478, 2019.

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

Zavatone-Veth, J. and Pehlevan, C. Exact marginal prior distributions of finite bayesian neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Zou, D., Cao, Y., Zhou, D., and Gu, Q. Stochastic gradient descent optimizes over-parameterized deep relu networks. arxiv e-prints, art. *arXiv preprint arXiv:1811.08888*, 2018.
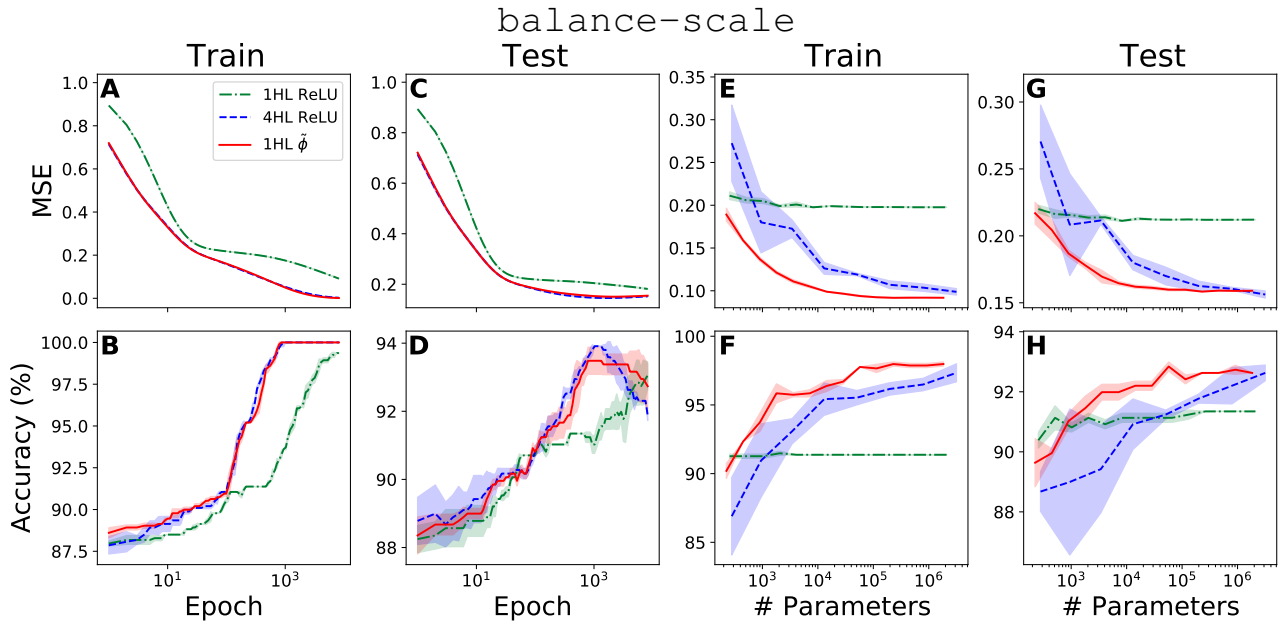
A. Additional Figures

Figure A.4. **(A-D)** Train and test MSE and accuracy for the `balance-scale` task throughout training for finite 1HL ReLU, 4HL ReLU, and 1HL $\tilde{\phi}$ networks. Metrics for the shallow $\tilde{\phi}$ network closely match those of the deep ReLU network. **(E-H)**. Final train and test MSE and accuracy for the three architectures with various widths, plotted as a function of the total number of parameters.
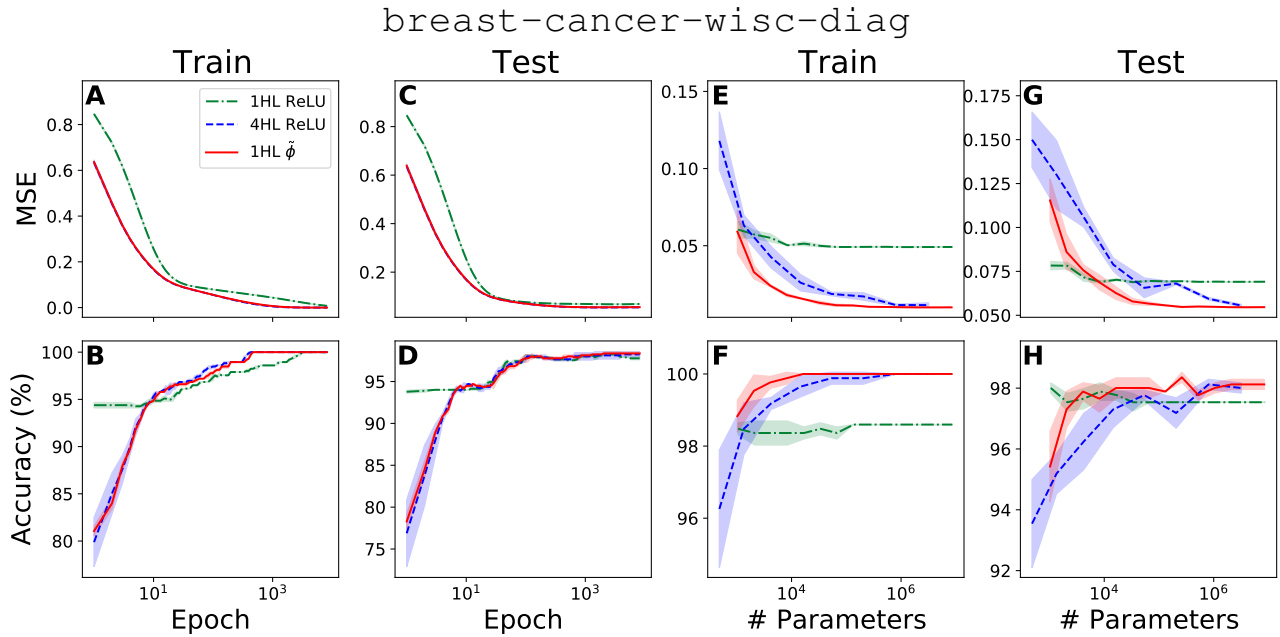

Figure A.5. **(A-D)** Train and test MSE and accuracy for the `breast-cancer-wisc-diag` task throughout training for finite 1HL ReLU, 4HL ReLU, and 1HL $\tilde{\phi}$ networks. Metrics for the shallow $\tilde{\phi}$ network closely match those of the deep ReLU network. **(E-H)**. Final train and test MSE and accuracy for the three architectures with various widths, plotted as a function of the total number of parameters.
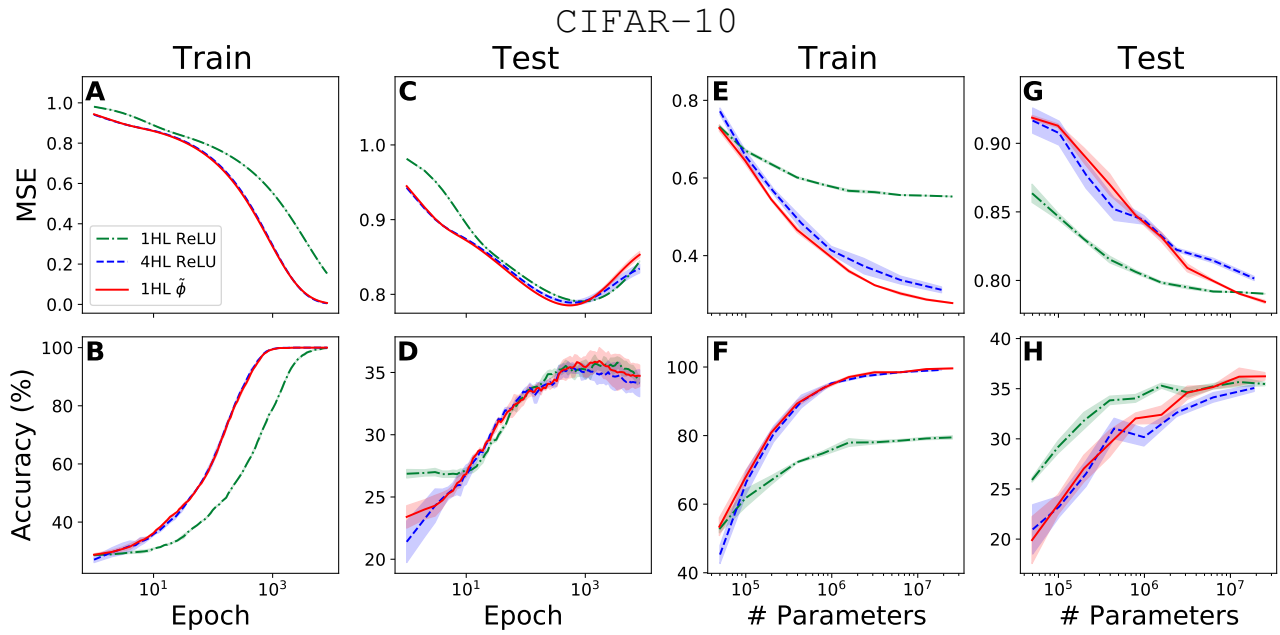
CIFAR-10

Figure A.6. **(A-D)** Train and test MSE and accuracy for a size-1k subset of `CIFAR-10` throughout training for finite 1HL ReLU, 4HL ReLU, and 1HL $\tilde{\phi}$ networks. Metrics for the shallow $\tilde{\phi}$ network closely match those of the deep ReLU network. **(E-H)**. Final train and test MSE and accuracy for the three architectures with various widths, plotted as a function of the total number of parameters. Due to the high input dimension, the 1HL mimic net has little to no advantage over the 4HL ReLU net in terms of parameter efficiency.
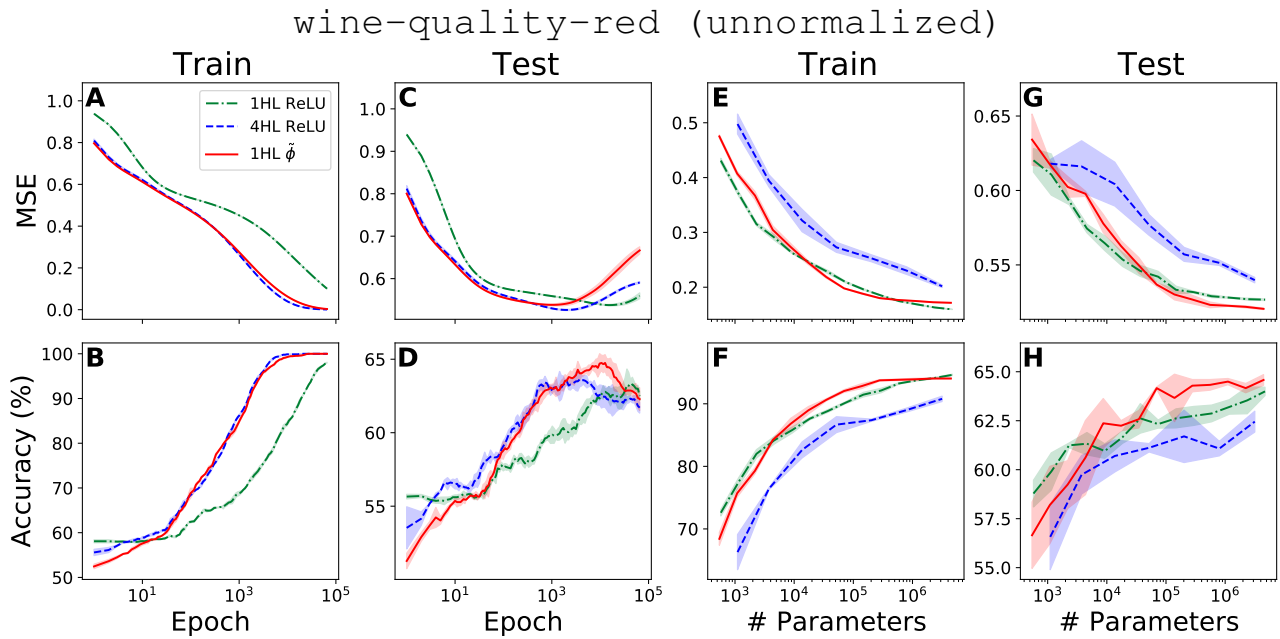
wine-quality-red (unnormalized)

Figure A.7. **(A-D)** Train and test MSE and accuracy for the `wine-quality-red` task *without uniform input normalization*. Metrics for the shallow $\tilde{\phi}$ network nonetheless still closely match those of the deep ReLU network. **(E-H)**. Final train and test MSE and accuracy for the three architectures with various widths, plotted as a function of the total number of parameters.

Figure A.8. **(A-D)** Train and test MSE and accuracy for the `balance-scale` task *without uniform input normalization*. For this task, metrics for the shallow $\tilde{\phi}$ network differ from those of the deep ReLU network. **(E-H)**. Final train and test MSE and accuracy for the three architectures with various widths, plotted as a function of the total number of parameters.
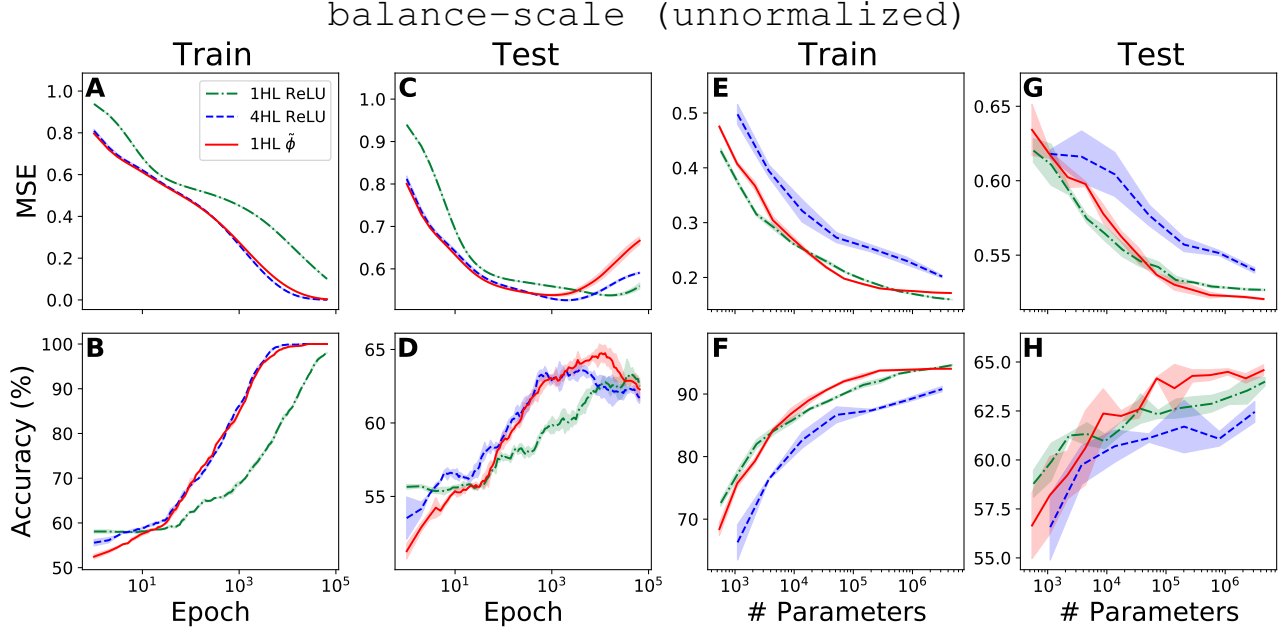
B. Experimental Details

For all engineered 1HL networks, we use $\sigma_w = 1, \sigma_b = 0$. For all ReLU and erf networks, we use $\sigma_w = \sqrt{2}, \sigma_b = 0.1$ for all layers except the readout layer, for which we use $\sigma_w = 1, \sigma_b = 0$. We define MSE as $\mathbb{E}_{(x,y)}\left[(f(x) - y)^2\right]$, without the factor of $\frac{1}{2}$.

Naively, when training an infinitely-wide network, the NTK only describes the *mean* learned function, and the true learned function will include an NNGP-kernel-dependent fluctuation term reflecting the random initialization. However, by *centering* the model — that is, storing a copy of the parameters at $t = 0$ and redefining $\hat{f}_t(x) := \hat{f}_t(x) - \hat{f}_0(x)$ throughout optimization and at test time — this term becomes zero (Lee et al., 2020), and so we neglect the NNGP kernel in our activation function design and use this trick in our experiments.

Arbitrary NTKs with a Single Hidden Layer. When fitting a polynomial to $K(c)$, we use a least-squares fit with 10% of the weight on the point at $c = 1$ to encourage better agreement with the 4HL ReLU NTK.

In terms of Hermite polynomials, the activation functions plotted in Figure 1 are as follows:

$$\phi_A = 0.837\,h_0 + 0.271\,h_1 - 0.151\,h_2 - 0.050\,h_3 + 0.101\,h_4 + 0.084\,h_5, \tag{22}$$

$$\phi_B = 1.230\,h_0 + 0.639\,h_1 + 0.539\,h_4 + 0.426\,h_5, \tag{23}$$

$$\phi_C = 0.153\,h_0 + 0.826\,h_1 - 0.004\,h_2 - 0.005\,h_3 + 0.219\,h_4 + 0.577\,h_5, \tag{24}$$

$$\phi_D = 0.001\,h_0 + 0.010\,h_1 - 0.500\,h_3 + 0.009\,h_5, \tag{25}$$

$$\phi_E = 0.707\,h_1 - 0.004\,h_2 - 0.003\,h_3 + 0.447\,h_4, \tag{26}$$

$$\phi_F = 1.001\,h_1 - 0.572\,h_3 + 0.004\,h_4 + 0.229\,h_5, \tag{27}$$

$$\phi_G = 1.002\,h_0 - 0.805\,h_2 + 0.404\,h_4 + 0.011\,h_5. \tag{28}$$

We found that inverting the sign of the h_2 and h_3 coefficients, while not changing the network's infinite-width NTK, did reduce its finite-width fluctuations and improve the agreement illustrated in Figure 1. Instead of directly computing the empirical NTK of a single 1HL network with width 2^{18}, which required too much memory, we equivalently averaged the empirical NTKs of 2^4 1HL networks with width 2^{14}.

Achieving 4HL Behavior with a 1HL Network.

We first optimize the activation function with form given by Equation 19. We use the Nelder-Mead algorithm to minimize the sum squared error between the target kernel and the engineered 1HL kernel evaluated on values $c \in [-1, 1]$. The desired kernel is the infinite width 4HL ReLU NTK. One would compare this with the analytical, infinite-width NTK of the 1HL network, but we find that, because our choice of activation function has discontinuous derivative (and is not ReLU), `neural_tangents` has difficulty computing its τ-transform, so we instead compare against a randomly-initialized empirical kernel with width 2^{12}. To attenuate statistical fluctuations, we average the empirical kernel over 20 initializations. This process yields the activation function

$$\tilde{\phi}(z) = 3.8001\,\mathrm{ReLU}(z - 1.0600) - 0.0794\cos(11.8106z + 0.9341) + 0.0968z + 0.9010. \tag{29}$$

We then use this activation function to train finite width nets and compare the performance of 1HL ReLU, 4HL ReLU, and 1HL $\tilde{\phi}$ networks. For networks of width 4096, we train each net for 2^{16} epochs, averaging the results over five random initializations. Using $k = 3$-fold cross-validation on the training data, we choose the optimal stopping time for each net and then train networks of varying width, again averaging results over five random initializations. For the experiments in the main text using the `wine-quality-red` dataset, we use stopping times of 2000 for the 4HL ReLU and $\tilde{\phi}$ networks and 20000 for the 1HL $\tilde{\phi}$ networks. Experiments on the `balance-scale` dataset were identical, but used stopping times of 400 epochs for all networks. Experiments on the `breast-cancer-wisc-diag` datasets used stopping times of 2^{10} epochs for all architectures. Experiments on the `CIFAR-10` datasets used stopping times of 800 epochs for all architectures.

All datasets except `CIFAR-10` were taken from the UCI repository (Dua & Graff, 2017).

Case Study: the Parity Problem. We use networks of width 128, trained via full-batch gradient descent with step size 0.1. We stop when either train MSE is below 10^{-3} or after 10k epochs. The 1HL sine networks always stopped due to low train MSE, while the 4HL ReLU networks, which trained much slower, generally timed out. The slow training of ReLU networks here can be understood as a consequence of the presence of very small eigenvalues in their NTK data-data kernel matrix (Cao et al., 2019; Yang & Salman, 2019). We average performance over 30 trials.

C. Review of Hermite Polynomials

As Hermite polynomials are of central importance to our main result, here we provide a more complete introduction. We note that there are differing conventions for Hermite polynomials; in this work we use the *normalized probabilist's* Hermite polynomials. The first several such polynomials and the general formula are

$$h_0(z) = 1,$$
$$h_1(z) = z,$$
$$h_2(z) = \frac{1}{\sqrt{2}}\left(z^2 - 1\right),$$
$$h_3(z) = \frac{1}{\sqrt{3!}}\left(z^3 - 3z\right),$$
$$\cdots$$
$$h_k(z) = \sum_{\substack{\ell=0 \\ k-\ell\ \mathrm{even}}}^{k} \frac{(-1)^{(k-\ell)/2}\sqrt{k!}}{(k-\ell)!!\ell!}z^\ell,$$
$$\cdots$$

In addition to those given in the text, we note the useful identities

$$h_k(z) = \frac{z}{\sqrt{k}}h_{k-1}(z) - \sqrt{\frac{k-1}{i}}h_{k-2}(z), \tag{30}$$
$$h_k(-z) = (-1)^k h_k(z). \tag{31}$$

Because the Hermite polynomials are orthonormal w.r.t. the standard normal Gaussian metric, we can isolate a particular coefficient of $\phi(z) = \sum_{i=0}^{\infty} b_i h_i(z)$ by computing

$$b_i = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-z^2/2} h_i(z) \phi(z) dz. \tag{32}$$

With the use of Equation 9, the Hermite decomposition of an activation function immediately yields its τ-transform (though for special functions there are often faster methods of obtaining it, such as directly computing the expectation in Equation 5). Here we reproduce results from Daniely et al. (2016) and Cho & Saul (2009) for the τ-transforms of a few notable functions:

$$\tau_{e^{az}}(c) = e^{a^2(c^2+1)}, \tag{33}$$

$$\tau_{\sin(az)}(c) = e^{-a^2} \sinh(a^2 c), \tag{34}$$

$$\tau_{\cos(az)}(c) = e^{-a^2} \cosh(a^2 c), \tag{35}$$

$$\tau_{\text{ReLU}(z)}(c) = \frac{1}{2\pi} \left[(\pi - \cos^{-1}(c))c + (1 - c^2)^{1/2} \right]. \tag{36}$$

D. Proofs of Equation 6, Corollary 3.2, and Corollary 3.3

Proof of Equation 6. First, we Hermite-expand ϕ as $\phi(z) = \sum_{k=0}^{\infty} b_k h_k(\sigma^{-1} z)$. This yields

$$\tau_\phi(c; \sigma^2) = \sum_{k=0}^{\infty} b_k^2 c^k. \tag{37}$$

Using Equation 10 to take the derivatives of h_k, we find that $\phi'(z) = \sum_{k=0}^{\infty} b_k \sigma^{-1} \sqrt{k} h_{k-1}(\sigma^{-1} z)$, and thus

$$\tau_{\phi'}(c; \sigma^2) = \sigma^{-2} \sum_{k=0}^{\infty} b_k^2 k c^{k-1} = \frac{\partial_c}{\sigma^2} \tau_\phi(c; \sigma^2). \qquad \square$$

A related equation appears in Appendix A of Poole et al. (2016) but is not derived, and this equation for $c = 1$ is derived by Daniely et al. (2016), so we do not claim priority, but we nonetheless hope this proof of the general identity will be useful for the community.

Proof of Corollary 3.2. *Property (a).* Setting $\sigma_b = 0$ in Equations 7 and 8 and replacing $\tau_{\phi'}$ with τ_ϕ using Equation 6 yields that

$$K_\phi^{(\text{NNGP})}(c) = \sigma_w^2 \tau_\phi(c; \sigma_w^2), \tag{38}$$

$$K_\phi^{(\text{NTK})}(c) = K^{(\text{NNGP})}(c) + c\partial_c \tau_\phi(c; \sigma_w^2). \tag{39}$$

Replacing ϕ with ϕ' and again using Equation 6, we find that

$$K_{\phi'}^{(\text{NNGP})}(c) = \partial_c \tau_\phi(c; \sigma_w^2) = \frac{\partial_c}{\sigma_w^2} K_\phi^{(\text{NNGP})}(c), \tag{40}$$

$$K_{\phi'}^{(\text{NTK})}(c) = K_{\phi'}^{(\text{NNGP})}(c) + \frac{c\partial_c^2}{\sigma_w^2} \tau_\phi(c; \sigma_w^2) = \frac{\partial_c}{\sigma_w^2} K_\phi^{(\text{NTK})}(c). \tag{41}$$

Property (b). This follows from the fact that $\tau_{\alpha\phi}(c; \sigma^2) = \alpha^2 \tau_\phi(c; \sigma^2)$.

Property (c). This follows from the fact that $\tau_{\phi(-z)}(c; \sigma^2) = \tau_{\phi(z)}(c; \sigma^2)$. $\qquad \square$

Proof of Corollary 3.3. This corollary is true because the kernels of deep FCNs are compositional in their functional form, but certain functions cannot be decomposed into multiple nonlinear PSD polynomials. To show this, we begin by reproducing the recursion relation for the NNGP kernel (see, e.g., Appendix E of Lee et al. (2019)). In terms of the kernel at layer ℓ, the kernel at layer $\ell + 1$ is

$$K_{\ell+1}^{(\text{NNGP})}(c) = \sigma_w^2 \tau_\phi \left(\frac{K_\ell^{(\text{NNGP})}(c)}{K_\ell^{(\text{NNGP})}(1)}; K_\ell^{(\text{NNGP})}(1) \right) + \sigma_b^2, \tag{42}$$

with base condition $K_0^{(\text{NNGP})}(c) = \sigma_w^2 c + \sigma_b^2$. If ϕ is nonlinear (i.e. is not an affine function), then the τ-transform in Equation 42 is nonlinear in its first argument, and we have $K_{\ell+1}^{(\text{NNGP})}(c) = f(K_\ell^{(\text{NNGP})}(c))$ for some nonlinear PSD function f.

Thus, after two or more nonlinear hidden layers, the NNGP kernel consists of the composition of multiple nonlinear PSD functions. However, certain PSD polynomials cannot be decomposed into multiple nonlinear PSD functions in this way. For example,

$$K(c) = c^2 + c \tag{43}$$

cannot be written as $K(c) = f(g(c))$ for any nonlinear PSD functions f and g: to be nonlinear and PSD, they must each have a positive term of at least order two when expressed as a power series, but then their composition must have at least a quartic term, and as f and g are PSD, there can be no negative contribution to cancel this high-order term.[9] Kernels such as this can only be achieved with exactly one nonlinear hidden layer (though additional hidden layers with *affine* activation functions are fine).

As can be seen from its recursion relations (again see Appendix E of Lee et al. (2019)), the NTK at any layer takes the form

$$K_\ell^{(\text{NTK})}(c) = K_\ell^{(\text{NNGP})}(c) + [\text{PSD function}], \tag{44}$$

so, if a particular architecture's NNGP kernel must necessarily have extra, unwanted positive terms (such as the inevitable quartic term discussed above), so must the NTK, because the additional PSD contribution cannot cancel any terms. This completes the proof.

---

[9]We note that, while there exist many nonlinear pairs of functions such as $f(c) = c^6 + c^3$, $g(c) = \sqrt[3]{c}$ that yield $K(c) = f(g(c))$, one will always be non-PSD: the flaw with this example is that $\sqrt[3]{c}$ does not have a positive-semidefinite power series expansion and thus cannot be a valid kernel transformation.