AN EMPIRICAL ANALYSIS OF THE LAPLACE AND NEURAL TANGENT KERNELS

A Thesis

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Mathematics

By

Ronaldas Paulius Lencevičius

2022

SIGNATURE PAGE

THESIS:	AN EMPIRICAL ANALYSIS OF THE LAPLACE AND NEURAL TANGENT KERNELS
AUTHOR:	Ronaldas Paulius Lencevičius
DATE SUBMITTED:	Summer 2022
	Department of Mathematics and Statistics

Dr. James Risk _____
Thesis Committee Chair
Mathematics & Statistics

Dr. Manuchehr Aminian _____
Mathematics & Statistics

Dr. Adam King _____
Mathematics & Statistics

ACKNOWLEDGMENTS

To Ada, Percy, and Sabrina,

to Raimis, Diana, Boson, and Curie,

to all my friends, family, and faculty,

...and to Dr. Risk's care, patience, and mentorship.

ABSTRACT

The neural tangent kernel is a kernel function defined over the parameter distribution of an infinite width neural network. Despite the impracticality of this limit, the neural tangent kernel has allowed for a more direct study of neural networks and a gaze through the veil of their black box. More recently, it has been shown theoretically that the Laplace kernel and neural tangent kernel share the same reproducing kernel Hilbert space in the space of $\mathbb{S}^{d-1}$ alluding to their equivalence. In this work, we analyze the practical equivalence of the two kernels. We first do so by matching the kernels exactly and then by matching posteriors of a Gaussian process. Moreover, we analyze the kernels in $\mathbb{R}^d$ and experiment with them in the task of regression.

Contents

List of Tables

List of Figures

List of Algorithms

List of Symbols

L_2	Square-integrable function space
$\mathbb{R}^d$	d-dimensional real space
$\mathbb{S}^{d-1}$	Unit d-sphere space
$\mathcal{H}$	Hilbert space
$\mathcal{H}_0$	Pre-Hilbert space
$\mathcal{H}_k$	Reproducing kernel Hilbert space (RKHS)
$\mathcal{X}$	Data space
$\mathcal{GP}$	Gaussian process (GP)
$\mathcal{MVN}$	Multivariate normal
$\mathcal{N}$	Normal
μ	Metric
X	Matrix
$\mathbf{x}$	Vector
x	Scalar
$\sqcup_*$	Test/predictions
$D = L + 1$	Depth (neural tangent kernel)
R^2	Coefficient of determination
β	Bias parameter (neural tangent kernel)

ℓ	Length-scale parameter (Matérn kernel)
ϵ	Noise
ν	Smoothness parameter (Matérn kernel)
ρ	Pearson correlation coefficient
σ^2	Variance
θ	General parameter set
L	Loss function
T_k	Integral operator
cov/var	Covariance/variance
$\mathbb{E}$	Expected value
k	Kernel
$\|\cdot\|$	Norm
$\langle\cdot,\cdot\rangle$	Inner product
$\sigma(\cdot)$	Activation function
d_θ	Absolute difference function given kernel parameters θ
λ_i	Eigenvalue
ϕ_i	Eigenfunction

Chapter 1

Introduction

The relation between artificial neural networks and Gaussian processes has been established since the early 1990s. In 1989 researchers determined that a single layer neural network can, under the right conditions, approximate any continuous function as the layer width tends to infinity [15, 20, 41, 23]. With this idea, Neal [35] later found that a single hidden layer neural network with normally distributed weights and biases at initialization can be represented as a closed form Gaussian process when the layer width approach infinity and hence converges to a normal distribution. It is then possible to inform neural networks via specified priors and analyze them over the neural network parameter distribution. This was further expanded in 2018 to neural networks of many layers by Lee et al. [30].

This perspective opened new doors to analysis but did not explain the effectiveness of training neural networks using the widely used gradient descent approach. Jacot et al. [28] developed the solution to this by further generalizing infinite-width neural networks via a recursively defined kernel called the neural tangent kernel. The neural tangent kernel can be used to represent and analyze a given infinite-

width neural network during training with a specific depth, activation, and variance initialization. Future works utilized this kernel and expanded on the 1989 single layer results by showing that wide neural networks trained under gradient descent work as linear models and that empirically, finite networks also share those attributes [31]. In addition, further neural tangent kernel parameterizations were discovered for convolutional, recurrent, and graph neural network architectures [5, 3, 16]. Furthermore, the neural tangent kernel was also shown to generalize with neural networks that allow for regularization and gradient noise during training [12].

During the same time, Belkin et al. [7] empirically found that overfitted kernel methods display a similar phenomenon to overparameterized deep models where, despite reaching zero training loss, the test data would show good performance. Moreover, their work showed parallels between rectified linear unit (ReLU) activated neural networks and the Laplace kernel in the task of fitting random labels. Since then it has been shown theoretically that the Laplace and neural tangent kernels do in fact perform similarly to their neural network counterparts since both kernels share the same reproducing kernel Hilbert space $\mathcal{H}_k$ of predictions in the $\mathbb{S}^{d-1}$ unit d-sphere [11, 21]. This is consequential because the Laplace kernel has a very simple, well understood formulation whereas the neural tangent kernel has an unwieldy recursive formulation. In essence, the Laplace kernel can be used to analyze deep neural networks without the computational or theoretical difficulties of the neural tangent kernel.

However, this leaves some questions unanswered such as the practical equivalence of the kernels, the ability to find matching elements from the $\mathcal{H}_k$ of the kernels, whether the kernels share these similarities in the space of $\mathbb{R}^d$, and if the

Gaussian kernel, which comes from the same family as the Laplace, provides similar results. We attempt to provide answers by analyzing the neural tangent, Laplace, and Gaussian kernels via the framework of Gaussian process regression.

Chapter 2 defines the general data fitting problem, Gaussian processes for regression, and neural networks and their equivalence to Gaussian processes. Chapter 3 develops the theory for reproducing kernel Hilbert spaces and their relevance to the task of regression. Chapter 4 introduces the kernels used in our analysis and results showcasing the conditions for which the Laplace and neural tangent kernels can be made equal. Chapter 5 provides synthetic experiments for matching Gaussian process regression results (i.e. posterior means) between the various kernels, comparing the kernels in the space of $\mathbb{R}^d$, and improving regression results for the neural tangent kernel. Lastly, Chapter 6 showcases real world experiments of the kernels using the lessons learned from the previous chapters. We summarize the contributions of this thesis as follows:

- We find empirical evidence for the importance of the neural tangent kernel's bias parameter in equating it to the Laplace kernel.
- We derive the partial derivative of the neural tangent kernel with respect to its bias parameter which can be used to optimize the bias during regression fitting.
- We develop a method for matching the Laplace and neural tangent kernels' Gaussian process regression posteriors, which are elements of $\mathcal{H}_k$.
- We implement a `Python` programming language package called `scikit-ntk` which is an implementation of the neural tangent kernel that can be used directly with the popular `scikit-learn` machine learning toolkit.

Chapter 2

Regression

In this chapter we will introduce the basic data modeling problem and present two relevant modeling tools: Gaussian process regression and neural networks. We also present the relation between these two tools that allows practitioners to study the black box architecture of neural networks through the lens of Gaussian processes.

2.1 Data Fitting Problem

A single output data fitting problem begins with a set of n data points $\{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$ where $\mathbf{x}_i = [x_1, \ldots, x_d]^\top \in \mathcal{X} \subseteq \mathbb{R}^d$ is a single input vector and $y_i \in \mathbb{R}$ is a output value usually referred to as a target or response[1]. It is important to note that $\mathcal{X}$ can be a metric space but for our purpose it is sufficient to assume that it is a subset of the d-dimensional real space. In general, $\mathcal{X}$ may be determined by the type of data used or a transformation that is applied to the data (e.g. the unit d-sphere space $\mathbb{S}^{d-1} := \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = 1\}$).

We can form an $n \times d$ size matrix $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$ of n observations and d

[1] We will be using response to refer to y_i from here on out.

independent variables and a vector $\mathbf{y} = [y_1, \ldots, y_n]^\top$ of dependent outputs which combine into a set $(X, \mathbf{y})$ which we call a *training set*. The training set can be seen as a snapshot of some phenomenon that relates the training set inputs to response values via some function f:

$$y = f(\mathbf{x}) + \epsilon \tag{2.1}$$

where ϵ is some additive noise assumed to be independent of f and other noise terms. Depending on the problem, we may have $\epsilon = 0$ which indicates exact observations. Otherwise, we assume that

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{2.2}$$

indicating normally distributed noise with mean zero and fixed variance $\sigma^2 \in \mathbb{R}$.

The goal of the data fitting problem is to find a function f that best fits the training set while also best generalizing to unseen data of the phenomenon we are trying to model. This is done by minimizing an empirical loss functional between the true response values y_i and estimated values $f(\mathbf{x}_i)$:

$$f_{opt} = \arg\min_f \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) \right\}. \tag{2.3}$$

For regression, the loss function is usually absolute error $L(y_i, f(\mathbf{x}_i)) = |y_i - f(\mathbf{x}_i)|$, squared error $L(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2$, or some variation of either.

Without any restrictions or further assumptions placed on this process it is possible to perfectly interpolate (or "overfit") over the training set thus losing any generalization of unseen data. This is not a desirable outcome since the purpose of the problem is to capture additional information about the underlying phenomenon that generated the training set. As such, it is reasonable to place further assumptions on things including but not limited to the dataset (i.i.d., transformations), function properties (continuity, time dependence), model type (nonlinear regressor,

support vector machine), and model properties (parameterization type, regularization). These assumptions allow for the data fitting process to more productively use the training set to explain the underlying phenomenon. An example of a commonly used set of assumptions is a linear parametric model:

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d \\ &= \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}, \end{aligned} \tag{2.4}$$

where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]^\top$. With the squared error loss function, Equation (2.3) reduces to

$$\begin{aligned} f_{opt} &= \arg\min_f \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) \right\} \\ &= \arg\min_{\beta_0,\ldots,\beta_d} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \boldsymbol{\beta}^\top \mathbf{x}_i\right)^2 \right\}. \end{aligned} \tag{2.5}$$

However, this is a highly restrictive assumption which is not applicable in many cases.

On the other hand, regularization is a modification that is applicable to all models. Regularization helps prevent overfitting by placing a penalty on the model's objective function (Equation (2.3)):

$$f_{opt} = \arg\min_f \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + P_\lambda(f) \right\} \tag{2.6}$$

where $P_\lambda(\cdot) \in \mathbb{R}$ is a penalization term for a given predictive function f that is scaled by $\lambda \in \mathbb{R}$ and added to the summation. Effectively, $P_\lambda(\cdot)$ can be seen as a restriction on *smoothness* (i.e. properties of differentiability). The idea is to change the search space of the objective function from all possible models, including sophisticated ones that interpolate through the data, to simpler models that attempt to capture the underlying phenomenon instead of attaining zero training loss. Penalization for linear models (Equation (2.4)) includes ridge regression where $P_\lambda(f) = \lambda \sum_{j=1}^{d} \beta_j^2$

and lasso regression where $P_\lambda(f) = \lambda \sum_{j=1}^{d} |\beta_j|$. The regularization framework in Equation (2.6) is very general. For example, if instead we are searching through the space of all functions f with two continuous derivatives and utilizing the residual sum of squares loss, Equation (2.6) becomes

$$\begin{aligned} f_{opt} &= \arg\min_f \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + P_\lambda(f) \right\} \\ &= \arg\min_f \left\{ \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2 + \lambda \int f''(\mathbf{x}_i)^2 dx \right\} \end{aligned} \tag{2.7}$$

with the unique solution being a natural cubic spline [26, pg. 110].

The data fitting problem is one of compromises. While adding additional assumptions helps by reducing search time and/or limiting the search space, this still results in a complex task since we are going from one set of infinite functions to another. In addition, there may be fundamental issues with our training set since it is just a small sample of the overall phenomenon and thus sensitive to sampling methods and low signal to noise ratio. As for the model, we are limited by "no free lunch" theorem's [49, 48] implication that there is no best way to choose a model or even a best model for a specific task. There are many ways to approach the data fitting problem so our choices in models, regularization, data augmentation, etc. are then informed by both the task and quality of data. In this paper we focus on Gaussian processes which are a nonparametric regression method highly related to kernel ridge regression.

2.2 Gaussian Processes

At its simplest, a Gaussian process is a set of random variables indexed by time where any finite set composes a multivariate normal distribution. At the heart of

a Gaussian process is a positive definite covariance function.

Definition 2.2.1 (Positive Definite Function)**.** *Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric function. Given $n \in \mathbb{N}$, inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, and constants $c_1, \ldots, c_n \in \mathbb{R}$ where $\mathbf{c} = [c_1, \ldots, c_n]^\top$ we say that k is positive definite and forms a positive definite matrix K of all pairwise evaluations of the inputs on k if*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{c}^\top K \mathbf{c} \geq 0$$

Symmetry in the inputs is required in order for a covariance function to be real valued. Covariance functions are also referred to as kernel functions which will be expanded on in Chapters 3 and 4. From here we can define a Gaussian process from a functional context which gives us the ability to build regression models through them [47, 29].

Definition 2.2.2 (Gaussian Process)**.** *Let $f : \mathcal{X} \to \mathbb{R}$ be a real valued random function. We then say that $f(\mathbf{x})$ is a Gaussian process (GP) defined by mean function $m : \mathcal{X} \to \mathbb{R}$ and positive definite covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ if for any finite set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathcal{X}$ of any size $n \in \mathbb{N}$ we have the random vector $\mathbf{f}_X = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^\top \in \mathbb{R}^n$ such that*

$$\mathbf{f}_X \sim \mathcal{MVN}(\mathbf{m}_X, K_{XX}) \tag{2.8}$$

where $\mathbf{m}_X = [m(\mathbf{x}_1), \ldots, m(\mathbf{x}_n)]^\top \in \mathbb{R}^n$ is the mean vector and $K_{XX} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n \in \mathbb{R}^{n \times n}$ is the covariance matrix. We define m and k for inputs $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ as follows:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{z}) &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{z}) - m(\mathbf{z}))] \end{aligned} \tag{2.9}$$

and denote such a process as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{z})). \tag{2.10}$$

The implication of this definition is if a GP exists, then so does a corresponding m and k. However, it is not necessary to begin with a process as shown via the Kolmogorov Existence Theorem [17, Theorem 12.1.3, pg. 443].

Theorem 2.2.1 (Kolmogorov Existence Theorem for Gaussian Processes)**.** *Let $\mathcal{X}$ be any set such that there exists a function $m : \mathcal{X} \to \mathbb{R}$ and a positive definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Then there exists a GP on $\mathcal{X}$ with mean function m and covariance function k.*

As a result, there is a one-to-one relationship between a GP and its mean and covariance functions. Properties such as continuity, periodicity, and differentiability are tied to the mean and covariance functions which allows for an open view of the inner workings of a model built on a GP. Furthermore, it allows for users to insert prior knowledge into the data fitting process making GPs a powerful Bayesian modeling tool.

To apply a GP to the task of regression it is necessary to build a joint distribution over a training set and a previously ignored *test set*, $[\mathbf{x}_{*_1}, \ldots, \mathbf{x}_{*_m}]^\top = X_* \subseteq \mathbb{R}^{m \times d}$ where each $\mathbf{x}_{*_i} \in \mathcal{X} \subseteq \mathbb{R}^d$. The test set mirrors the training set in Section 2.1 with the only differences being that the test set is usually disjoint from the training set and potentially contains a different number of observations compared to the training set ($m \neq n$). The test set will be used analyze unseen data $\mathbf{f}_* \in \mathbb{R}^m$ through a posterior distribution conditional on the known $\mathbf{y}$. First, we choose a covariance function k and build covariance matrices K to inform our data fitting task from Equation 2.1 following the same noise assumptions as in Equation 2.2. We can then build the following joint distribution of the training data and testing

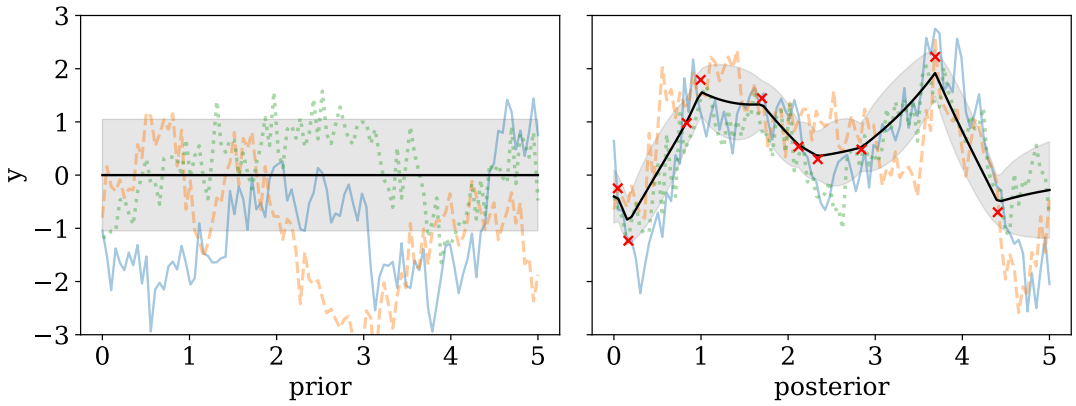

Figure 2.1: Sample paths of the Laplace covariance function from a GP prior and posterior. The solid line represents $m(\mathbf{x})$ and $\bar{\mathbf{f}}_*$ in the left and right panel respectively. The shaded bands represent $\text{cov}(\mathbf{f}_*)$. The red $\mathbf{x}$'s in the right panel are the training points.

data:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{MVN}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} K_{XX} + \sigma^2 I_n & K_{XX_*} \\ K_{X_*X} & K_{X_*X_*} \end{bmatrix} \right) \tag{2.11}$$

where

$$\begin{aligned} K_{XX} &= \left[k(\mathbf{x}_i, \mathbf{x}_j)\right]_{i,j=1}^{n} \\ K_{X_*X_*} &= \left[k(\mathbf{x}_{*_i}, \mathbf{x}_{*_j})\right]_{i,j=1}^{m} \\ K_{XX_*} &= \left[k(\mathbf{x}_i, \mathbf{x}_{*_j})\right]_{i,j=1}^{i=n,j=m} \\ K_{X_*X} &= K_{XX_*}^{\top} \end{aligned} \tag{2.12}$$

Equation (2.11) forms the prior distribution for our GP regressor. To get the posterior distribution, we condition $\mathbf{f}_*$ on the known data which by properties of $\mathcal{MVN}$ is itself $\mathcal{MVN}$ [25]:

$$\begin{aligned} \mathbf{f}_* | X, y, X_* &\sim \mathcal{MVN}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)) \\ \bar{\mathbf{f}}_* &= K_{X_*X}\left[K_{XX} + \sigma^2 I_n\right]^{-1}\mathbf{y} \\ \text{cov}(\mathbf{f}_*) &= K_{X_*X_*} - K_{X_*X}\left[K_{XX} + \sigma^2 I_n\right]^{-1} K_{XX_*} \end{aligned} \tag{2.13}$$

Viewing the procedure from the practical modeling perspective, training the GP regressor can be thought of as pre-computation of $[K_{XX} + \sigma^2 I_n]^{-1}$ while prediction computes the remainder of the values involving the test set X_*. In addition, with knowledge of the full distribution we can sample the prior and posterior distributions to view the bounds and functions generated by the GP as seen in Figure 2.1. The matrix operations involved in calculating the posterior are $O(n^3)$ and thus require approximate methods for large scale datasets. Rasmussen and Williams calculate the inverse using Cholesky decomposition and provide a number of approximate methods for computation [47, pg. 19, pg. 171].

2.3 Neural Networks

A neural network is one of the key tools used in machine learning. They provide a way to model highly nonlinear phenomena and have been found to generalize exceptionally to a variety of complex tasks. One drawback of neural networks is the difficulty of analyzing their black box nature. Traditional tools for inference and uncertainty estimation cannot be easily applied to them; however, there are equivalences that aid in this task. To start, we adopt notation from [24] and define the *neuron* which is the basic building block of a neural network.

Definition 2.3.1 (Neuron)**.** *Let* $\mathbf{x} = [x_1, \ldots, x_d]^\top \in \mathcal{X}$ *be an input vector with* $d \in \mathbb{N}$*. Then the k-th* ***neuron*** a_k *is defined as:*

$$a_k = \sigma\left(\sum_{i=1}^{d} w_{k_i} x_i + b_k\right) = \sigma\left(\mathbf{w}_k^\top \mathbf{x} + b_k\right) \tag{2.14}$$

where $\mathbf{w}_k = [w_{k_1}, \ldots, w_{k_d}]^\top$ *is a vector of weights and* $b_k \in \mathbb{R}$ *is the bias value.* σ *is an* ***activation function*** *which is a fixed transformation that is used to inject nonlinear behavior to the resulting outputs.*

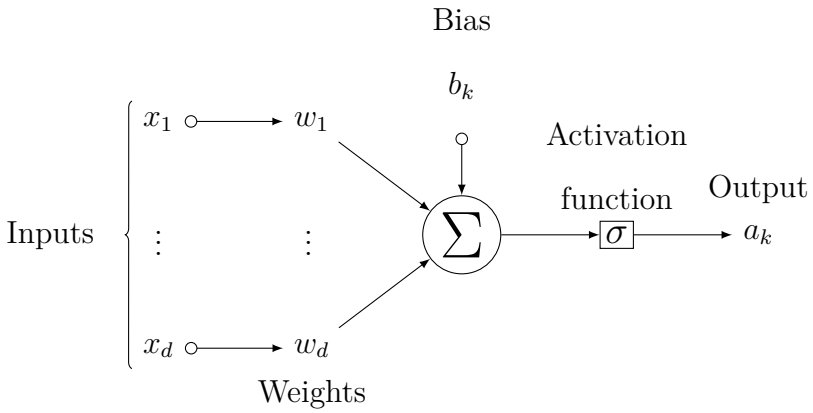

Figure 2.2: A visualization of a neuron courtesy of [32].

Nonlinear activations allow neural networks to model complex behavior that traditional linear models cannot. A combination of $l \in \mathbb{N}$ neurons into a vector is interpreted as l-width layer. By combining many layers of varying widths, we can build the simplest architecture of a neural network called the *multilayer perceptron*.

Definition 2.3.2 (Multilayer Perceptron)**.** *Let* $\mathbf{x} \in \mathcal{X}$ *and* $h^{(i)}$ *be the* i*-th* ***hidden layer*** *where* $i \in \{1, \dots, L\}$ *represents a finite amount of layers with* $L \in \mathbb{N}$*. Then a* ***multilayer perceptron architecture*** *is defined as follows:*

$$\begin{aligned} h^{(1)} &= \sigma^{(1)}\left(\boldsymbol{W}^{(1)}\mathbf{x} + \beta^{(1)}\right) \\ h^{(2)} &= \sigma^{(2)}\left(\boldsymbol{W}^{(2)}h^{(1)} + \beta^{(2)}\right) \\ &\vdots \\ h^{(L)} &= \sigma^{(L)}\left(\boldsymbol{W}^{(L)}h^{(L-1)} + \beta^{(L)}\right) \\ f(\mathbf{x};\theta) &= \boldsymbol{w}h^{(L)} + \beta^{(L+1)} \end{aligned} \tag{2.15}$$

where $\boldsymbol{W}^{(i)}$ *are weight matrices for each layer,* $\boldsymbol{w}$ *is the vector of weights for the desired output dimension of the function* f*,* $\beta^{(i)}$ *is the bias vector, and* $\sigma^{(i)}$ *is the layer dependant activation function. The 1st dimension of* $\boldsymbol{W}^{(i)}$ *repre-*

sents the number of neurons (i.e. the width of the hidden layer $h^{(i)}$) whilst the 2nd dimension is determined by the dimension of $h^{(i-1)}$. $h^{(1)}$ is called the ***input layer*** *while $f(\mathbf{x};\theta)$ is the* ***output layer*** *where θ is a set of all parameters $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{w}, \beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(L+1)}$ in the network. The resulting network is considered to be $L+1$ layers deep.*

A multilayer perceptron can be modified to include transformation layers (e.g. a *dropout layer* which randomly drops weights from the previous layer), differing activation functions per layer, differing layer widths, etc. The training procedure for such networks follows the data fitting procedure in Equation (2.3)

$$\theta_{opt} = \arg\min_{\theta} \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i;\theta)) \right\}, \tag{2.16}$$

so the optimized network is $f_{opt}(\mathbf{x}) := f(\mathbf{x};\theta_{opt})$ with optimal parameters θ_{opt} found during the minimization procedure over the unfixed network parameters θ. The minimization is done by computing the gradient of the loss function with respect to

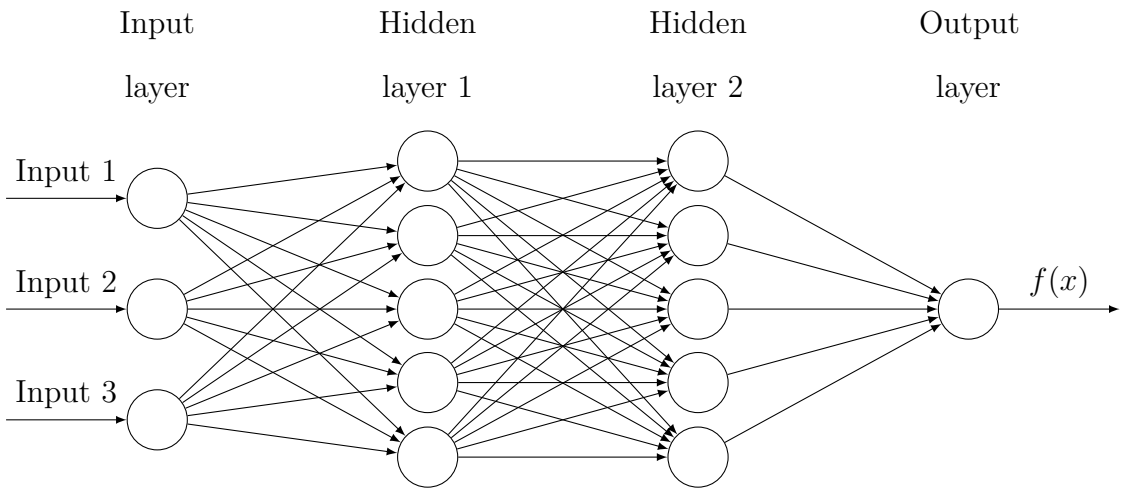

Figure 2.3: A visualization of a 2 layer fully connected neural network courtesy of [32].

θ. This is done efficiently via *back-propagation* [38] which uses information provided by the outputs $f(\cdot)$ in a single gradient step and computes the gradient of the loss over the parameters starting from the output layer and ending at the input layer. On the other hand, prediction is done by *forward propagation* which uses an input to calculate an output f_* on the optimized network through a forward pass using Equation 2.15.

The specific neural networks we consider in this thesis are *infinite width fully connected rectified linear unit (ReLU) activated neural networks*. *Infinite width* refers to the size of all hidden layers. *Fully connected* means that every individual neuron in every layer $i \in \{1, \ldots, L\}$ sends its output to every neuron in layer $i+1$. Lastly, the *ReLU activation function* is defined as follows for $x \in \mathbb{R}$:

$$\begin{gathered} \sigma(x) = \max(0, x) \\ \text{where } \sigma(\cdot) : \mathbb{R} \to \mathbb{R}. \end{gathered} \tag{2.17}$$

Neal [35] showed that neural networks in the infinite width limit converge to a GP while Williams [46] developed the computation of the covariance function. Let us assume a single hidden layer network with weights w_{k_i} and biases b_k, each independent and identically distributed (iid) normal with mean 0 and respective variances σ_w^2 and σ_b^2. The gist of this equivalence is that each neuron in the hidden layer then becomes iid normal with mean zero and finite variance. By the Central Limit Theorem, as the width H of the hidden layer tends to infinity, it also forms a normal distribution. By following Definition 2.2.2, we form a stochastic process over the indexed set of n inputs and corresponding outputs which form a multivariate

normal distribution with mean zero and finite covariance for all inputs:

$$\begin{aligned} m(\mathbf{x}) &= 0 \\ k(\mathbf{x}, \mathbf{z}) &= \sigma_b^2 + \sigma_w^2 H \mathbb{E}[h_j(\mathbf{x}) h_j(\mathbf{z})] \\ &= \sigma_b^2 + \omega_w^2 \mathbb{E}[h_j(\mathbf{x}) h_j(\mathbf{z})] \\ \text{where } \sigma_w &= \omega_w H^{-1/2} \text{ and } j \in \{1, \ldots, L\}. \end{aligned} \tag{2.18}$$

This single layer infinite width network satisfies the definition of a GP. This can be further extended to networks of multiple layers by applying the same procedure by over all network weights and biases. It should be noted that this formulation describes an untrained network at initialization.

Although somewhat impractical in practice, infinite width networks provide a way to study the properties of neural networks. Yang [50] expanded on this and showed that the GP and neural network equivalence applies to all modern architectures. The result described in this section is related to the neural tangent kernel [28] which can be used as a GP covariance function to implement an infinite width neural network which is further discussed in Chapter 4.

Chapter 3

Reproducing Kernel Hilbert Spaces

Kernels exist in a variety of contexts throughout statistics, probability, and mathematics. They can be thought of as the kernel of a probability density (mass) function used in kernel density estimation, a positive definite kernel used in a variety of kernel methods, a null space in linear algebra, an integral transform in calculus, and a reproducing kernel considered in functional analysis.

Although kernels span a large variety of subjects, we are going to focus on them as they pertain to functional analysis via *reproducing kernel Hilbert spaces*. In this chapter, we will define kernels, their reproducing kernel Hilbert spaces, and properties of these spaces in the context of data fitting.

3.1 Reproducing Kernels

A *kernel* is a class of functions that map two values to the real line:

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \tag{3.1}$$

In our context, we are interested in kernels that are positive definite and symmetric because this allows them to be real valued and used in methods such as kernel ridge regression, support vector machines, and Gaussian processes. As mentioned in Chapter 2, a positive definite kernel and covariance function are one and the same in the context of GPs. In order to further develop kernels, we need to work within a general vector space, namely a Hilbert space.

Definition 3.1.1 (Hilbert space)**.** *A* ***Hilbert space*** $\mathcal{H}$ *is a vector space with the following properties:*

1. $\mathcal{H}$ *contains an inner product* $\langle \cdot, \cdot \rangle_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ *which induces the norm* $\langle x, x \rangle_{\mathcal{H}} = ||x||^2_{\mathcal{H}}$ *for* $x \in \mathcal{H}$*,*

2. $\mathcal{H}$ *is complete (i.e. every Cauchy sequence in* $\mathcal{H}$ *converges to some element of* $\mathcal{H}$*).*

Although our definition of a Hilbert space constitutes a real valued inner product, it can be defined over more general spaces (see Axler [6, Chapter 8 pg. 211]). In our case it is sufficient to work in the space of $\mathbb{R}$. We are now ready to define a key space regarding kernels.

Definition 3.1.2 (Reproducing Kernel Hilbert Space)**.** *Let* $\mathcal{H}_k$ *be a Hilbert space of real functions defined on* $\mathcal{X}$ *and norm* $\|f\|^2_{\mathcal{H}_k} = \langle f, f \rangle_{\mathcal{H}_k}$ *for* $f \in \mathcal{H}_k$*. The function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *is called a reproducing kernel of* $\mathcal{H}_k$ *if:*

1. *For all* $\mathbf{x} \in \mathcal{X}$, $k(\cdot, \mathbf{x}) \in \mathcal{H}_k$,

2. *For all* $\mathbf{x} \in \mathcal{X}$ *and for all* $f \in \mathcal{H}_k$, $\langle f(\cdot), k(\cdot, \mathbf{x}) \rangle = f(\mathbf{x})$.

Property 2 in the definition above is called the *reproducing property*. Reproducing kernel Hilbert spaces (RKHSs) do not require for kernel functions to be positive definite explicitly; however, positive definite kernels have a nice property regarding RKHS:

Theorem 3.1.1 (Moore-Aronszajn Theorem [4])**.** *For a positive definite function* $k(\cdot, \mathbf{x})$ *on* $\mathcal{X} \times \mathcal{X}$ *there exists only one RKHS.*

This theorem guarantees that for any symmetric and positive definite kernel, there exists a unique RKHS and vice-versa. Using this idea we can build an RKHS from a positive definite kernel alone. To illustrate this given a symmetric and positive definite kernel on $\mathcal{X}$, we begin by defining a *pre-Hilbert space* $\mathcal{H}_0$.

$$\begin{aligned}\mathcal{H}_0 &:= \operatorname{span}\{k(\cdot, \mathbf{x}) : \mathbf{x} \in \mathcal{X}\} \\ &= \left\{ f(\cdot) = \sum_{i=1}^{n} c_i k(\cdot, \mathbf{x}_i) : n \in \mathbb{N}, c_1, \ldots, c_n \in \mathbb{R}, \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X} \right\}. \quad (3.2)\end{aligned}$$

with a valid inner product (see [8, pg. 20]) defined for any $f := \sum_{i=1}^{n} a_i k(\cdot, \mathbf{x}_i)$ and $g := \sum_{j=1}^{m} b_j k(\cdot, \mathbf{x}_j)$

$$\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i=1}^{n} \sum_{j=1}^{m} a_i b_j k(\mathbf{x}_i, \mathbf{x}_j) \quad (3.3)$$

and a norm induced by the inner product

$$\langle f, f \rangle_{\mathcal{H}_0} = ||f||^2_{\mathcal{H}_0} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{a}^\top K \mathbf{a} \quad (3.4)$$

where $\mathbf{a} = [a_1, \ldots, a_n]^\top$ is a vector and $K = [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^{n}$ is a matrix. To form the RKHS $\mathcal{H}_k$ we must bundle the rest of the possible elements within $\mathcal{H}_0$ by defining

its closure with respect to $||\cdot||_{\mathcal{H}_0}$

$$\mathcal{H}_k = \Bigg\{ f(\cdot) = \sum_{i=1}^{\infty} c_i k(\cdot, \mathbf{x}_i) : n \in \mathbb{N}, c_1, c_2, \cdots \in \mathbb{R}, \mathbf{x}_1, \mathbf{x}_2, \cdots \in \mathcal{X} \text{ where } \|f\|_{\mathcal{H}_k}^2 := \lim_{n\to\infty} \left\| \sum_{i=1}^{n} c_i k(\cdot, \mathbf{x}_i) \right\|_{\mathcal{H}_0}^2 = \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) < \infty \Bigg\}, \quad (3.5)$$

thus completing our construction. For more details refer to Kanagawa et al. [29, pg. 11] and Berlinet and Thomas-Agnan [8, Theorem 3 pg. 19-21].

3.2 Mercer Representation

There is another way to represent RKHSs that provides a further route of analyzing kernels via spectral decomposition. This will require a deeper dive into functional analysis for which further details can be found in Axler [6, Chapter 10 pg. 280], Steinwart and Christmann [40, Appendix A.5 pg 497], and Rudin [37].

We begin with a measurable space $\mathcal{X}$ with μ being its finite Borel measure with $\mathcal{X}$ being its support. We then consider the space of square-integrable functions $L_2(\mathcal{X}, \mu)$ that exist on $\mathcal{X}$ with respect to metric μ and the kernel k on $\mathcal{X}$. Then we define an integral operator $T_k : L_2(\mathcal{X}, \mu) \to L_2(\mathcal{X}, \mu)$ such that for $f \in L_2(\mathcal{X}, \mu)$

$$T_k f := \int_{\mathcal{X}} k(\cdot, \mathbf{x}) f(\mathbf{x}) d\mu(\mathbf{x}) \quad (3.6)$$

which is compact, positive, and self-adjoint [40, Theorem 4.27]. As a result, we can apply the Spectral Theorem [40, Theorem A.5.13] which shows that there exists $(\phi_i, \lambda_i)_{i\in I}$ where $(\phi_i)_{i\in I} \subset L_2(\mathcal{X}, \mu)$ is the orthonormal system of countable eigenfunctions and $(\lambda_i)_{i\in I}$ is the corresponding family of eigenvalues for indices $I \subseteq \mathbb{N}$ such that for strictly positive eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots > 0$:

$$T_k f = \sum_{i\in I} \lambda_i \langle \phi_i, f \rangle_{L_2} \phi_i. \quad (3.7)$$

With this we can now develop an expansion of kernels via orthonormal functions:

Theorem 3.2.1 (Mercer's theorem [40, 33]). *Let $\mathcal{X}$ be a compact metric space, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a continuous kernel, and μ be a finite Borel measure with support on $\mathcal{X}$. Then for $(\phi_i, \lambda_i)_{i \in I}$ and inputs $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ we have*

$$k(\mathbf{x}, \mathbf{z}) = \sum_{i \in I} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{z}), \tag{3.8}$$

where the convergence is absolute and uniform over the inputs.

Lastly, we can redevelop RKHSs using orthonormal eigenfunctions to represent kernels:

Theorem 3.2.2 (Mercer Representation of RKHSs [40]). *Let $\mathcal{X}$ be a compact metric space, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a continuous kernel, μ be a finite Borel measure with support on $\mathcal{X}$ and $(\phi_i, \lambda_i)_{i \in I}$. We then define the RKHS $\mathcal{H}_k$*

$$\mathcal{H}_k := \left\{ f := \sum_{i \in I} a_i \sqrt{\lambda_i} \phi_i : ||f||^2_{\mathcal{H}_k} := \sum_{i \in I} a_i^2 < \infty \right\}, \tag{3.9}$$

with an inner product defined for any $f := \sum_{i \in I} a_i \sqrt{\lambda_i} \phi_i$ and $g := \sum_{i \in I} b_i \sqrt{\lambda_i} \phi_i$

$$\langle f, g \rangle_{\mathcal{H}_k} = \sum_{i \in I} a_i b_i \tag{3.10}$$

where $(\sqrt{\lambda_i} \phi_i)_{i \in I}$ is an orthonormal basis of $\mathcal{H}_k$ and operator $T_k^{1/2} : L_2(\mathcal{X}, \mu) \to H$ is an isometric isomorphism. The $\mathcal{H}_k$ in Equation (3.9) is the same as in Equation (3.5).

As a result, Theorem 3.2.1 gives a general way of analyzing kernels via their resulting eigenfunctions and corresponding eigenvalues while Theorem 3.2.2 connects this analysis back to RKHSs. One way to do this is by observing and comparing the rate of eigenvalue decay between two separate kernels [21]. Furthermore, both

theorems indicate that an RKHS $\mathcal{H}_k$ is a subset of the $L_2(\mathcal{X}, \mu)$ space. Because of this, the Mercer notion of kernel and RKHS relies on a choice of measure μ to work; however, as shown in Section 3.1, both kernel and RKHS are independent of any measure. As a result, regardless of the choice of measure we still end up with the same RKHS [29]. The only difference here is that the new measure will result in a new eigensystem.

3.3 Representer Theorem

Let us backtrack to the data fitting problem established in Section 2.1. Consider the dataset $\{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$. From such a setup, it is natural to wish to determine whether there exists a function f that is generating some sort of signal in our dataset. We can do so by regression using a variety of methods to accomplish this task; however, in general we wish to minimize an empirical risk functional [39].

Theorem 3.3.1 (Representer Theorem)**.** *Let $\{\mathbf{x}_i\}_{i=1}^n \subset \mathcal{X}$ and $\{y_i\}_{i=1}^n \subset \mathbb{R}$. Consider functions defined as $f : \mathcal{X} \to \mathbb{R}$ defined in a RKHS $\mathcal{H}_k$ where k represents a kernel. Now consider the following empirical risk minimization problem:*

$$f_{opt} = \arg\min_{f \in \mathcal{H}_k} \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda ||f||_k^2 \right\} \tag{3.11}$$

$L : \mathbb{R}^2 \to \mathbb{R}$ represents the data fitting term (usually referred to as a loss function) and $\lambda ||f||_k^2$ is a regularization term with factor $\lambda \geq 0$. The minimizer $f_{opt} \in \mathcal{H}_k$ can be represented pointwise as follows:

$$f_{opt}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = K_{\mathbf{x}X}\boldsymbol{\alpha}, \qquad \mathbf{x} \in \mathcal{X} \tag{3.12}$$

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]^\top = (K_{XX} + n\lambda I_n)^{-1}\mathbf{y}$.

As a result, the potentially infinite dimensional problem of finding f_{opt} only depends on a finite sum and on the data in question. In addition, it is guaranteed to exist and is unique. It also gives us a generic way of looking various regression tools from the more abstract perspective of RKHSs. This includes deep neural networks as shown by Unser [43]. It is important to note that Equation (3.12) shows that the minimizer belongs to the RKHS regardless of the dataset used.

Regularization is one form of the representer theorem. By applying Theorem 3.3.1 and slightly adjusting the regularized regression problem presented in Equation (2.7), we can view the penalty term as one determined by a RKHS endowed with a kernel k:

$$f_{opt} = \underset{f \in \mathcal{H}_k}{\arg\min} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2 + \lambda ||f||^2_{\mathcal{H}_k} \right\} \tag{3.13}$$

with a unique solution

$$K_{\mathbf{x}X}(K_{XX} + n\lambda I_n)^{-1}\mathbf{y}. \tag{3.14}$$

This form of regression is called *kernel ridge regression* (KRR) [29] and only depends on functions defined by the kernel k and its corresponding RKHS $\mathcal{H}_k$. Alternatively, a GP regressor defined by the same kernel k has a unique posterior mean function computed by the marginal log likelihood

$$K_{\mathbf{x}X}(K_{XX} + \sigma^2 I_n)^{-1}\mathbf{y}. \tag{3.15}$$

Given this, Kanagawa et al. [29] concisely summarize the following known result:

Proposition 1. *If $\sigma^2 = n\lambda$ then the GP posterior mean function and the KRR solution are the same.*

This result ties a GP posterior mean to a KRR solution and the resulting RKHS $\mathcal{H}_k$. Although any valid kernel is tied to an RKHS, our definition of a GP regressor

does not directly rely on that connection. By relating GP regression to KRR we can better analyze kernels via the GP framework while having confidence in the underlying RKHS theory. In particular interest is attempting to find kernel hyperparameters of a GP for the Laplace and neural tangent kernels so that the resulting posterior means are the same. Through this matching of posterior means (and underlying hyperparameters) we can gain insight to the underlying RKHS of each kernel which we explore later in Chapter 5.

Chapter 4

Types of Kernels and their Equivalences

Having developed the key theory behind kernels we turn to define the kernels used in this thesis. Our empirical analysis will focus on the Laplace, Gaussian, and neural tangent kernels. We define the kernels over their respective inputs $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ and set of parameters θ; however, when it is clear from context we may drop the inputs so that

$$k(\theta) := k(\mathbf{x}, \mathbf{z}; \theta). \tag{4.1}$$

Moving forward it should be noted that by the reproducing property of a given kernel k, the elements of its RKHS can be represented as

$$f(\cdot) = \sum_{i=1}^{\infty} c_i k(\cdot, \mathbf{x}_i) \text{ where } c_i \in \mathbb{R}, \mathbf{x}_i \in \mathcal{X} \tag{4.2}$$

which shows that RKHS member functions inherit properties that are dependent on the kernel.

4.1 Matérn Class of Kernels

Definition 4.1.1 (Matérn Kernel)**.** *Let* $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$ *be inputs,* $\ell > 0$ *be the length-scale parameter, and* $\nu > 0$ *be the smoothness parameter such that*

$$k_{Mat}(\mathbf{x}, \mathbf{z}; \nu, \ell) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{z}\| \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{z}\| \right) \tag{4.3}$$

where $||\cdot||$ *is the* L_2*-norm,* Γ *is the Gamma function, and* K_ν *is a modified Bessel function of the 2nd kind [1, Section 9.6].*

The Matérn class of kernels defines a set of functions dependent on their smoothness ν. By varying ν we define our two kernels of interest. As $\nu \to \infty$, the Matérn kernel becomes equivalent to the well known *Gaussian kernel*[2]:

$$k_{Gaus}(\mathbf{x}, \mathbf{z}; \ell) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\ell^2} \right). \tag{4.4}$$

By setting $\nu = \frac{1}{2}$, the Matérn kernel becomes equivalent to the *Laplace kernel*:

$$k_{Lap}(\mathbf{x}, \mathbf{z}; \ell) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|}{\ell^2} \right). \tag{4.5}$$

Both kernels are very similarly defined and as such fall under the umbrella of exponential class of kernels as well. Despite their similar forms, the two kernels have some stark differences. The Gaussian kernel produces an RKHS of continuous, infinitely differentiable functions for all possible length-scales [47, Section 4.2.1], and its eigenvalues decay exponentially [34]. In contrast, elements of the Laplace kernel's RKHS are continuous, nowhere differentiable for all possible length-scales [47, Section 4.2.1], and its eigenvalues decay polynomially [21]. A process defined by the Laplace kernel is called an *Ornstein–Uhlenbeck process* [42] which was shown to describe the velocity of a Brownian particle.

[2] Also known as the *radial basis function* or the *squared exponential kernel*.

4.2 Neural Tangent Kernel

As mentioned in Section 2.3, infinite width neural networks behave as GPs and can be studied through the function space. One hindrance with this approach is that in order to develop a GP via the method outlined previously, one must determine the covariance kernel of the GP using all the parameters of that architecture. Cho and Saul [13] found that instead of this approach, it is possible to use the *arc-cosine kernel* to build various finite network architectures via a single kernel. Our focus is on the *neural tangent kernel* defined by Jacot et al. [28] which describes the behavior of a neural network trained by gradient descent.

Definition 4.2.1 (Finite Neural Tangent Kernel)**.** *Let $f(\cdot;\theta)$ be a neural network with finite number of parameters θ. Then, the* ***finite neural tangent kernel*** *for the neural network and inputs $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$ is defined as a sum containing tensor products of partials with respect to the p-th parameter of f:*

$$k_{NTK}(\mathbf{x}, \mathbf{z}; \theta) = \sum_{p=1}^{P} \partial_{\theta_p} f(\mathbf{x}; \theta) \otimes \partial_{\theta_p} f(\mathbf{z}; \theta), \tag{4.6}$$

where P is the total number of network parameters and $\mathbb{S}^{d-1}$ is the unit d-sphere space defined as

$$\mathbb{S}^{d-1} := \left\{ \mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = 1 \right\}. \tag{4.7}$$

Definition 4.2.1 refers to the neural tangent kernel for finite width and depth neural networks. However, this kernel can also be used with kernel methods to represent infinitely wide neural network architectures through an explicit recursively defined kernel. The neural network and GP equivalence discussed in Section 2.3 makes this possible by making the network in question independent of the parameters θ and instead dependent on the resulting GP. The details of this are further

discussed in the appendix of Jacot et al. [28]. In our definitions we adopt their notation alongside notation from Bietti and Mairal [9].

Definition 4.2.2 (Infinite Neural Tangent Kernel)**.** *Given a fully connected infinite width network with $L+1$ layers, $\beta \geq 0$ bias, and with $h \in \{1, \ldots, L\}$ we define the deterministic* ***infinite neural tangent kernel*** *recursively for inputs $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$ as*

$$\begin{aligned} k_{NTK}(\mathbf{x}, \mathbf{z}; L+1, \beta) &:= \Theta^{(L)}(\mathbf{x}, \mathbf{z}) \\ \Theta^{(h)}(\mathbf{x}, \mathbf{z}) &= \Theta^{(h-1)}(\mathbf{x}, \mathbf{z})\dot{\Sigma}^{(h)}(\mathbf{x}, \mathbf{z}) + \Sigma^{(h)}(\mathbf{x}, \mathbf{z}) + \beta^2, \end{aligned} \tag{4.8}$$

with the base case

$$\begin{aligned} \Sigma^{(0)}(\mathbf{x}, \mathbf{z}) &= \mathbf{x}^\top \mathbf{z} \\ \Theta^{(0)}(\mathbf{x}, \mathbf{z}) &= \Sigma^{(0)}(\mathbf{x}, \mathbf{z}) + \beta^2 \end{aligned} \tag{4.9}$$

and components

$$\begin{aligned} \Sigma^{(h)}(\mathbf{x}, \mathbf{z}) &= \frac{c_\sigma}{2} \kappa_1\left(\lambda^{h-1}\right) \sqrt{\Sigma^{(h-1)}(\mathbf{x}, \mathbf{x})\Sigma^{(h-1)}(\mathbf{z}, \mathbf{z})} \\ \dot{\Sigma}^{(h)}(\mathbf{x}, \mathbf{z}) &= \frac{c_\sigma}{2} \kappa_0\left(\lambda^{h-1}\right), \end{aligned} \tag{4.10}$$

where $c_\sigma = 2$ for ReLU activated networks. We then define the cosine normalization [22]:

$$\lambda^{(h-1)}(\mathbf{x}, \mathbf{z}) = \frac{\Sigma^{(h-1)}(\mathbf{x}, \mathbf{z})}{\sqrt{\Sigma^{(h-1)}(\mathbf{x}, \mathbf{x})\Sigma^{(h-1)}(\mathbf{z}, \mathbf{z})}}, \tag{4.11}$$

such that $|\lambda^{(h-1)}| \leq 1$. Lastly, we define the arc-cosine kernels of degree 0 and 1 respectively [13]:

$$\begin{aligned} \kappa_0(u) &= \frac{1}{\pi}\left(\pi - \arccos(u)\right) \\ \kappa_1(u) &= \frac{1}{\pi}\left(u(\pi - \arccos(u)) + \sqrt{1-u^2}\right). \end{aligned} \tag{4.12}$$

In this thesis, we refer to the infinite neural tangent kernel as the NTK for brevity. It should be noted that Definitions 4.2.1 and 4.2.2 are also valid in the space of $\mathbb{R}^d$. Geifman et al. [21] further define the normalized kernel $\frac{1}{L+1} k_{NTK}(\mathbf{x}, \mathbf{z}; L+$

$1, \beta)$ when $\beta = 0$. We improve on this by empirically finding the more general case of normalization. Let $\beta \geq 0$, then it can be shown that

$$\ddot{k}_{NTK}(\mathbf{x}, \mathbf{x}; L+1, \beta) = \frac{1}{(L+1)(\beta^2+1)} k_{NTK}(\mathbf{x}, \mathbf{x}; L+1, \beta) = 1 \tag{4.13}$$

for all $\mathbf{x} \in \mathbb{R}^d$. We will be using this normalized form for the remainder of our work.

This recursive formalization depends entirely on the depth of the network and bias β. In practice, we want to be able to find the optimal β parameter for the given regression problem. As such, this requires the gradient of $\ddot{k}_{NTK}$ which we define independent of input for a given depth $L+1$ and bias β

$$\frac{\partial}{\partial \beta} \ddot{k}_{NTK}(\mathbf{x}, \mathbf{z}) = \frac{1}{(L+1)(\beta^2+1)} \left(\frac{\partial}{\partial \beta} k_{NTK}(\mathbf{x}, \mathbf{z}) - \frac{2\beta}{\beta^2+1} k_{NTK}(\mathbf{x}, \mathbf{z}) \right). \tag{4.14}$$

The full derivation of the gradient can be found in Appendix A. From here, we will utilize the notation $D = L + 1$ to refer to the *depth* or the number of layers of a given network defined by the NTK.

4.3 RKHS Inclusion

Given two kernels, it is natural to compare the space of functions that they are capable of producing and seeing if there is any overlap. This is made possible by a consequence of Theorem 3.1.1 which allows us to determine if one RKHS is a subset of another. In regards to this thesis, we are interested in determining equality between the RKHSs of two kernels and thus equality of the kernels themselves.

Theorem 4.3.1 (Subset Inclusion [8], p. 30)**.** *Let k_1 and k_2 be continuous positive definite kernels on $\mathcal{X}_1 \times \mathcal{X}_1$ and $\mathcal{X}_2 \times \mathcal{X}_2$ respectively with $\mathcal{H}_{k_1}$ and $\mathcal{H}_{k_2}$ denoting*

their respective RKHS. Then, $\mathcal{H}_{k_1} \subset \mathcal{H}_{k_2}$ if and only if there exists a constant B such that $B^2 k_2 - k_1$ is a positive definite kernel.

While this is a powerful theorem, in practice it presents a problem when trying to take into consideration a kernel's set of parameters in relation to an RKHS. In fact, a parameter can have a great effect on the functions that belong to the RKHS. This is illustrated in Walder [44, Lemma 3.1.2, pg. 34] where we have two Gaussian kernels defined by length-scale parameters $\ell_1, \ell_2 \in \mathbb{R}$. We let $k_{Gaus}(\cdot, \cdot; \ell)$ be the general Gaussian reproducing kernel defined by some parameter $\ell \in \mathbb{R}$ that defines an RKHS $\mathcal{H}_k$. If $\ell_1 > \frac{1}{2}\ell$ and $\ell_2 > \frac{1}{2}\ell$, then by utilizing the inner product of $\mathcal{H}_k$ we can create a new reproducing kernel defined in that space for some inputs $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$

$$\langle k_{Gaus}(\cdot, \mathbf{x}; \ell_1), k_{Gaus}(\cdot, \mathbf{z}; \ell_2) \rangle_{\mathcal{H}_k} = k_{Gaus}(\mathbf{x}, \mathbf{z}; \ell_1 + \ell_2 - \ell) \tag{4.15}$$

which shows that there exists a new kernel function with length-scale $\ell_1 + \ell_2 - \ell$ that is a member of $\mathcal{H}_k$. However, if one of the inequalities is not satisfied, then the kernel corresponding to that length-scale is *not* in the RKHS.

Another way to look at this is that by scaling the parameter of the Gaussian kernel, you also scale the $\mathbb{R}^d$ input space [40, Proposition 4.37, pg. 132]. Without the rescaling the input space, there is no guarantee that the scaled Gaussian kernel will maintain the same RKHS.

This challenges the notion that a kernel can produces an all-encompassing RKHS independent of its set of parameters. Hence, a kernel and its parameters must be observed together in the practical analysis of RKHSs. There are two ways to think about this in terms of equality between two RKHSs: there are specific parameters for which the RKHSs produce the same set of functions *or* there is a family of many RKHSs over all possible parameters for which all possible sets of functions are the

same. In the upcoming sections and chapters we analyze the practical equivalence of the Laplace kernel and the NTK by viewing their equivalence through parameter matching.

4.4 Equivalence of the Laplace and Neural Tangent Kernels

It is clear that neural networks and kernel methods have some latent overlap. Belkin et al. [7] empirically found similarities between the Laplace kernel and ReLU activated neural networks when used on the task of fitting random labels. As such, it motivates the question: *Do the Laplace and neural tangent kernels have the same RKHS and if so to what extent?* This question is answered in theory by Geifman et al. [21] and Chen and Xu [11] who showed subset equality between the Laplace RKHS $\mathcal{H}_{Lap}$ and the NTK RKHS $\mathcal{H}_{NTK}$ in the space of $\mathbb{S}^{d-1}$. The forward direction [21] was shown by eigenvalue analysis by way of Mercer's Theorem 3.2.1. The backward direction [11] was shown by utilizing the Subset Inclusion Theorem 4.3.1 and singularity analysis.

This result begs a further question: *What does the practical equivalence of these kernels look like?* Since they are dual representations of each other, this poses a challenge because the NTK relies on depth D that is in the natural numbers which is in great contrast to the Laplace parameter ℓ which is in the positive reals. Due to the vast differences in parameterization, we hypothesize that the bias β plays a role in bridging the gap between the depths.

We begin by considering the matching of the neural tangent kernel $\ddot{k}_{NTK}$ for set depth $D \in \mathbb{N}$ and $\beta \in \mathbb{R}^+$ with the Laplace kernel k_{Lap} for some $\ell \in \mathbb{R}^+ - \{0\}$. It

would then suffice that finding a matching kernel for some inputs $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$ would be shown as follows:

$$\begin{aligned} k_{Lap}(\mathbf{x}, \mathbf{z}; \ell) &= \ddot{k}_{NTK}(\mathbf{x}, \mathbf{z}; D, \beta) \\ \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|}{\ell}\right) &= \ddot{k}_{NTK}(\mathbf{x}, \mathbf{z}; D, \beta) \\ \frac{\|\mathbf{x} - \mathbf{z}\|}{\ell} &= -\log\left(\ddot{k}_{NTK}(\mathbf{x}, \mathbf{z}; D, \beta)\right) \\ \ell &= \frac{\|\mathbf{x} - \mathbf{z}\|}{-\log\left(\ddot{k}_{NTK}(\mathbf{x}, \mathbf{z}; D, \beta)\right)}. \end{aligned} \tag{4.16}$$

Further, we define d_θ as a measure of differences between the kernels given a set of parameters θ:

$$d_{D,\beta,\ell}(\mathbf{x}, \mathbf{z}) = \left|\ddot{k}_{NTK}(\mathbf{x}, \mathbf{z}; D, \beta) - k_{Lap}(\mathbf{x}, \mathbf{z}; \ell)\right|. \tag{4.17}$$

For the kernels to be the same, it is necessary that Equation (4.16) provides a length-scale where the kernels are indeed the same. This is consequential because if this can be done consistently, then the kernels are interchangeable regardless of any optimization procedures that are done while utilizing kernel methods. This brings us to the motivating question: *Can we find a length-scale for which both kernels are identical?*

Table 4.1 showcases this conundrum. Using Equation (4.16) we find that for fixed D and β we can find ℓ such that d_θ of the two kernels is near zero for a *single* input but not for any other input as illustrated in the 1st d_θ column in Table 4.1. Here $\mathbf{x}_1, \mathbf{z}_1$ produce a difference near zero but $\mathbf{x}_2, \mathbf{z}_2$ are nearly 0.1 apart which indicates that this type of matching only works on an input by input basis.

To improve on this, we attempt to vary β and try to find one for a given D such that the ℓ found using Equation (4.16) matches *for all* $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$. This procedure is described in Algorithm 4.1. The idea behind this procedure is that we cannot

Input : $dim, n, D \in \mathbb{N}$ and $b, seed \in \mathbb{R}$

Initialize empty list for means M and variances V

Initialize list of β values B in range $[0, b]$

foreach *β in list B* **do**

Set *seed*

Initialize $\ddot{k}_{NTK}(D, \beta)$

Initialize empty list for length-scales L

for $i \leftarrow 0$ **to** n **do**

$\mathbf{x}_i \leftarrow$ vector size dim with random x entries normalized to $\mathbb{S}^{d-1}$

$\mathbf{z}_i \leftarrow$ vector size dim with random z entries normalized to $\mathbb{S}^{d-1}$

Append length-scale $\ell_i = \frac{||\mathbf{x}_i - \mathbf{z}_i||}{-\log(\ddot{k}_{NTK}(\mathbf{x}_i, \mathbf{z}_i; D, \beta))}$ to L

end

Append $\bar{\ell} = \mathbb{E}[L]$ to M

Append $\text{var}[L]$ to V

end

Return: $\bar{\ell}$ in M and β in B corresponding to $\min V$

Algorithm 4.1: Length-scale and bias matching procedure.

Inputs	$d_{D=3,\beta,\ell}(\mathbf{x}_i, \mathbf{z}_i)$	
	$\beta = 0,\ \ell \approx 1.815$	$\beta \approx 2.122,\ \ell \approx 2.036$
$\mathbf{x}_1 = [0.8027\ 0.2299\ 0.5503]$ $\mathbf{z}_1 = [0.7982\ 0.3818\ 0.4658]$	≈ 0.0	0.001296
$\mathbf{x}_2 = [0.0389\ 0.9663\ 0.2545]$ $\mathbf{z}_2 = [0.6941\ 0.5958\ 0.4040]$	0.0980	0.0000187

Table 4.1: An illustration of the discrepancy between kernel differences d_θ while trying to match ℓ to $\ddot{k}_{NTK}$ of depth $D = 3$ using Equation (4.16). *Left column:* Random inputs in $\mathbb{S}^{d-1}$. *Right table:* Table of differences for specific parameters. Column 1 fixes $\beta = 0$ during matching. Column 2 optimizes β and ℓ using Algorithm 4.1.

directly find an ℓ and β that produce $d_\theta = 0$ using all possible $\mathbf{x}, \mathbf{z}$ so instead we can approximate it by using $n \in \mathbb{N}$ points $(\mathbf{x}_i, \mathbf{z}_i)$ where each $\mathbf{x}_i, \mathbf{z}_i \in \mathbb{S}^{d-1}$. We set D of the NTK and calculate ℓ for some β over every point using Equation (4.16). We then take the mean and variance of the resulting set of length-scales $L = \{\ell_1, \ldots, \ell_n\}$ for that specific β and record it. We repeat this over $m \in \mathbb{N}$ number of β's bounded between 0 and some upper bound (dependent on D). The minimum variance in this experiment indicates that the corresponding mean length-scale $\bar{\ell}_j$ and β_j where $j \in \{1, \ldots, m\}$ are the optimal kernel parameters that produce a small or approximately zero d_θ. This can be summarized as follows:

$$\ell, \beta = \underset{\bar{\ell}_j, \beta_j}{\arg\min} \left\{ \text{var}\left[L\right] \quad \forall\ \beta_1, \ldots, \beta_m \right\}. \tag{4.18}$$

As $n \to \infty$ the entire space gets utilized so the resulting ℓ and β should be optimal.

In our experiments, for depths 1 and 2, $\text{var}[L] \to 0$ as $\beta \to \infty$, depths 3, 4, and 5 attain non-trivial minimums at various β, and for depth > 6 the global minima is near zero. Figure 4.1 illustrates the global minima for depths 1, 3, 5, and 6. The 2nd d_θ column in Table 4.1 illustrates the effect using the optimal ℓ and β values.

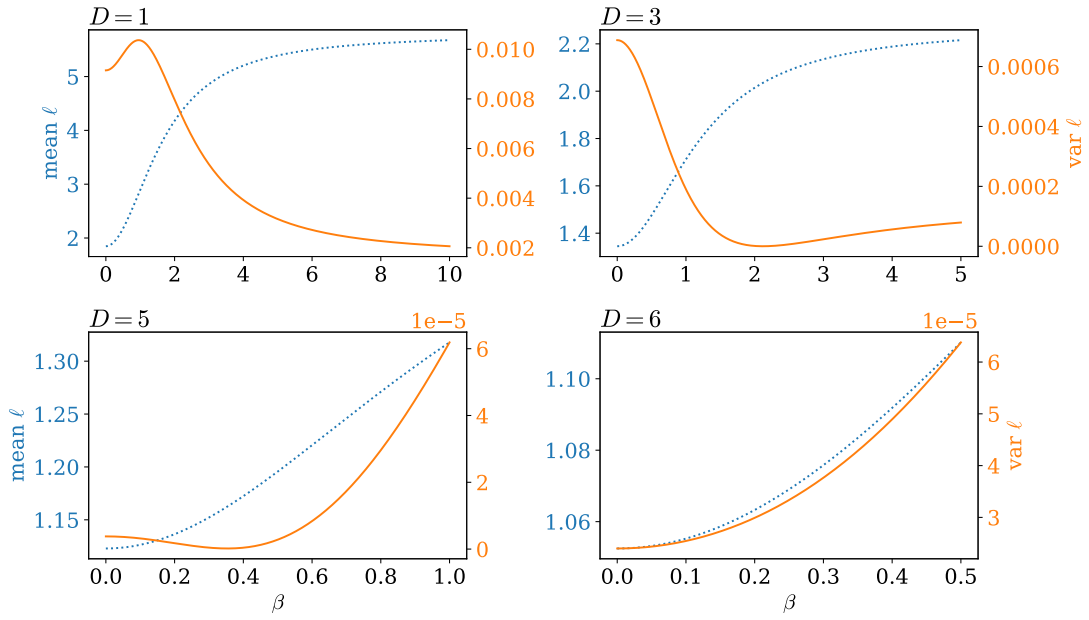

Figure 4.1: Mean and variance plots of ℓ given specific β calculated using $n = 1000$ sample of input pairs for various depths. The solid orange line represents the variance while the dotted blue line represents the mean.

We can see that both points have small d_θ.

Depth 6 is an interesting case because as seen in Figure 4.2, as we look closer to zero we are still unable to attain a global minimum for the variance despite it appearing that $\beta = 0$. The rough looking nature of the plot is due to β being squared in the NTK formulation which results in values less than 10^{-14}. The floating point precision of `numpy` for the machine used to compute this is 10^{-15}. Beyond depth 6, β needs to be orders of magnitude smaller and thus cannot be accurately computed.

In summary, we gained significant insight to the empirical matching of these kernels. In order to equate the kernels, that is, over all possible inputs, we are indeed dependent on both the Laplace kernel length-scale ℓ *and* the NTK bias β.

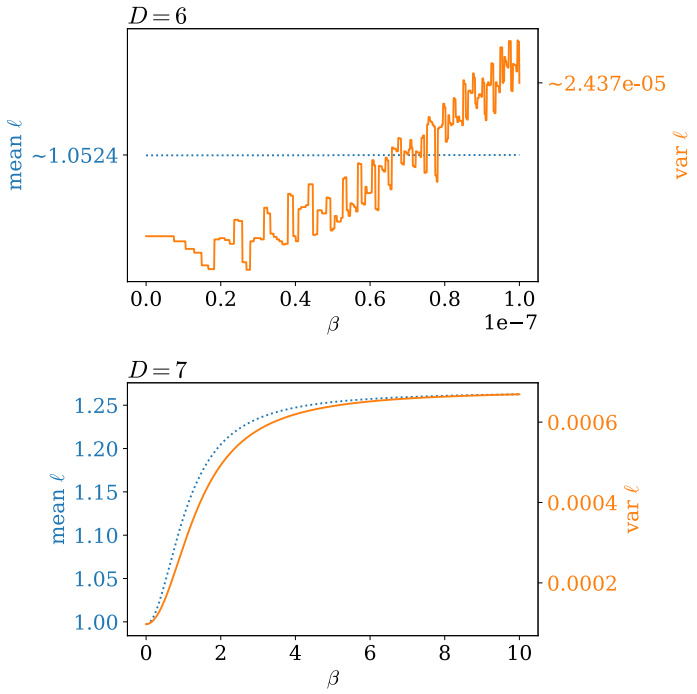

Figure 4.2: Solid orange line represents the variance and the dotted blue line represents the mean. *Top:* A zoom in of depth $D = 6$ for $\beta \in [0, 10^{-7}]$. Due to the zoom, the mean values are all concentrated around ≈ 1.0524 and all variance values are near $\approx 2.437 \cdot 10^{-5}$. The difference between the minimum and maximum variance shown is approximately 10^{-18}. *Bottom:* A showcase of a typical plot past depth 6.

In addition, through Figures 4.1 and 4.2 we have evidence to support the idea that as the NTK depth increases, the optimal bias and length-scale both decrease. From the context of machine learning methods, increasing the depth of a neural network also increases the generalization properties and the susceptibility for overfitting. This is reflected in kernel methods, where a relatively low length-scale parameter produces functions that capture more fine grained detail by closer interpolating over the given dataset (e.g. overfitting).

Chapter 5

Synthetic Experiments

In this chapter we analyze the similarities and differences between the Laplace kernel and the NTK over a number of synthetically generated datasets using GP regression. Specifically, we are interested in matching the posterior means generated by the GP regressor under kernel assumptions, determining the effectiveness of matching in $\mathbb{R}^d$ and $\mathbb{S}^{d-1}$, and analyzing the influence of data transformations on the quality of posterior mean predictions for the NTK. Section 5.1 describes the experimental setup used in this chapter, Section 5.2 focuses on posterior mean matching when $d = 2$, Section 5.3 showcases the the differences in posterior mean matching in $\mathbb{R}^2$ and $\mathbb{S}^1$ on more complex surfaces, and Section 5.4 deals with the quality of posterior mean predictions using high dimensional input datasets.

5.1 Setup

In our experiments, we utilize `scikit-learn` [36] which is a machine learning library for the `Python` programming language. One contribution of this thesis is an

implementation of the NTK[3] that is directly compatible with `scikit-learn`'s GP module. Using this implementation we are able to compute NTK values, optimize the NTK's bias, and train GP models that represent infinite width neural networks. We also use their GP module for the Matérn kernel which yields the Laplace ($\nu = \frac{1}{2}$) and Gaussian ($\nu \to \infty$) kernels in our experiments.

We generate both noisy and noiseless synthetic data for our experiments. The added data noise ϵ is normally distributed with mean zero and variance σ^2 chosen depending on the dataset,

$$\epsilon \sim \mathcal{N}(0, \sigma^2). \tag{5.1}$$

The number of samples we generate depends on the data. In addition, the data is split 50-50 into a training set $(X, \mathbf{y})$ and a testing set $(X_*, \mathbf{y}_*)$ for each experiment. Lastly, we normalize and/or rescale our datasets depending on experiment. *Normalizing* in this context means that we map $\mathbb{R}^d$ inputs (each observation or row vector in X) to the d-sphere space $\mathbb{S}^{d-1}$ by way of the L_2-norm:

$$\mathbf{x} \mapsto \frac{\mathbf{x}}{||\mathbf{x}||_{L_2}}. \tag{5.2}$$

On the other hand, *rescaling* refers to subtracting the sample mean m and dividing sample variance s^2 for a variable (column vector in X) $\mathbf{x}$:

$$\frac{\mathbf{x} - m}{s^2}. \tag{5.3}$$

Our GP regressor setup begins with the kernels; namely, the NTK, Laplace, and Gaussian kernels. During GP regression, the kernel hyperparameters are trained via optimization to best fit the data. Up until now we have been referring to kernel *parameters* because those values are intrinsic to the definition of a kernel. However,

[3] https://github.com/392781/scikit-ntk

in the context of GPs, we refer to those kernel parameters as hyperparameters since they do not directly influence the definition of a GP model. Thus, we define *hyperparameters* as the parameters that are optimized in order to tune a specific model but do not directly describe that model.

All kernels contain the constant value parameter c. In the case when a dataset is noisy, we include a white noise variance σ^2. There are additional hyperparameters that are used to define specific kernels: D and β for the NTK and ν and ℓ for the Matérn kernel. All the listed hyperparameters can be *unfixed* meaning that they can be optimized during training by maximizing the marginal log likelihood of the GP as outlined by Williams and Rasmussen [47].

Furthermore, GPs contain optimization specific options: $n_{restart}$, $\mathbf{y}_{rescale}$, and α. $n_{restart}$ controls the number of optimizer restarts that are done during training in order to find the optimal hyperparameters. Each restart randomly chooses a new initial value from within pre-specified bounds which helps combat what may be a complicated loss surface with many local minima. $\mathbf{y}_{rescale}$ is a boolean value that when true, rescales the response variable during training in order to aid with fitting and optimization. It should be noted that $\mathbf{y}_{rescale}$ is undone at time of prediction. Finally, α is a small positive value that is added to the diagonal of $[K_{XX} + \sigma^2 I_n]^{-1}$ (Equation (2.13)) to ensure positive definiteness in light of any numerical issues. All relevant modifications and parameters are summarized in Table 5.1.

Our main task throughout this chapter is to perfectly match the posterior means of the Laplace kernel and NTK. We use the Gaussian kernel as a comparison that is outside the RKHS in question for our experiments. To accomplish this task we will be optimizing Matérn kernels with $\nu = \frac{1}{2}$ (Laplace) and $\nu \to \infty$ (Gaussian). We begin with an objective function as outlined in Algorithm 5.1 that we will minimize

Data	Description	Value
Normalization	Transforming data to $\mathbb{S}^{d-1}$ (Equation (5.2))	–
Rescaling	Subtracting m, dividing s^2 (Equation (5.3))	–
Optimization	**Description**	**Value**
$n_{restart}$	Number of optimizer restarts	9
$\mathbf{y}_{rescale}$	Response variable rescaling during training	`true`
α	Value to ensure positive definiteness	10^{-5}
Kernel	**Description**	**Value**
D	NTK depth	$2, 3, 10$
β	NTK bias	Unfixed
ν	Matérn kernel smoothness	$\frac{1}{2}, \infty$
ℓ	Length-scale	Unfixed
c	Constant value	Unfixed
σ^2	Data noise variance	Unfixed

Table 5.1: Summary of variables managed in all synthetic experiments. Values left blank are determined per experiment. Unfixed values are ones allowed to be optimized during experiments.

during the optimization process.

The purpose of the objective function is to minimize the RMSE between the Matérn and NTK posterior means by varying the Matérn kernel parameter ℓ,

$$\left\|\bar{\mathbf{f}}_{NTK_*} - \bar{\mathbf{f}}_{Mat_*}\right\|_{L_2}. \tag{5.4}$$

The procedure is outlined below in Algorithm 5.2 utilizes parameters for the GPs and hyperparameters for the kernels as expressed in Table 5.1. It should be noted that OPTIMIZE utilizes `sklearn.gaussian_process.GaussianProcessRe-`

Input : $k_{Mat}(\theta), \bar{\mathbf{f}}_{NTK_*}, X, \mathbf{y}, X_*$

$f_{Mat_{opt}}, \theta_{opt} \leftarrow$OPTIMIZE$\{f_{Mat} \sim \mathcal{GP}(0, k_{Mat}(\theta))|_{X,\mathbf{y}} : n_{restart}, \mathbf{y}_{rescale}, \alpha\}$

$\bar{\mathbf{f}}_{Mat_*} \leftarrow f_{Mat_{opt}}(X_*)$

Return: $\left\|\bar{\mathbf{f}}_{NTK_*} - \bar{\mathbf{f}}_{Mat_*}\right\|_{L_2}$

Algorithm 5.1: Objective function `obj_func` for posterior mean matching optimization.

Define : $n_{restart}$, $\mathbf{y}_{rescale}$, α, *noise*

Input : $D, \nu, X, \mathbf{y}, X_*$ (Train and test data pre-processed the same way)

NTK fitting and optimization

$\ddot{k}_{NTK}(\theta) \leftarrow c \cdot \ddot{k}_{NTK}(D, \beta); \quad \theta := \{D, \beta, c\}$

if *noise is True* **then** $\ddot{k}_{NTK}(\theta) \leftarrow \ddot{k}_{NTK}(\theta) + \sigma^2 \delta_{\mathbf{s}=\mathbf{t}}; \quad \theta = \{\ldots, \sigma^2\}$

$f_{NTK_{opt}}, \theta_{opt} \leftarrow$ OPTIMIZE$\{f_{NTK} \sim \mathcal{GP}(\,0, \ddot{k}_{NTK}(\theta)\,)|_{X,\mathbf{y}} : n_{restart}, \mathbf{y}_{rescale}, \alpha\}$

$\bar{\mathbf{f}}_{NTK_*} \leftarrow f_{NTK_{opt}}(X_*)$

β_{opt}, $c_{opt}, \sigma^2_{opt} \leftarrow \{\ldots, \beta, c, \sigma^2\}_{\theta_{opt}}$

Matérn posterior matching

$k_{Mat}(\theta) \leftarrow c_{opt} \cdot k_{Mat}(\nu, \ell); \quad \theta := \{\nu, \ell, c_{opt}\}$

if *noise is True* **then** $k_{Mat}(\theta) \leftarrow k_{Mat}(\theta) + \sigma^2 \delta_{\mathbf{s}=\mathbf{t}}); \quad \theta = \{\ldots, \sigma^2\}$

$f_{Mat_{opt}}, \theta_{opt} \leftarrow$MINIMIZE$\{$`obj_func`$(k_{Mat}(\theta), \bar{\mathbf{f}}_{NTK_*}, X, y, X_*)\}$

$\ell_{opt} \leftarrow \{\ldots, \ell, \ldots\}_{\theta_{opt}}$

Return: $c_{opt}, \sigma^2_{opt}, \beta_{opt}, \ell_{opt}, f_{NTK_{opt}}, f_{Mat_{opt}}$

Algorithm 5.2: Posterior mean matching for Matérn kernels. The optimized model is denoted as $f_{opt}(\cdot) := f(\cdot; \theta_{opt})$. OPTIMIZE refers to the GP optimization process of maximizing the marginal log likelihood.

`gressor`'s `fit` function and MINIMIZE uses the `scipy.optimize.minimize_scalar` function to perform the key steps in our procedure.

We control NTK's depth parameter D and the choice of Matérn kernel smoothness ν. It should be noted that for our experiments we only allow the constant value c and error term σ^2 parameters to optimize for the NTK while the respective parameters for the Matérn kernels inherit the NTK's optimized values. The reasoning lies in the relation between a kernel and a GP covariance function. For values $\mathbf{s}, \mathbf{t} \in \mathcal{X}$, the covariance function of a GP $f \sim \mathcal{GP}(0, k)$ is related directly to its kernel (and thus f depends on choice of k):

$$\text{cov}(f(\mathbf{s}), f(\mathbf{t})) = k(\mathbf{s}, \mathbf{t}) \tag{5.5}$$

So, a scaled GP is expressed as $\sqrt{c}f \sim \mathcal{GP}(0, ck)$ because its covariance function simplifies as follows:

$$\text{cov}(\sqrt{c}f(\mathbf{s}), \sqrt{c}f(\mathbf{t})) = (\sqrt{c})^2\text{cov}(f(\mathbf{s}), f(\mathbf{t})) = ck(\mathbf{s}, \mathbf{t}). \tag{5.6}$$

Lastly, a GP that includes an additive noise term is expressed as $f + \epsilon \sim \mathcal{GP}(0, k + \sigma^2\delta_{\mathbf{s}=\mathbf{t}})$ where $\sigma^2\delta_{\mathbf{s}=\mathbf{t}}$ is the *white noise kernel* with a constant value represented by the noise variance σ^2 and the kernel itself being the Kronecker delta $\delta_{\mathbf{s}=\mathbf{t}}$. This is justified because

$$\begin{aligned}
&\text{cov}(f(\mathbf{s}) + \epsilon(\mathbf{s}), f(\mathbf{t}) + \epsilon(\mathbf{t})) = \\
&\text{cov}(f(\mathbf{s}), f(\mathbf{t})) + \\
&\text{cov}(f(\mathbf{s}), f(\mathbf{t}) + \epsilon(\mathbf{t})) + \text{cov}(\epsilon(\mathbf{s}), f(\mathbf{t}) + \epsilon(\mathbf{t})) + \\
&\text{cov}(f(\mathbf{s}) + \epsilon(\mathbf{s}), f(\mathbf{t})) + \text{cov}(f(\mathbf{s}) + \epsilon(\mathbf{s}), \epsilon(\mathbf{s})) + \\
&\text{cov}(\epsilon(\mathbf{s}), \epsilon(\mathbf{t})) \\
&= k(\mathbf{s}, \mathbf{t}) + \sigma^2\delta_{\mathbf{s}=\mathbf{t}}.
\end{aligned} \tag{5.7}$$

As a result, if we wish to compare just the kernels while maintaining their expressivity through transformations, it is necessary to keep the constant value and noise variance terms the same between kernels.

We find that our procedure is a difficult optimization task due to the objective function landscape containing many local minima. Our `Python` implementation[4] deals with this by modifying the minimization process to search by slowly expanding the search bounds for ℓ over multiple runs. In our experiments we consider two metrics for posterior mean matching: Root mean-squared error (RMSE) and Pearson correlation coefficient (ρ). The former gives an idea of the ability to minimize the objective function's loss and latter is used to see how well the posterior means of different kernels overlap with each other.

5.2 Illustrative Example

To motivate the remainder of this work, we will first attempt to answer the following question and analyze the results produced: *Does our posterior mean matching procedure work?* We begin by creating a simple 2D input parametric curve as follows:

$$\begin{gathered} x_1 = (y^2 + 1) \cdot \sin(t) \\ x_2 = (y^2 + 1) \cdot \cos(t) \\ y \in [-2, 2] \\ \text{where } t \in [-2\pi, 2\pi]. \end{gathered} \tag{5.8}$$

We independently sample 100 values of $t \sim U[-2, 2]$ and $y \sim U[-2\pi, 2\pi]$ where $U[a, b]$ is the uniform distribution between points a and b, inclusive. We generate

[4]Implementation details at https://github.com/392781/neural-laplace.

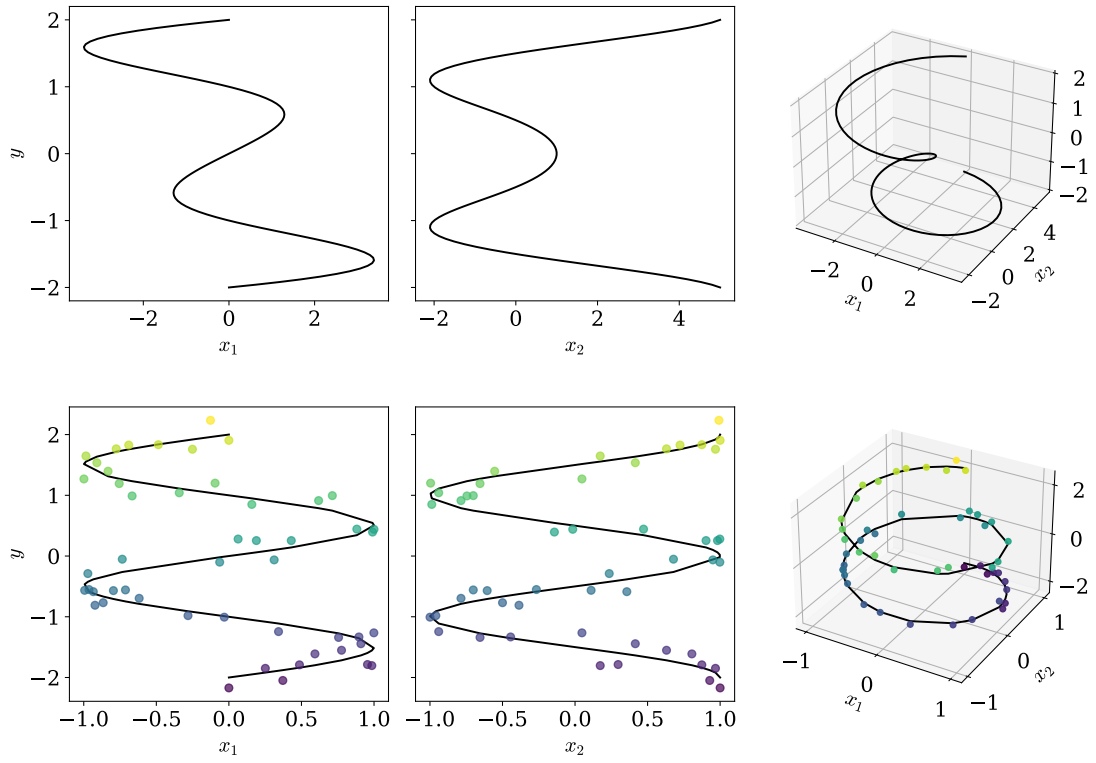

Figure 5.1: *Top:* The parametric curve defined in Equation (5.8). *Bottom:* Equation (5.8) with inputs normalized and noisy training points shown.

x_1 and x_2 using the t and y samples. Although the curve is defined by starting with samples of y and t, we treat points (x_1, x_2) as the inputs and y as the output during fitting. We chose this curve because it very clearly illustrates the posterior mean matching between the NTK and Matérn kernels. The dependence on t allows us to connect the points in a way that other surfaces and datasets do not. We then split our inputs (x_1, x_2) and output y into a training set size $n = 50$ and testing set size $m = 50$. We make two separate experiments: non-noisy and noisy with $\epsilon \sim \mathcal{N}(0, 0.15^2)$. We utilize the experiment setup outlined in Table 5.1. In addition, we normalize the data inputs (x_1, x_2) to $\mathbb{S}^1$ prior to training. We then perform the posterior mean matching as outlined in Algorithm 5.2.

		$D = 2$		$D = 3$		$D = 10$	
Metric	Noise	Lap	Gaus	Lap	Gaus	Lap	Gaus
RMSE	No	0.0860	0.3352	0.0250	0.3610	0.0079	0.3670
	Yes	0.0563	0.0228	0.0314	0.0412	0.0494	0.0915
ρ	No	0.9974	0.9590	0.9998	0.9509	0.9999	0.9462
	Yes	0.9954	0.9991	0.9984	0.9980	0.9929	0.9783

Table 5.2: The results of posterior mean matching the NTK to Matérn kernels for the parametric dataset in $\mathbb{S}^1$.

Fitting the parametric curve shows a number of interesting results. The optimization procedure performs well and can accurately match the mean posteriors of the Laplace and NTK GPs. Table 5.2 indicates this with very small RMSE values and Figure 5.2 visually shows an exact match. More than that, the posterior means correlate almost perfectly ($\rho > 0.99$) regardless if the GPs were trained on noisy

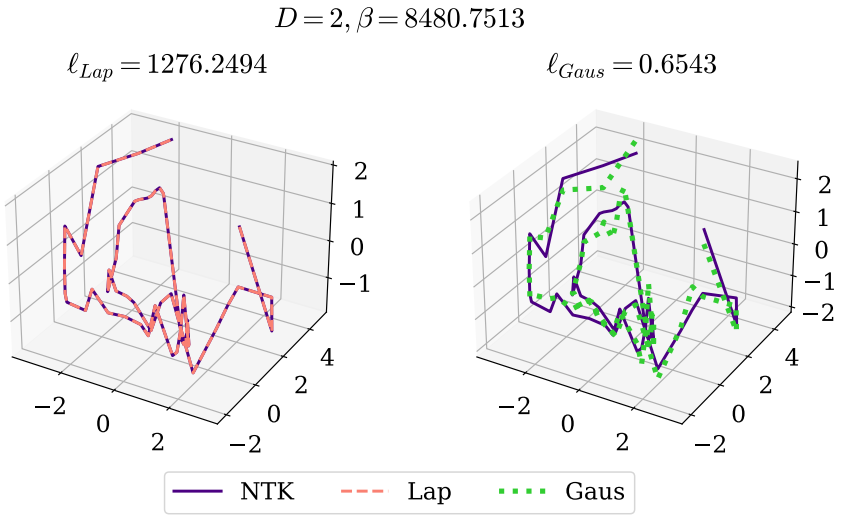

Figure 5.2: Posterior means generated by fitting non-noisy parametric curve data in $\mathbb{S}^1$ and predicted on out of sample data in $\mathbb{S}^1$. For visualization purposes we set $(x_1, x_2) \in \mathbb{R}^2$.

data or not.

On the other hand, the Gaussian kernel and NTK fail to perfectly match up in the case where no noise is present as can be seen in Table 5.2. Although there is a high correlation between the posterior means, Figure 5.2 illustrates how the Gaussian kernel cannot match the NTK's predictions to the exact effect that the Laplace kernel does. This is also indicated by a higher resulting RMSE during optimization.

One important thing of note is the low RMSE and high correlation values in Table 5.2 for both the Laplace and Gaussian kernels when noisy data is present. This can be explained by the fact that adding the noise term σ^2 to our kernels gives the resulting regression additional smoothness and increased bounds for error while fitting. When matching posterior means, the regressors can use this inferred noise to their advantage to better minimize our objective function resulting in the values seen in the table. Without σ^2, our GP regressors would attempt to perfectly interpolate through the data.

D	Noise	β	ℓ_{Lap}	ℓ_{Gaus}	c	σ^2
2	No	8480.751	1276.249	0.6543	32.495	–
	Yes	0.000010	1.0569	0.8414	0.2734	0.7910
3	No	453.111	765.133	0.6540	21.515	–
	Yes	0.000014	1.0046	0.8242	0.2722	0.7916
10	No	469.676	0.3702	0.6249	6.6946	–
	Yes	0.000010	0.3026	0.2152	0.2350	0.8177

Table 5.3: Kernel hyperparameter results for posterior mean matching with inputs in $\mathbb{S}^1$.

Lastly, Table 5.3 shows the trained kernel hyperparameters after the posterior mean matching procedure. As noted in Section 4.4, we see that the Laplace length-scale decreases as NTK depth increases. The Gaussian length-scale stays approximately the same regardless of depth when the data is non-noisy. Both the Laplace and Gaussian length-scales decrease as depth increases when the data is noisy which may be explained by the additional smoothness added by the noise term σ^2.

5.3 Synthetic 2D Input Case

In this section we look at more complicated functions in the 2D input space. Since the Laplace kernel and NTK share the same RKHS in $\mathbb{S}^{d-1}$, it is of interest to determine how these kernels behave in the space of $\mathbb{R}^d$. We consider 3 different functions found in literature summarized in Tables 5.4 and 5.5.

During the calculation of the NTK, the computational space complexity scales exceptionally poorly due to the recursive nature of the kernel which caused issues on our hardware. Thus this required a way to minimize the dataset size while retaining enough information to generalize our model. For the following experiments, we

Dataset	$f(x_1, x_2)$
Ackley [2]	$-20\exp\left[-\frac{1}{5}\sqrt{\frac{1}{2}(x_1^2+x_2^2)}\right] - \exp\left[\frac{1}{2}(\cos 2\pi x_1 + \cos 2\pi x_2)\right] + e + 20$
Franke [18]	$0.75\exp\left(-\frac{(9x_1-2)^2}{4} - \frac{(9x_2-2)^2}{4}\right) + 0.75\exp\left(-\frac{(9x_1+1)^2}{49} - \frac{9x_2+1}{10}\right) + 0.5\exp\left(-\frac{(9x_1-7)^2}{4} - \frac{(9x_2-3)^2}{4}\right) - 0.2\exp\left(-(9x_1-4)^2 - (9x_2-7)^2\right)$
Nonpoly. [45]	$\frac{1}{6}\left[(30+5x_1\sin(5x_1))(4+\exp(-5x_2)) - 100\right]$

Table 5.4: Three 2D input functions with $(x_1, x_2) \in \mathbb{R}^2$.

Ackley	Franke	Nonpolynomial
$x_1, x_2 \sim U[1, 7]$	$x_1, x_2 \sim U[-0.5, 1]$	$x_1, x_2 \sim U[0, 2]$
$\epsilon \sim \mathcal{N}(0, 0.75^2)$	$\epsilon \sim \mathcal{N}(0, 0.10^2)$	$\epsilon \sim \mathcal{N}(0, 1)$

Table 5.5: 2D input function input sampling distributions and noise used for the noisy experiments.

utilize Latin hypercube sampling [27]. To understand Latin hypercube sampling it is simpler to define Latin square sampling first. *Latin square sampling* is where only one sample is chosen in each row and column of a square grid. As a result, *Latin hypercube sampling* is a generalization of the Latin square giving a sample that tries to yield approximately equidistant points in given boundaries. We use it because it better captures the overall surface of these functions during training whilst utilizing fewer training points.

In these experiments we have 2 groups ($\mathbb{R}^2$ and normalized to $\mathbb{S}^1$) each containing a non-noisy and noisy dataset for a total of 4 datasets per function. For each function we sample a grid of 1000 points using Latin hypercube sampling. We split

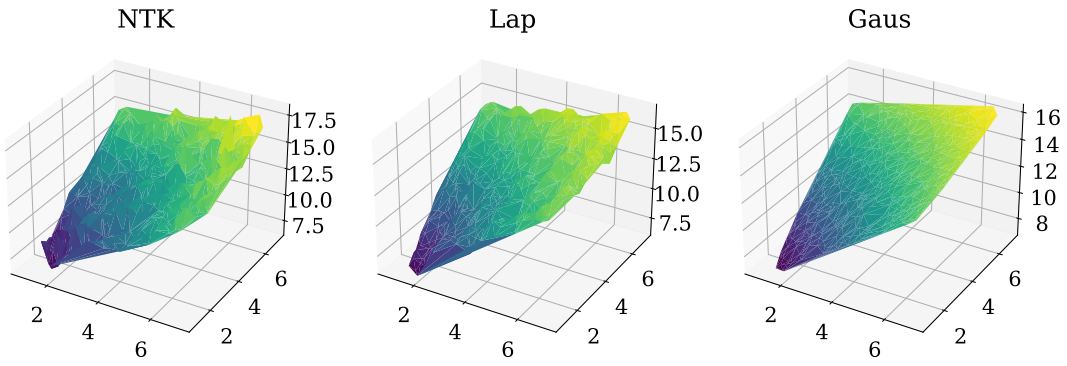

Figure 5.3: Posterior means for the noisy the Ackley function in $\mathbb{R}^2$ for NTK depth $D = 2$ with test size $m = 500$. GPs were trained using $n = 500$ inputs $x_1, x_2 \in [1, 7]$ generated using Latin hypercube sampling.

the sample into a training dataset size $n = 500$ and a testing dataset size $m = 500$. For all other parameters we adopt the defaults laid out in Table 5.1.

One reoccurring theme in these experiments is the white noise term σ^2 performing the role of smoothing out the predictions. As mentioned in the previous section, taking data noise into account allows for more leeway in matching using Algorithm 4.1. Figure 5.3 illustrates this via the posterior means of the Ackley function. The smoothed surface of the NTK is similar to that of a plane and thus closely matched

			$D = 2$		$D = 3$		$D = 10$	
Metrics	Dataset	Noise	Lap	Gaus	Lap	Gaus	Lap	Gaus
RMSE	Ackley	No	0.7881	0.6695	0.7868	0.6693	0.7790	0.6682
		Yes	0.3296	0.3115	0.3370	0.3206	0.3816	0.3712
	Franke	No	0.0947	0.0957	0.0946	0.0968	0.0935	0.0965
		Yes	0.0487	0.0532	0.0496	0.0541	0.0551	0.0595
	Nonpoly	No	2.6526	2.5916	2.6526	2.5916	2.6513	2.5905
		Yes	0.8275	0.8411	0.8307	0.8448	0.8461	0.8640
ρ	Ackley	No	0.9407	0.9558	0.9408	0.9559	0.9419	0.9560
		Yes	0.9890	0.9900	0.9884	0.9894	0.9849	0.9857
	Franke	No	0.9330	0.9328	0.9332	0.9310	0.9341	0.9315
		Yes	0.9791	0.9753	0.9781	0.9744	0.9724	0.9688
	Nonpoly	No	0.3760	0.4255	0.3760	0.4255	0.3761	0.4254
		Yes	0.8000	0.7994	0.7985	0.7967	0.7883	0.7809

Table 5.6: Posterior mean matching results for the 2D input surface datasets in $\mathbb{R}^2$ with test size $m = 500$. GPs were trained using $n = 500$ inputs x_1, x_2 generated using Latin hypercube sampling over the respective function domains.

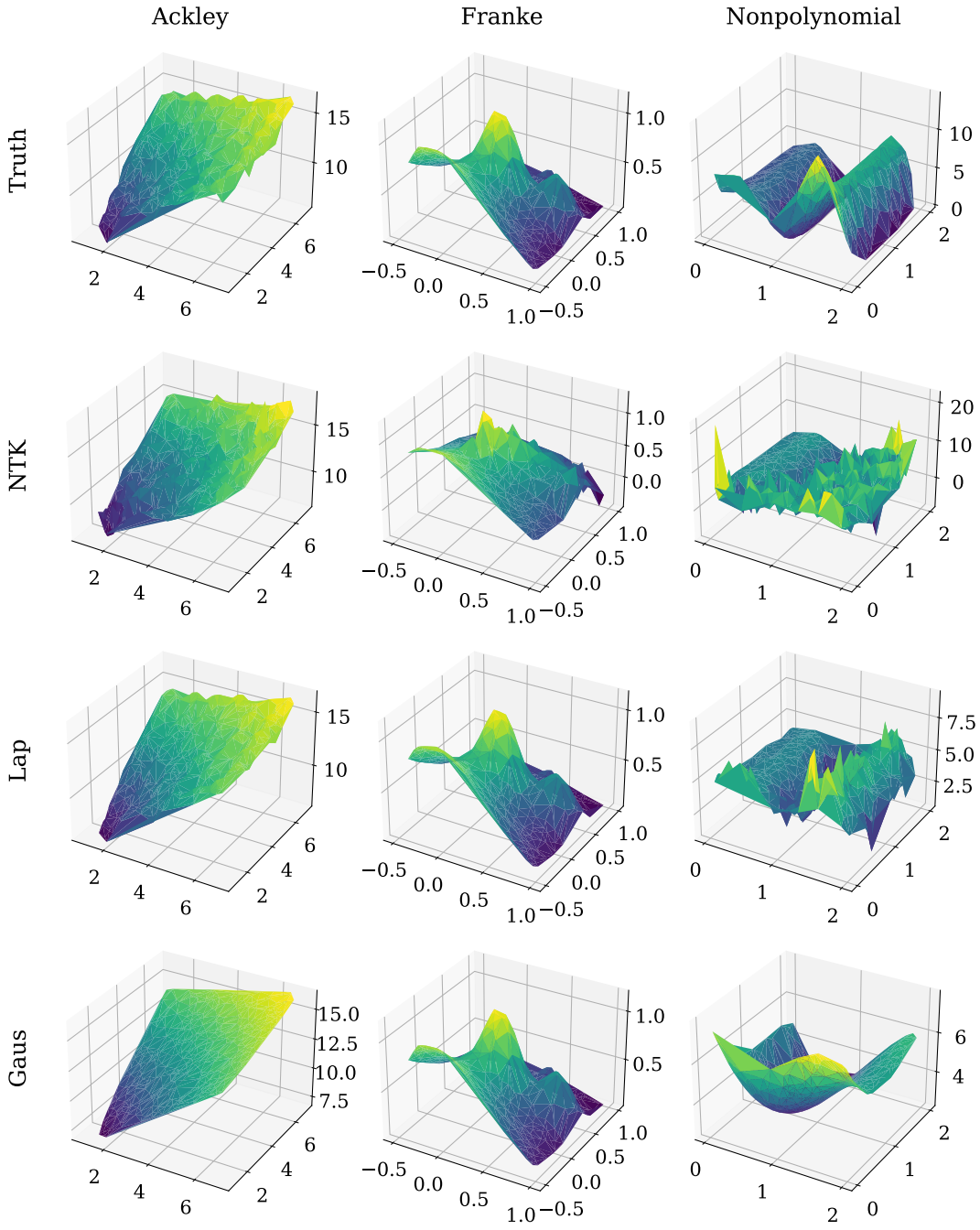

Figure 5.4: Posterior means of the non-noisy 2D input functions trained in $\mathbb{R}^2$ for NTK depth $D = 3$ with test size $m = 500$. GPs were trained using $n = 500$ inputs x_1, x_2 generated using Latin hypercube sampling over the respective function domains.

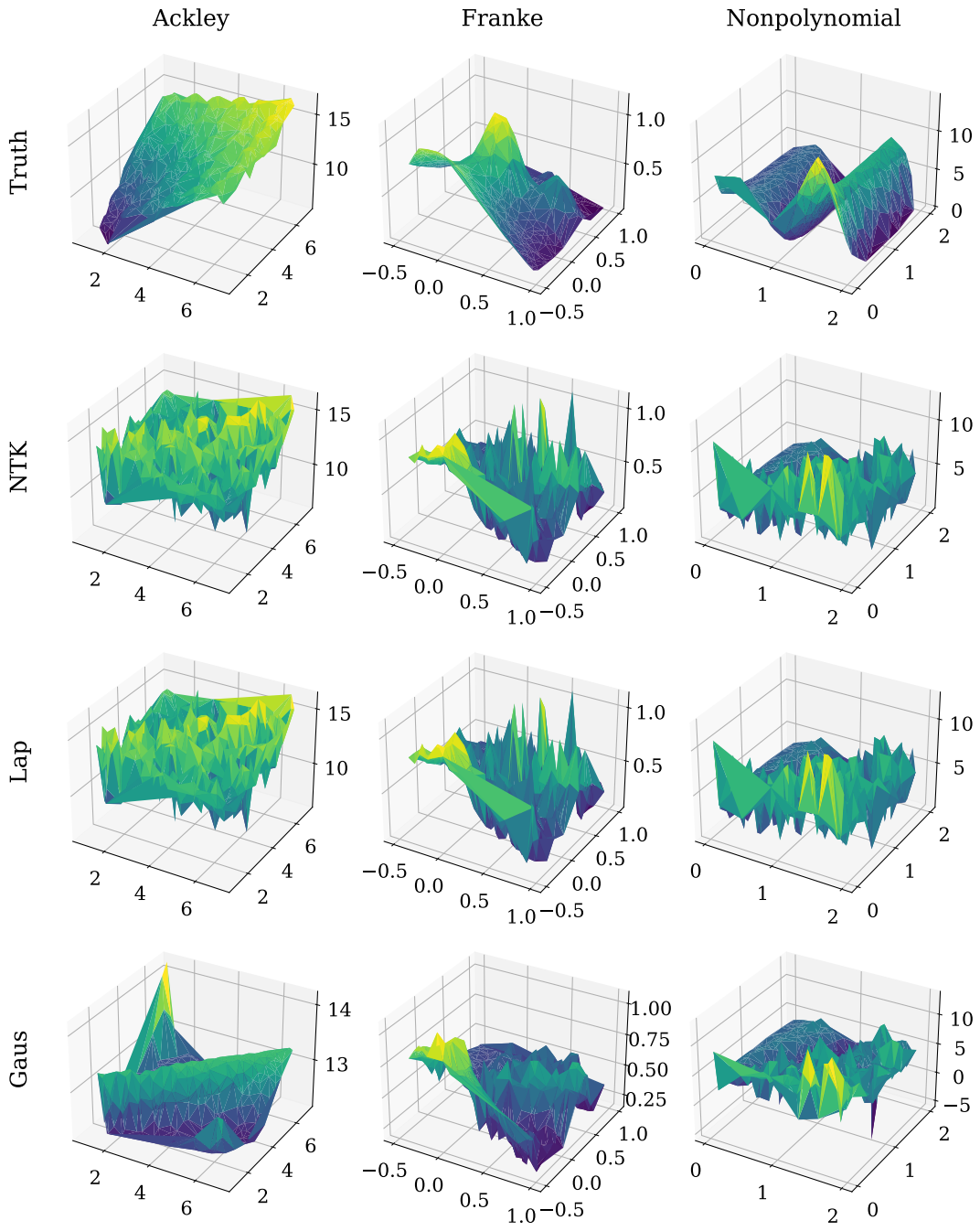

Figure 5.5: Posterior means of the non-noisy 2D input functions trained in $\mathbb{S}^1$ for NTK depth $D = 3$ with test size $m = 500$. GPs were trained using $n = 500$ inputs x_1, x_2 generated using Latin hypercube sampling over the respective function domains. For visualization $(x_1, x_2) \in \mathbb{R}^2$.

to by the Laplace and Gaussian kernels. Much like noisy data in $\mathbb{S}^1$, this effect is seen in all noisy $\mathbb{R}^2$ experiments where the posterior mean correlation is always higher compared to the corresponding non-noisy experiments.

Looking at the $\mathbb{R}^2$ experiments in Table 5.6 and Figure 5.4, it is apparent from the correlations of the posterior means that the Laplace kernel is not able to match the NTK. This is the same case for the Gaussian kernel as well. The nonpolynomial dataset seems to be especially hard to match ($\rho \approx 0.5$). The Laplace and Gaussian kernels both provide similar resulting correlations in all configurations in $\mathbb{R}^2$. Based on these experiments, the conclusion is that the elements of the Laplace kernel's and NTK's RKHSs (e.g. the posterior means) do not fully overlap in the general space $\mathbb{R}^d$.

As mentioned previously, this is in contrast to the space of $\mathbb{S}^{d-1}$ where we observe nearly perfect correlation and low RMSE values during the posterior mean matching procedure for the Laplace kernel and NTK. Details of these experiments are can be found in Table C.1 in Appendix C. Figure 5.5 shows that the Laplace kernel and NTK have effectively the same posterior means whereas the Gaussian kernel's posterior falls short of matching. The two spaces presented in the figures have very different posterior means. Arguably, the space of $\mathbb{R}^2$ provides results more in-line with the ground truth than the space of $\mathbb{S}^1$. This observation is further explore in the next section. Lastly, we would like to draw attention to the noisy Ackley function in $\mathbb{S}^1$ which was a difficult dataset to fit utilizing the NTK. We discuss the details of this in Figure C.2 in Appendix C.

5.4 Synthetic High Dimensional Cases

Knowing that we can reliably match the posterior means of the Laplace kernel and NTK in $\mathbb{S}^{d-1}$, we now turn our focus to the quality of predictions while utilizing these kernels. Specifically, we focus on the NTK's ability to generate meaningful posterior means. We test this using the multidimensional Friedman datasets [19, 10] which are used to benchmark and test linear regression models. The datasets and their features are summarized in Tables 5.7 and 5.8 respectively.

We maintain the same experiment setup as stated in Table 5.1; however, we

Dataset	$f(x_1, x_2, \ldots, x_d)$	d
Friedman 1	$10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x^4 + 5x^5$	10
Friedman 2	$\left(x_1^2 + (x_2 x_3 - (x_2 x_4)^{-2})^2\right)^{1/2}$	4
Friedman 3	$\arctan\left(\frac{x_2 x_3 - (x_2 x_4)^{-1}}{x_1}\right)$	4

Table 5.7: Friedman datasets along with their corresponding input dimensions where $(x_1, \ldots, x_d) \in \mathbb{R}^d$. Friedman 1 dataset's output is independent of the last five input variables hence why the dataset has a total of 10 input dimensions.

Friedman 1	Friedman 2	Friedman 3
$x_1, \ldots, x_{10} \sim U[0, 1]$	$x_1 \sim U(0, 100]$ $x_2 \sim U[40\pi, 560\pi]$ $x_3 \sim U[0, 1]$ $x_4 \sim U[1, 11]$	
$\epsilon \sim \mathcal{N}(0, 1.5)$	$\epsilon \sim \mathcal{N}(0, 5)$	$\epsilon \sim \mathcal{N}(0, 0.15)$

Table 5.8: Friedman data feature distributions. Friedman 2 and 3 share the same feature distributions. The noise term ϵ is applied only for the noisy data cases.

utilize the *coefficient of determination* (R^2) as a new metric for this section as a way to assess regression performance:

$$R^2 = 1 - \frac{\textit{residual sum of squares}}{\textit{total sum of squares}} = 1 - \frac{\sum_i (y_{*_i} - f_{*_i})^2}{\sum_i (y_{*_i} - \bar{y}_*)^2}, \tag{5.9}$$

where y_{*_i} is the ground truth value from the testing set and f_{*_i} is the predicted value. R^2 can attain a maximum value of 1 indicating perfect interpolation, a value of 0 indicating performance equivalent to the mean of the ground truth $\bar{y}_*$, and arbitrary negative values indicating worse performance than $\bar{y}_*$. Our goal in this section is to test the impact of rescaling inputs and/or outputs while using the NTK in both $\mathbb{R}^d$ and $\mathbb{S}^{d-1}$. Alongside this, we continue to perform posterior mean matching to the NTK for the Matérn kernels.

For each dataset we generate 200 samples with $n = 100$ for the training set and $m = 100$ for the testing set. In the experiments where both normalization and rescaling is used on the input space, we first rescale our inputs and then normalize them to $\mathbb{S}^{d-1}$. It is important to rescale first because doing otherwise does not guarantee that our inputs will lie in $\mathbb{S}^{d-1}$. We also treat rescaling inputs ($X_{rescale}$) and outputs ($\mathbf{y}_{rescale}$) as separate experiments. Output rescaling is only done during training as outlined in Table 5.1.

Our experiment results show that rescaling the inputs has a significant positive impact on regression results for the NTK. This is best illustrated in Table 5.9 where we consistently see that input rescaling increases our R^2 value significantly. On the other hand output rescaling does not have a significant impact on the quality of the regression as indicated by the None and $\mathbf{y}_{rescale}$ columns. That said, output rescaling helps with hyperparameter optimization. Without it we hit upper bounds during optimization for hyperparameters like constant value or NTK's bias. Lastly, applying both input and output rescaling yields the best regression results. The

		Non-noisy				Noisy			
D	Sp.	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both
2	$\mathbb{S}^9$	0.6549	0.7792	0.6447	0.8054	0.5919	0.6993	0.5843	0.7154
	$\mathbb{R}^{10}$	0.7917	0.7857	0.7924	0.7954	0.6947	0.7000	0.7054	0.7086
3	$\mathbb{S}^9$	0.6481	0.7672	0.6353	0.7880	0.5924	0.6913	0.5820	0.7054
	$\mathbb{R}^{10}$	0.7860	0.7697	0.7833	0.7788	0.6901	0.6898	0.7010	0.7007
10	$\mathbb{S}^9$	0.5956	0.6302	0.5785	0.6565	0.5538	0.5781	0.5406	0.5992
	$\mathbb{R}^{10}$	0.7569	0.6301	0.7069	0.6595	0.6711	0.5727	0.6459	0.6048

Table 5.9: Friedman 1 R^2 results for NTK posterior means with training done using various data transformations. The None column is the baseline with no input or output rescaling, $X_{rescale}$ column is with input rescaling only, $\mathbf{y}_{rescale}$ column is with output rescaling during training only, and the Both column applies both types of rescaling.

data transformations do not greatly impact Laplace kernel and NTK posterior mean matching as seen in Table 5.10. We find similar results for the Friedman 2 and 3 datasets found in Tables C.2, C.3, C.4, and C.5 in Appendix C.

We move on to visualizations of the different data transformations. In the remainder of the figures in this section, we include a bottom row that contains av-

	Non-noisy				Noisy			
D	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both
2	0.9974	≈ 1	0.9981	0.9961	0.9985	0.9991	0.9995	0.9951
3	0.9983	≈ 1	0.9991	0.9986	0.9988	0.9996	0.9992	0.9983
10	0.9985	0.9989	0.9992	0.9999	0.9984	0.9990	0.9993	0.9999

Table 5.10: Friedman 1 ρ results for Laplace kernel and NTK posterior mean matching in $\mathbb{S}^9$ with training done using various data transformations.

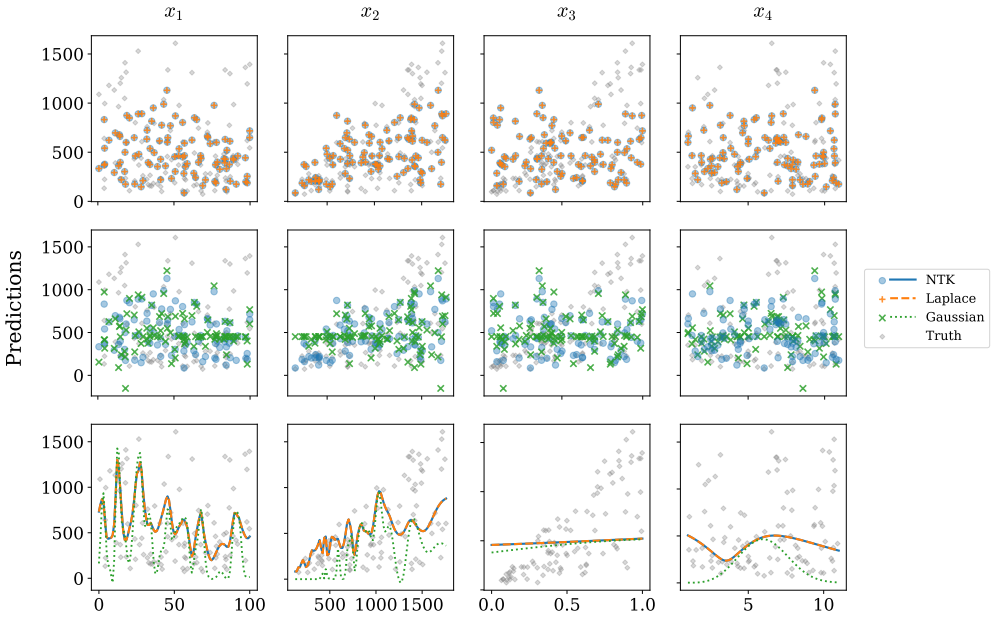

Figure 5.6: Predictions for non-noisy Friedman 2 in $\mathbb{S}^3$ for $D = 2$ with $\mathbf{y}_{rescale}$. *Top:* NTK and Laplace predictions overlayed. *Middle:* NTK and Gaussian predictions overlayed. *Bottom:* Averaged prediction plots of all kernels.

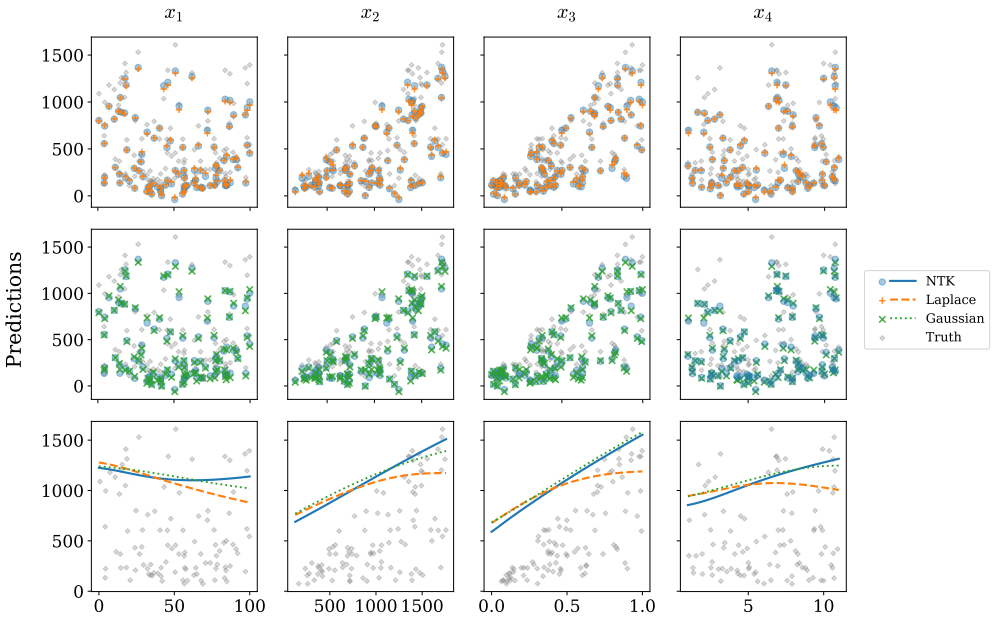

Figure 5.7: Predictions for non-noisy Friedman 2 in $\mathbb{S}^3$ for $D = 2$ with $X_{rescale}$ and $\mathbf{y}_{rescale}$. The rows of the figure are laid out as in Figure 5.6.

eraged prediction plots of all the kernels. These plots are created by using a trained GP to predict over a dataset where we uniformly sample from one input dimension while the remaining inputs are set as the mean of their respective distribution (Table 5.8). As a result, we can visualize the sampled dimension as a 2D plot (e.g. x_1 vs f_*).

We begin by differentiating the posterior results for scaled and unscaled inputs. For this we turn to the non-noisy Friedman 2 dataset in $\mathbb{S}^3$ with outputs rescaled and NTK $D = 2$. Figures 5.6 and 5.7 showcase unscaled and scaled inputs respectively. In the unscaled input case in Figure 5.6, the GP poorly generalizes for all inputs aside from x_2 ($R^2 \approx -0.071$, Table C.2). On the other hand, the scaled inputs in Figure 5.7 do a much better job in producing a good regression fit over the test data ($R^2 \approx 0.855$, Table C.2). These results indicate that input rescaling seems to improve regression predictions and does not impact our ability to match posterior means. We include results for the noisy Friedman 2 dataset with the same setup in Figures C.3 and C.4 in Appendix C.

We now backtrack a bit to address the high R^2 values in $\mathbb{R}^d$ as opposed to $\mathbb{S}^{d-1}$. This is almost always the case regardless of dataset or configuration as seen in Tables 5.9 (Friedman 1), C.2 (Friedman 2), and C.3 (Friedman 3). We provide a visualization of this phenomenon for the non-noisy Friedman 3 dataset with both inputs and outputs rescaled and NTK $D = 2$. Figures 5.8 and 5.9 on the following page showcase the posterior means of $\mathbb{R}^4$ ($R^2 \approx 0.692$) and $\mathbb{S}^3$ ($R^2 \approx 0.033$) respectively. The low R^2 of $\mathbb{S}^3$ is attributed close values between the residual sum of squares and the total sum of squares in Equation (5.9). The differences of the predictions appear to be very minor between the spaces. Looking at the first row of both figures it appears that the inputs in $\mathbb{R}^4$ have a slightly tighter fit over the test

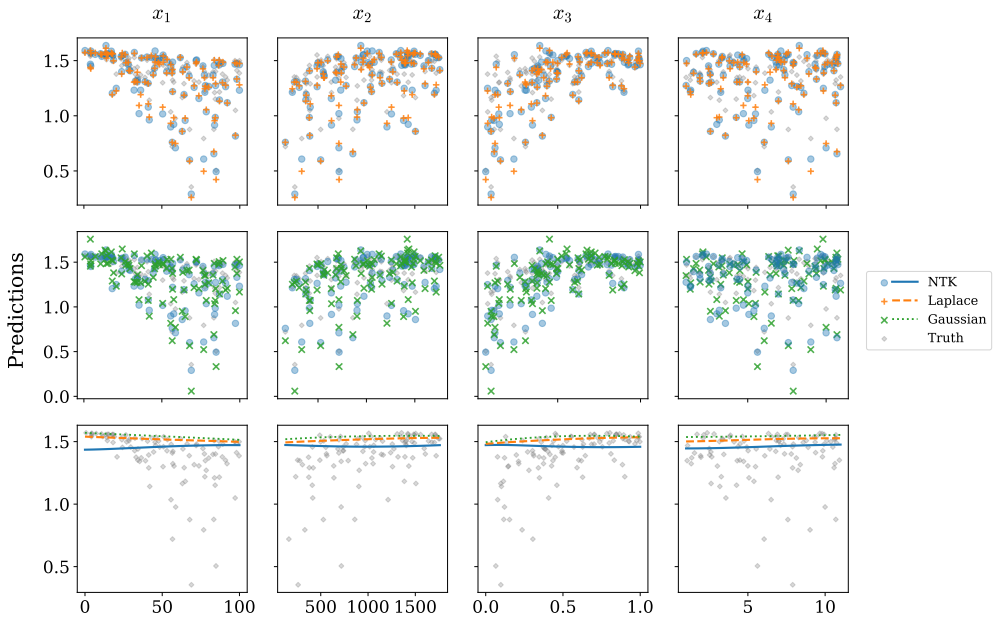

Figure 5.8: Predictions for non-noisy Friedman 3 in $\mathbb{R}^4$ for $D = 2$ with $X_{rescale}$ and $\mathbf{y}_{rescale}$. The rows of the figure are laid out as in Figure 5.6.

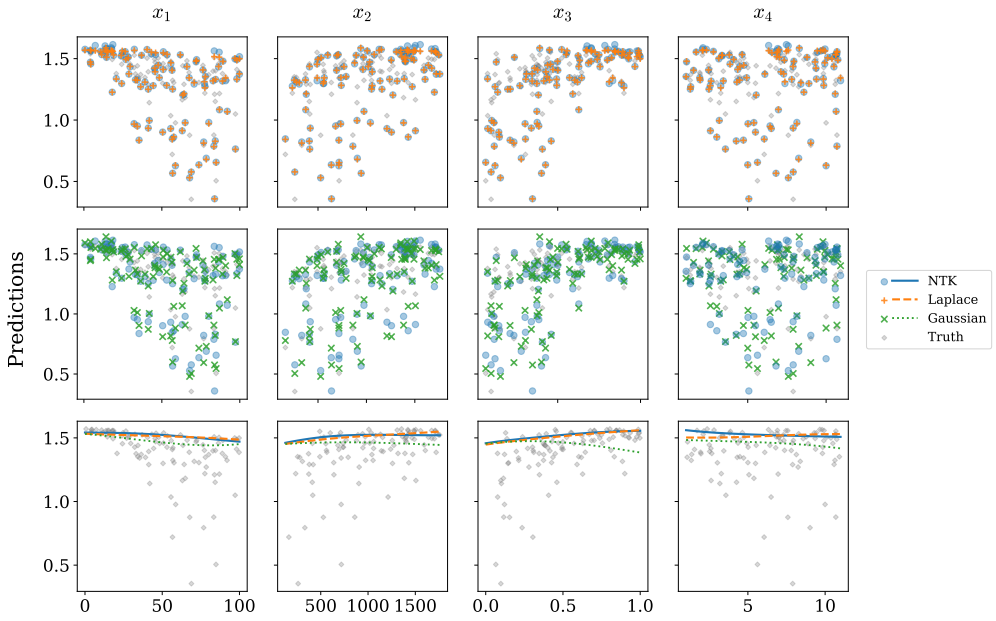

Figure 5.9: Predictions for non-noisy Friedman 3 in $\mathbb{S}^3$ for $D = 2$ with $X_{rescale}$ and $\mathbf{y}_{rescale}$. The rows of the figure are laid out as in Figure 5.6.

data. This can best be seen on the x_3 input in both rows. What is of interest is that the averaged prediction plots (row 3) for both figures appear to be very similar. That said, this is in-line with the plots seen in Section 5.3 where the posteriors trained in $\mathbb{R}^d$ were more related to the ground truth compared to posteriors trained in $\mathbb{S}^{d-1}$. We include Figures C.5 and C.6 in Appendix C which showcases the same setup for noisy Friedman 3.

Chapter 6

Real World Experiments

In this chapter we use the lessons we learned from Chapter 5 to evaluate the individual regression performance of the Laplace kernel, Gaussian kernel, and NTK individually on two real world datasets in a setting where the models are allowed to train to their own accord. In Section 6.1 we outline the experiment setup and datasets and in Section 6.2 we present our regression results.

6.1 Setup and Datasets

We setup our experiments with the findings in Chapter 5 in mind for the best possible GP fit. We summarize the experimental treatment in Table 6.1. The key differences are that we perform input rescaling ($X_{rescale}$) in all experiments. In addition, we split our data 75-25 into a training set $(X, \mathbf{y})$ and a testing set $(X_*, \mathbf{y}_*)$ respectively. Our experiments test inputs in $\mathbb{R}^d$ or $\mathbb{S}^{d-1}$ resulting in only 2 configurations per dataset. Since we are working with real world data, we assume that noise is present and apply the white noise kernel to our three kernels. We use R^2 and RMSE as our regression metrics.

Data	Description	Value
Normalization	Transforming inputs to $\mathbb{S}^{d-1}$	–
$X_{rescale}$	Input variable rescaling	`true`
Optimization	**Description**	**Value**
$n_{restart}$	Number of optimizer restarts	9
$\mathbf{y}_{rescale}$	Response variable rescaling during training	`true`
α	Value to ensure positive definiteness	10^{-5}
Kernel	**Description**	**Value**
D	NTK depth	$2, 3, 10$
β	NTK bias	Unfixed
ν	Matérn kernel smoothness	$\frac{1}{2}, \infty$
ℓ_{Lap}, ℓ_{Gaus}	Length-scale	Unfixed
$c_{NTK}, c_{Lap}, c_{Gaus}$	Constant value	Unfixed
$\sigma^2_{NTK}, \sigma^2_{Lap}, \sigma^2_{Gaus}$	Data noise variance	Unfixed

Table 6.1: Summary of variables managed in all real world experiments.

Define : $n_{restart}$, $\mathbf{y}_{rescale}$, α, $noise$

Input : $\theta, X, \mathbf{y}, X_*$ (Train and test data pre-processed the same way)

$k(\theta) \leftarrow c \cdot k(\theta) + \sigma^2 \delta_{\mathbf{s}=\mathbf{t}}$

$f_{k_{opt}}, \theta_{opt} \leftarrow \text{OPTIMIZE}\{f_k \sim \mathcal{GP}(\, 0, k(\theta)\,)|_{X,\mathbf{y}} : n_{restart}, \mathbf{y}_{rescale}, \alpha\}$

$\bar{\mathbf{f}}_{k_*} \leftarrow f_{k_{opt}}(X_*)$

Return: $\bar{\mathbf{f}}_{k_*}$

Algorithm 6.1: Experiment procedure for fitting a GP to real world data. OPTIMIZE refers to the GP optimization process of maximizing the marginal log likelihood.

Data Name	Task	d	n
Concrete	Determine concrete compressive strength	8	1030
Fire	Determine area of forest burned during fire	4	517

Table 6.2: Summary of real world data.

Furthermore, we simplify the training procedure as we are not interested in posterior mean matching. The three kernels considered follow the same GP fitting and predicting procedure outlined in Equation (2.13) from Section 2.2. We summarize the pseudocode in Algorithm 6.1.

We utilize two real world datasets: the concrete compressive strength dataset [51] and the forest fire dataset [14]. We provide a brief summary of the data in Table 6.2. The Concrete dataset contains 1030 observations and 8 features which include 7 features describing the concrete's ingredients (kg/m^3) and 1 feature describing its age (days). The Fire dataset contains 517 observations and a total of 12 features. We drop all the features except for the 4 features describing the weather conditions (temperature (°C), relative humidity (%), wind (km/h), and rain (mm/m^2)). This is because Cortez and Morais [14] found that these features provided the best regression performance given their metrics and we found that additional features would result in kernels that would zero out resulting in meaningless posteriors. It should be noted that we also take the advice of the authors of the Fire dataset and transform the response variable "area" using a log transform since it is heavily right-skewed: $y_{area} := \log(y + 1)$.

6.2 Results

We find through our experiments that the NTK attains comparable R^2 values to the Laplace and Gaussian kernels for the Concrete dataset as seen in Table 6.3. In fact, we attain values of $R^2 \approx 0.9$ indicating a very close fit to the ground truth. There is only a minute difference between experiments done in $\mathbb{R}^8$ versus $\mathbb{S}^7$. We illustrate the concrete compression predictions in $\mathbb{S}^7$ in Figure 6.1. We note that the Laplace and Gaussian GP predictions seem to align closely to the NTK despite being trained separately.

On the other hand, the Fire dataset fails to produce predictions with R^2 values that are better than the baseline mean. In fact, the R^2 values are significantly worse than the baseline but do show an improvement with higher NTK depth as seen in

			NTK			Lap	Gaus
Data	Metric	Space	$D = 2$	$D = 3$	$D = 10$		
Concrete	RMSE	$\mathbb{R}^8$	4.9562	5.0614	5.6477	5.3210	5.3058
		$\mathbb{S}^7$	5.3571	5.2906	5.5710	5.4368	5.1869
	R^2	$\mathbb{R}^8$	0.9041	0.9000	0.8754	0.8894	0.8901
		$\mathbb{S}^7$	0.8879	0.8907	0.8788	0.8846	0.8949
Fire	RMSE	$\mathbb{R}^4$	78.302	78.278	78.262	78.415	78.449
		$\mathbb{S}^3$	78.302	78.277	78.262	78.456	78.462
	R^2	$\mathbb{R}^4$	-10594	-8415	-4154	-50077	-116771
		$\mathbb{S}^3$	-11031	-8539	-4154	-499246	-1254525

Table 6.3: Results for real world experiments in $\mathbb{R}^d$ and $\mathbb{S}^{d-1}$. Metrics for the Fire dataset are calculated by first inverting the log-transformation.

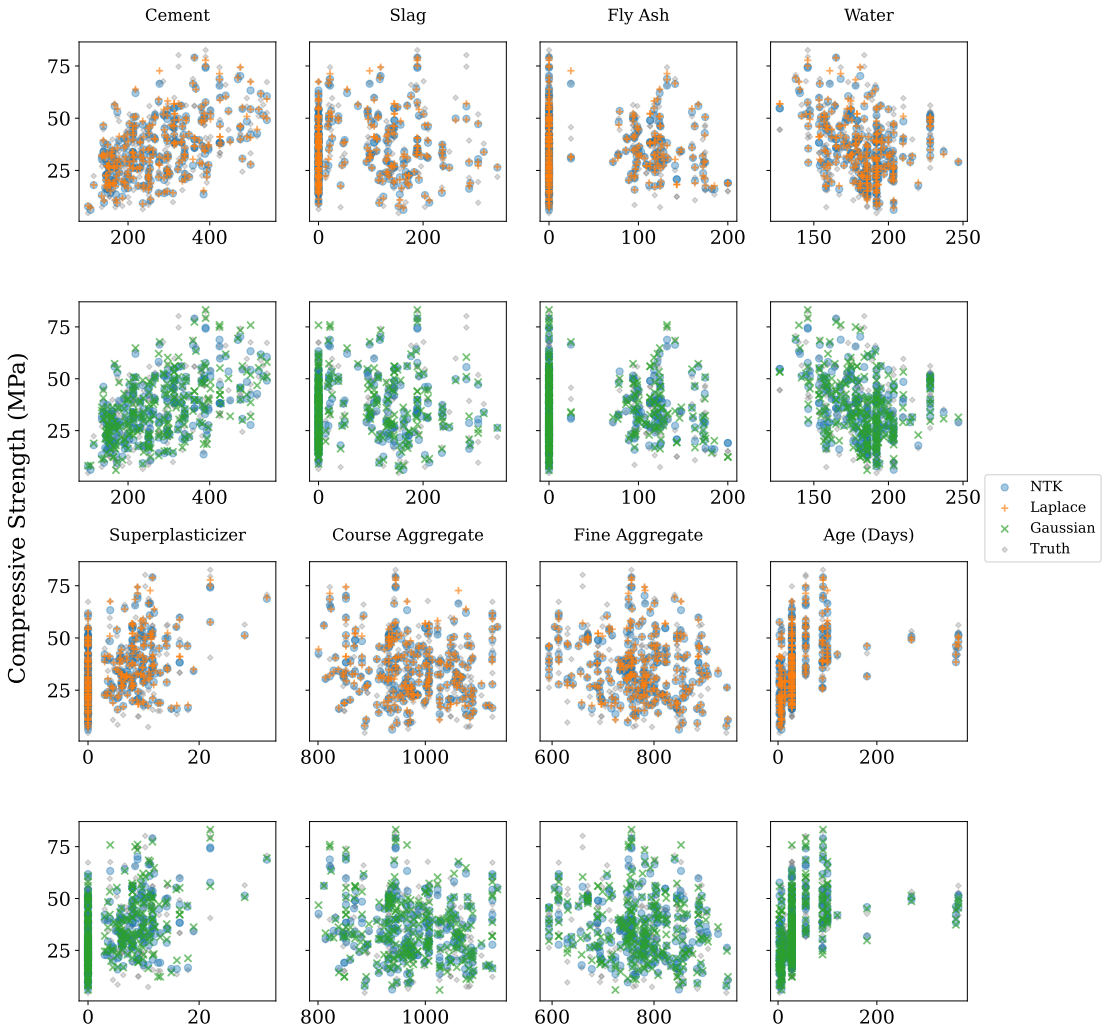

Figure 6.1: Concrete compression strength predictions over $\mathbb{S}^7$ with NTK depth $D = 10$ overlayed. Inputs are shown in $\mathbb{R}^8$ for visualization. The first and last two rows correspond to the input features all in kg/m^3 except for the Age.

Table 6.3. In Figure 6.2 we can see the area predictions for the NTK attempt to approximate the ground truth but does not capture the general landscape of the data. Upon further investigation this may be due to the kernel hyperparameter σ^2 overestimating the noise. We include Figure C.7 of the predictions without the white noise kernel applied in Appendix C. It is interesting to note that the Laplace and Gaussian predictions become constant. This is because the constant value completely zeros out for both kernels while length-scale becomes large. Lastly, we turn to the RMSE values for the Fire dataset. According to Cortez and Morais [14], their models attained RMSE values of approximately 64.7 whereas we get values close to 78 indicating a worse fit. Overall, this dataset presented a significantly difficult regression task.

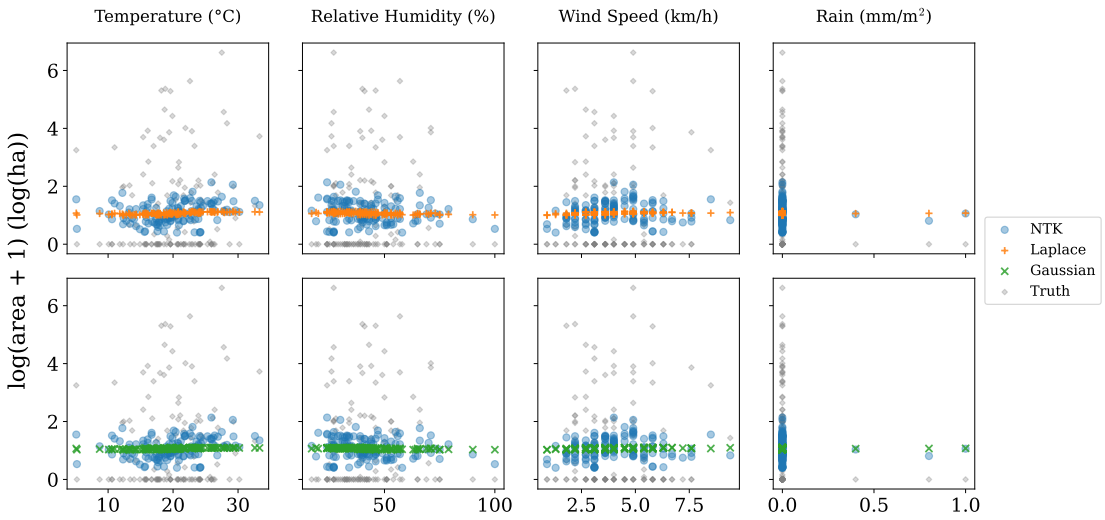

Figure 6.2: Fire area predictions over $\mathbb{S}^3$ with NTK depth $D = 10$ overlayed. Inputs are shown in $\mathbb{R}^4$ and output is log-transformed for visualization.

Chapter 7

Conclusion

In this work, we explored the empirical connections between the Laplace kernel and NTK, most notably, the direct relationship between the NTK's parameterization and how it can be reconciled with the parameterization of the Laplace kernel. We showed evidence for the importance of the bias parameter in equating the NTK to the Laplace kernel. Further, we developed a way to to reliably match the posterior means (elements of an RKHS) generated by the Laplace kernel and NTK GPs trained on various datasets. We further solidified the evidence that $\mathbb{R}^d$ is not a space in which the two kernels can be equal. Finally, we showcased that although the Gaussian kernel shares some similarities to the Laplace kernel, it fails to have the flexibility to match up with the other two kernels.

Though consequential, the NTK is a limited tool for machine learning tasks. As layers increase, it requires a high overhead for computation and runs into numerical issues during hyperparameter optimization due to its recursive nature. With the connections established to the Laplace kernel by Geifman et al. [21] and Chen and Xu [11], the NTK can be more easily applied to practical use and further motivate

additional work in kernel methods.

Future work to consider in this topic would be to fully develop the equality between two RKHSs. As discussed in Section 4.3, parameters of two kernels limit the functions a shared RKHS can include. As such, it would be interesting to explore how kernel parameters affect the resulting RKHS. The Laplace kernel and NTK are especially of interest since they have entirely different parameterizations yet still share the same space. In addition, it would be of interest to analyze the discretized nature of the NTK to see if it is possible to develop a direct parameter relation to the Laplace kernel other than pure optimization. Lastly, it is mathematically interesting to do a full treatment of the asymptotics of the NTK's bias parameter (Appendix B).

At the most essential level, RKHSs provide a robust theoretical framework for practical data modeling. For a full treatment of this topic we highly recommend the text by Berlinet and Thomas-Agnan [8]. For machine learning with Gaussian processes Williams and Rasmussen [47] give an in-depth overview of the topic. Lastly, Goodfellow et al. [24] serves as an excellent introduction to modern deep learning.

Bibliography

[1] Milton Abramowitz and Irene A. Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. US Government printing office, 1964.

[2] David Ackley. *A connectionist machine for genetic hillclimbing*, volume 28. Springer Science & Business Media, 2012.

[3] Sina Alemohammad, Zichao Wang, Randall Balestriero, and Richard Baraniuk. The recurrent neural tangent kernel. In *International Conference on Learning Representations*, 2021.

[4] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

[5] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Russ R. Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[6] Sheldon Axler. *Measure, integration & real analysis*. Springer Nature, 2020.

[7] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In *International Conference on Machine Learning*, pages 541–549. PMLR, 2018.

[8] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2004.

[9] Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[11] Lin Chen and Sheng Xu. Deep neural tangent kernel and laplace kernel have the same rkhs. In *International Conference on Learning Representations*, 2021.

[12] Zixiang Chen, Yuan Cao, Quanquan Gu, and Tong Zhang. A generalized neural tangent kernel analysis for two-layer neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, pages 13363–13373. Curran Associates, Inc., 2020.

[13] Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. *Advances in neural information processing systems*, 22, 2009.

[14] Paulo Cortez and Aníbal de Jesus Raimundo Morais. A data mining approach to predict forest fires using meteorological data. *New Trends in Artificial Intelligence, Proceedings of the 13th EPIA*, pages 512–523, 2007.

[15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[16] Simon S. Du, Kangcheng Hou, Russ R. Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[17] Richard M. Dudley. *Real analysis and probability*. CRC Press, 2018.

[18] Richard Franke. A critical comparison of some methods for interpolation of scattered data. Technical report, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 1979.

[19] Jerome H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19(1):1–67, 1991.

[20] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[21] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Basri Ronen. On the similarity between the laplace and neural tangent kernels. *Advances in Neural Information Processing Systems*, 33:1451–1461, 2020.

[22] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Reproducing kernel hilbert space, mercer's theorem, eigenfunctions, nyström method, and use of kernels in machine learning: Tutorial and survey. *arXiv preprint arXiv:2106.08443*, 2021.

[23] Federico Girosi and Tomaso Poggio. Networks and the best approximation property. *Biological cybernetics*, 63(3):169–176, 1990.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[25] Yuying Bella Guan. Introduction to gaussian processes for regression. Master's thesis, California State Polytechnic University, Pomona, May 2020.

[26] Trevor J. Hastie and Robert J. Tibshirani. *Generalized additive models*. Routledge, 2017.

[27] Ronald L. Iman, Jon C. Helton, and James E. Campbell. An approach to sensitivity analysis of computer models: Part i—introduction, input variable selection and preliminary variable assessment. *Journal of quality technology*, 13(3):174–183, 1981.

[28] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[29] Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K. Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences. *arXiv preprint arXiv:1807.02582*, 2018.

[30] Jaehoon Lee, Jascha Sohl-dickstein, Jeffrey Pennington, Roman Novak, Sam Schoenholz, and Yasaman Bahri. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1EA-M-0Z.

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

H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[32] Gonzalo Medina. Diagram of an artificial neural network, 2013. https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network.

[33] James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209:415–446, 1909. ISSN 02643952. URL http://www.jstor.org/stable/91043.

[34] Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *International Conference on Computational Learning Theory*, pages 154–168. Springer, 2006.

[35] Radford M. Neal. *Priors for Infinite Networks*, volume 118, chapter 2, pages 29–53. Springer Science New York, 1996.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[37] Walter Rudin. *Functional Analysis*. International series in pure and ap-

plied mathematics. McGraw-Hill, 1991. ISBN 9780070542365. URL https://books.google.com/books?id=Sh_vAAAAMAAJ.

[38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[39] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[40] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[41] Maxwell Stinchombe. Universal approximation using feed-forward networks with nonsigmoid hidden layer activation functions. *Proc. IJCNN, Washington, DC, 1989*, pages 161–166, 1989.

[42] George E. Uhlenbeck and Leonard S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[43] Michael Unser. A representer theorem for deep neural networks. *J. Mach. Learn. Res.*, 20(110):1–30, 2019.

[44] Christian J. Walder. *Efficient and Invariant Regularisation with Application to Computer Graphics*. PhD thesis, University of Queensland, January 2008.

[45] William J. Welch, Robert J. Buck, Jerome Sacks, Henry P. Wynn, Toby J. Mitchell, and Max D. Morris. Screening, predicting, and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[46] Christopher Williams. Computing with infinite networks. *Advances in neural information processing systems*, 9, 1996.

[47] Christopher K. I. Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.

[48] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[49] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[50] Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *Advances in Neural Information Processing Systems*, 32, 2019.

[51] I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998.

Appendices

Appendix A

Neural Tangent Kernel Gradient with respect to β

We derive the gradient used in optimizing the normalized recursive NTK's β bias parameter during GP training. We begin with the following proposition:

Proposition 2. *Let $\beta \geq 0$ and define the normalized recursive NTK using Definition 4.2.2 and Equation (4.13):*

$$\ddot{k}_{NTK}(L+1,\beta) = \frac{1}{(L+1)(\beta^2+1)} k_{NTK}(L+1,\beta) = \frac{1}{(L+1)(\beta^2+1)} \Theta^{(L)} \quad \text{(A.1)}$$

Then, the partial derivative with respect to β of $\ddot{k}_{NTK}(L+1,\beta)$ is defined as

$$\frac{\partial}{\partial\beta}\ddot{k}_{NTK}(L+1,\beta) = \frac{1}{(L+1)(\beta^2+1)} \left(\frac{\partial\Theta^{(L)}}{\partial\beta} - \frac{2\beta}{\beta^2+1}\Theta^{(L)} \right) \quad \text{(A.2)}$$

where

$$\frac{\partial\Theta^{(L)}}{\partial\beta} = 2\beta \left(\prod_{i=1}^{L} \Sigma^{(i)} \right) \left(1 + \sum_{i=1}^{L} \left(\prod_{j=1}^{i} \dot{\Sigma}^{(j)} \right)^{-1} \right) \quad \text{(A.3)}$$

Proving this Equation A.2 requires the use of the product rule $(uv)' = u'v + uv'$ where we will let $u = \frac{1}{(L+1)(\beta^2+1)}$ and $v = \Theta^{(L)}$ from Equation A.1. We will start with the substantially less involved task of deriving u followed by deriving v.

Proof. We begin with $\beta \geq 0$ and Equation A.1 and we set up the partial derivative with respect to β as follows:

$$\frac{\partial}{\partial\beta}\ddot{k}_{NTK}(L+1,\beta) = \frac{\partial}{\partial\beta}\left(\frac{1}{(L+1)(\beta^2+1)}\Theta^{(L)}\right)$$

We then let

$$u = \frac{1}{(L+1)(\beta^2+1)}$$
$$v^{(L)} = \Theta^{(L)}$$

so that we can utilize the product rule $(uv^{(L)})' = u'v^{(L)} + uv^{(L)'}$ in order to solve. We begin with finding the derivative of u:

$$\begin{aligned}
\frac{du}{d\beta} &= \frac{d}{d\beta}\left(\frac{1}{(L+1)(\beta^2+1)}\right) \\
&= \frac{d}{d\beta}\left((L+1)(\beta^2+1)\right)^{-1} \\
&= -\left((L+1)(\beta^2+1)\right)^{-2}(2\beta(L+1)) \\
&= -\frac{2\beta(L+1)}{(L+1)^2(\beta^2+1)^2} \\
\frac{du}{d\beta} &= -\frac{2\beta}{(L+1)(\beta^2+1)^2}
\end{aligned}$$

Which completes the derivative of u. Now we continue onto the partial derivative with respect to β for v which is defined recursively. As such, we begin with the base case and build our way towards the general case using the recursive formula from Definition 4.2.2:

$$\frac{\partial v^{(0)}}{\partial\beta} = \frac{\partial\Theta^{(0)}}{\partial\beta} = \frac{\partial}{\partial\beta}\left(\Sigma^{(0)} + \beta^2\right) = 2\beta$$

$$
\begin{aligned}
\frac{\partial v^{(1)}}{\partial \beta} = \frac{\partial \Theta^{(1)}}{\partial \beta} &= \frac{\partial}{\partial \beta}\left(\Theta^{(0)}\dot{\Sigma}^{(1)} + \Sigma^{(1)} + \beta^2\right) \\
&= \frac{\partial \Theta^{(0)}}{\partial \beta}\dot{\Sigma}^{(1)} + \frac{\partial}{\partial \beta}\Sigma^{(1)} + \frac{\partial}{\partial \beta}\beta^2 \\
&= 2\beta\dot{\Sigma}^{(1)} + 2\beta \\
&= 2\beta\left(\dot{\Sigma}^{(1)} + 1\right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial v^{(2)}}{\partial \beta} = \frac{\partial \Theta^{(2)}}{\partial \beta} &= \frac{\partial}{\partial \beta}\left(\Theta^{(1)}\dot{\Sigma}^{(2)} + \Sigma^{(2)} + \beta^2\right) \\
&= \frac{\partial \Theta^{(1)}}{\partial \beta}\dot{\Sigma}^{(2)} + \frac{\partial}{\partial \beta}\Sigma^{(2)} + \frac{\partial}{\partial \beta}\beta^2 \\
&= 2\beta\left(\dot{\Sigma}^{(1)} + 1\right)\dot{\Sigma}^{(2)} + 2\beta \\
&= 2\beta(\dot{\Sigma}^{(1)}\dot{\Sigma}^{(2)} + \dot{\Sigma}^{(2)} + 1)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial v^{(3)}}{\partial \beta} = \frac{\partial \Theta^{(3)}}{\partial \beta} &= \frac{\partial}{\partial \beta}\left(\Theta^{(2)}\dot{\Sigma}^{(3)} + \Sigma^{(3)} + \beta^2\right) \\
&= \frac{\partial \Theta^{(2)}}{\partial \beta}\dot{\Sigma}^{(3)} + \frac{\partial}{\partial \beta}\Sigma^{(3)} + \frac{\partial}{\partial \beta}\beta^2 \\
&= 2\beta\left(\dot{\Sigma}^{(1)}\dot{\Sigma}^{(2)} + \dot{\Sigma}^{(2)} + 1\right)\dot{\Sigma}^{(3)} + 2\beta \\
&= 2\beta\left(\dot{\Sigma}^{(1)}\dot{\Sigma}^{(2)}\dot{\Sigma}^{(3)} + \dot{\Sigma}^{(2)}\dot{\Sigma}^{(3)} + \dot{\Sigma}^{(3)} + 1\right) \\
&= 2\beta\left(\prod_{i=1}^{3}\dot{\Sigma}^{(i)} + \frac{\prod_{i=1}^{3}\dot{\Sigma}^{(i)}}{\dot{\Sigma}^{(1)}} + \frac{\prod_{i=1}^{3}\dot{\Sigma}^{(i)}}{\dot{\Sigma}^{(1)}\dot{\Sigma}^{(2)}} + \frac{\prod_{i=1}^{3}\dot{\Sigma}^{(i)}}{\prod_{i=1}^{3}\dot{\Sigma}^{(i)}}\right) \\
&= 2\beta\left(\prod_{i=1}^{3}\dot{\Sigma}^{(i)}\right)\left(1 + \frac{1}{\dot{\Sigma}^{(1)}} + \frac{1}{\dot{\Sigma}^{(1)}\dot{\Sigma}^{(2)}} + \frac{1}{\prod_{i=1}^{3}\dot{\Sigma}^{(i)}}\right) \\
&= 2\beta\left(\prod_{i=1}^{3}\dot{\Sigma}^{(i)}\right)\left(1 + \sum_{i=1}^{3}\left(\prod_{j=1}^{i}\dot{\Sigma}^{(j)}\right)^{-1}\right) \\
&\vdots \\
\frac{\partial v^{(L)}}{\partial \beta} = \frac{\partial \Theta^{(L)}}{\partial \beta} &= 2\beta\left(\prod_{i=1}^{L}\dot{\Sigma}^{(i)}\right)\left(1 + \sum_{i=1}^{L}\left(\prod_{j=1}^{i}\dot{\Sigma}^{(j)}\right)^{-1}\right)
\end{aligned}
$$

Thus, the partial derivative of v with respect to β is established. This leaves us to

establish the full partial derivative of $\ddot{k}_{NTK}(L+1,\beta)$ via the product rule:

$$\frac{\partial}{\partial\beta}\ddot{k}_{NTK}(L+1,\beta) = \frac{\partial}{\partial\beta}\left(uv^{(L)}\right) = \left(\frac{du}{d\beta}\right)\left(v^{(L)}\right) + (u)\left(\frac{\partial v^{(L)}}{\partial\beta}\right)$$

$$= \left(-\frac{2\beta}{(L+1)(\beta^2+1)^2}\right)\left(\Theta^{(L)}\right) + \left(\frac{1}{(L+1)(\beta^2+1)}\right)\left(\frac{\partial\Theta^{(L)}}{\partial\beta}\right)$$

$$= \frac{1}{(L+1)(\beta^2+1)}\left(-\frac{2\beta\Theta^{(L)}}{\beta^2+1} + \frac{\partial\Theta^{(L)}}{\partial\beta}\right)$$

$$= \frac{1}{(L+1)(\beta^2+1)}\left(\frac{\partial\Theta^{(L)}}{\partial\beta} - \frac{2\beta\Theta^{(L)}}{\beta^2+1}\right)$$

This completes the full derivation of the partial with respect to β for the normalized recursive NTK. □

Appendix B

Asymptotics of β

In this Appendix, we derive the limit of the NTK's parameter β when depth $D = 1$.

Proposition 3. *Let $\ddot{k}_{NTK}(1, \beta)$ be the shallow normalized neural tangent kernel as defined in Equation (4.13). Then the limit of $\beta \to \infty$ is as follows:*

$$\lim_{\beta \to \infty} \ddot{k}_{NTK}(1, \beta) = 2 - \frac{\arccos\left(\lambda^{(0)}\right)}{\pi} \tag{B.1}$$

Proof. We begin with the normalized NTK from Equation (4.13) with 1 hidden layer.

$$\ddot{k}_{NTK}(1, \beta) = \frac{1}{(\beta^2 + 1)} k_{NTK}(1, \beta) = \frac{1}{(\beta^2 + 1)} \Theta^{(1)} \tag{B.2}$$

Then, by substituting, we get

$$\begin{aligned}
\frac{1}{(\beta^2 + 1)} \Theta^{(1)} &= \frac{1}{(\beta^2 + 1)} \left(\Theta^{(0)} \dot{\Sigma}^{(1)} + \Sigma^{(1)} + \beta^2 \right) \\
&= \frac{1}{(\beta^2 + 1)} \left((\Sigma^{(0)} + \beta^2) \kappa_0 \left(\lambda^{(0)}\right) + \kappa_1 \left(\lambda^{(0)}\right) \sqrt{x^\top x z^\top z} + \beta^2 \right) \\
&= \frac{(\Sigma^{(0)} + \beta^2) \kappa_0 \left(\lambda^{(0)}\right)}{\beta^2 + 1} + \frac{\kappa_1 \left(\lambda^{(0)}\right) \sqrt{x^\top x z^\top z}}{\beta^2 + 1} + \frac{\beta^2}{\beta^2 + 1} \\
&= \frac{\Sigma^{(0)} \kappa_0 \left(\lambda^{(0)}\right)}{\beta^2 + 1} + \frac{\beta^2 \kappa_0 \left(\lambda^{(0)}\right)}{\beta^2 + 1} + \frac{\kappa_1 \left(\lambda^{(0)}\right) \sqrt{x^\top x z^\top z}}{\beta^2 + 1} + \frac{\beta^2}{\beta^2 + 1},
\end{aligned}$$

then by taking the limit with respect to β towards infinity:

$$\lim_{\beta\to\infty}\left(\frac{\Sigma^{(0)}\kappa_0\left(\lambda^{(0)}\right)}{\beta^2+1}+\frac{\beta^2\kappa_0\left(\lambda^{(0)}\right)}{\beta^2+1}+\frac{\kappa_1\left(\lambda^{(0)}\right)\sqrt{x^\top xz^\top z}}{\beta^2+1}+\frac{\beta^2}{\beta^2+1}\right)$$

$$\begin{aligned}
&= 0+\kappa_0\left(\lambda^{(0)}\right)+0+1\\
&= \frac{1}{\pi}(\pi-\arccos(\lambda^{(0)}))+1\\
&= 1-\frac{\arccos(\lambda^{(0)})}{\pi}+1\\
&= 2-\frac{\arccos(\lambda^{(0)})}{\pi}
\end{aligned}$$

thus completing the formulation. □

Appendix C

Additional Figures and Tables

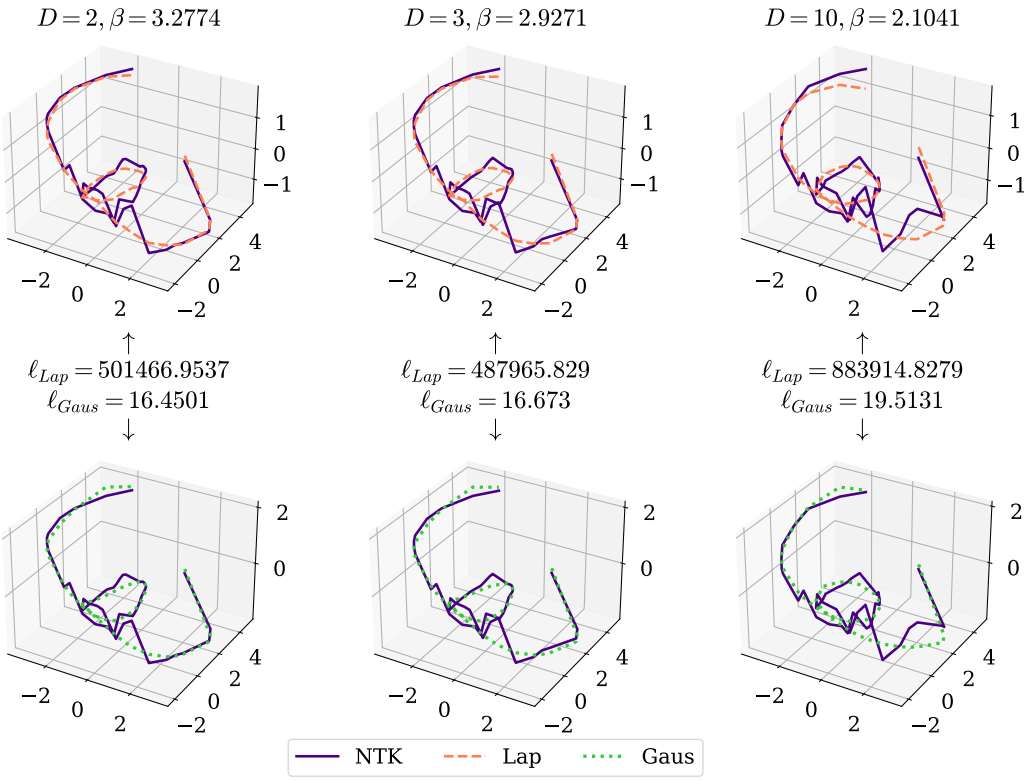

Figure C.1: Posterior means generated by fitting to data in $\mathbb{R}^2$ and predicted on out of sample data in $\mathbb{R}^2$. All kernels seem to be approximating the loop in the curve. The Laplace and Gaussian kernels provide almost the same predictions between them. In addition, the kernels seem to do a better job that $\mathbb{S}^1$ of approximating the underlying parametric curve. *Top:* NTK with Laplace kernel overlayed. *Bottom:* NTK with Gaussian kernel overlayed.

			$D = 2$		$D = 3$		$D = 10$	
Metrics	Dataset	Noise	Lap	Gaus	Lap	Gaus	Lap	Gaus
RMSE	Ackley	No	≈ 0	1.7505	0.0001	1.7505	0.0001	1.7504
		Yes	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
	Franke	No	0.0101	0.1413	0.0030	0.1412	0.0001	0.1408
		Yes	0.0046	0.0166	0.0028	0.0151	0.0031	0.0155
	Nonpoly	No	≈ 0	1.5981	≈ 0	1.5975	0.0001	1.5955
		Yes	0.0216	0.1924	0.0141	0.1866	0.0137	0.1685
ρ	Ackley	No	≈ 1	0.3526	≈ 1	0.3526	≈ 1	0.3526
		Yes	0.9926	0.9028	0.9968	0.8939	0.9985	0.8738
	Franke	No	0.9990	0.7782	0.9999	0.7757	≈ 1	0.7766
		Yes	0.9997	0.9943	0.9999	0.9952	≈ 1	0.9951
	Nonpoly	No	≈ 1	0.7492	≈ 1	0.7495	≈ 1	0.7502
		Yes	0.9999	0.9871	0.9999	0.9878	≈ 1	0.9911

Table C.1: Posterior mean matching results for the 2D input surface datasets in $\mathbb{S}^1$. As noted in Section 5.3, the noisy the Ackley function was difficult to properly fit resulting in a constant and meaningless posterior thus the zero RMSE should be looked at skeptically.

		Non-noisy				Noisy			
D	Sp.	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both
2	$\mathbb{S}^9$	-0.069	0.852	-0.071	0.855	0.136	0.849	0.126	0.852
	$\mathbb{R}^{10}$	0.229	0.855	0.240	0.935	0.232	0.852	0.242	0.935
3	$\mathbb{S}^9$	-0.070	0.851	-0.071	0.855	0.137	0.850	0.127	0.852
	$\mathbb{R}^{10}$	0.226	0.927	0.235	0.933	0.229	0.925	0.238	0.932
10	$\mathbb{S}^9$	-0.072	0.815	-0.073	0.822	0.128	0.814	0.129	0.821
	$\mathbb{R}^{10}$	0.218	0.810	0.221	0.894	0.221	0.873	0.224	0.892

Table C.2: Friedman 2 R^2 results for NTK posterior means with training done using various data transformations.

		Non-noisy				Noisy			
D	Sp.	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both	None	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both
2	$\mathbb{S}^9$	-0.198	0.023	-0.198	0.033	0.063	0.044	0.075	0.117
	$\mathbb{R}^{10}$	-0.187	0.667	-0.187	0.692	0.057	0.317	0.058	0.378
3	$\mathbb{S}^9$	-0.200	0.040	-0.200	0.055	0.064	0.025	0.066	0.125
	$\mathbb{R}^{10}$	-0.191	0.659	-0.190	0.696	0.051	0.313	0.117	0.374
10	$\mathbb{S}^9$	-0.211	0.209	-0.211	0.260	0.024	0.039	0.029	0.177
	$\mathbb{R}^{10}$	-0.195	0.532	-0.207	0.735	0.022	0.280	0.025	0.431

Table C.3: Friedman 3 R^2 results for NTK posterior means with training done using various data transformations.

	Non-noisy				Noisy			
D	N/a	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both	N/a	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both
2	≈ 1	0.9995	≈ 1	0.9995	0.9999	0.9993	0.9998	0.9993
3	≈ 1	0.9998	≈ 1	0.9998	≈ 1	0.9998	0.9999	0.9997
10	≈ 1	0.9996	≈ 1	≈ 1	≈ 1	0.9996	≈ 1	≈ 1

Table C.4: Friedman 2 ρ results for Laplace kernel and NTK posterior mean matching in $\mathbb{S}^9$ with training done using various data transformations.

	Non-noisy				Noisy			
D	N/a	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both	N/a	$X_{rescale}$	$\mathbf{y}_{rescale}$	Both
2	≈ 1	0.9993	≈ 1	0.9993	≈ 1	0.9997	≈ 1	0.9980
3	≈ 1	0.9999	≈ 1	0.9999	≈ 1	0.9999	≈ 1	0.9991
10	≈ 1	0.9983	≈ 1	0.9997	≈ 1	0.9953	≈ 1	0.9993

Table C.5: Friedman 3 ρ results for Laplace kernel and NTK posterior mean matching in $\mathbb{S}^9$ with training done using various data transformations.

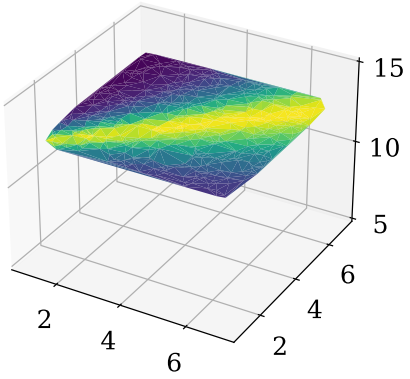

Figure C.2: NTK posterior mean of the noisy the Ackley function in $\mathbb{S}^1$ for NTK depth $D = 2$. The GP was trained using $n = 500$ inputs $x_1, x_2 \in [1, 7]$ generated using Latin hypercube sampling. The y values are all concentrated around ≈ 12.67 with a difference between the minimum and maximum being $\approx 10^{-8}$ indicating that the posterior mean has zeroed out. This is due to the kernel's constant value optimizing close to zero. Attempting to manually fit the GP while controlling the constant value and white noise provides similar results.

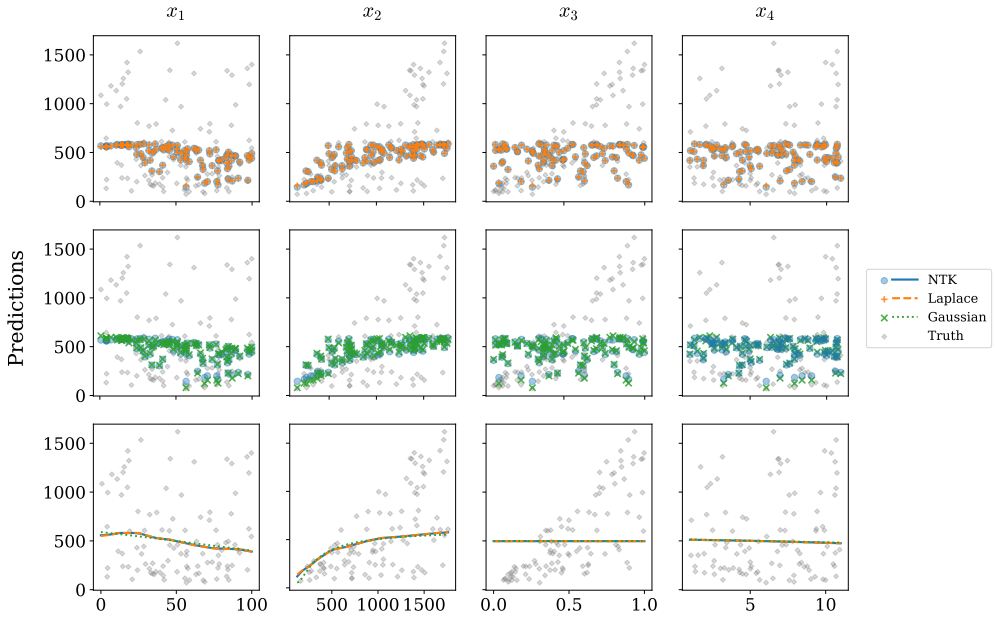

Figure C.3: Predictions for noisy Friedman 2 in $\mathbb{S}^3$ for $D = 2$ with $\mathbf{y}_{rescale}$. *Top:* NTK and Laplace predictions overlayed. *Middle:* NTK and Gaussian predictions overlayed. *Bottom:* Averaged prediction plots of all kernels.

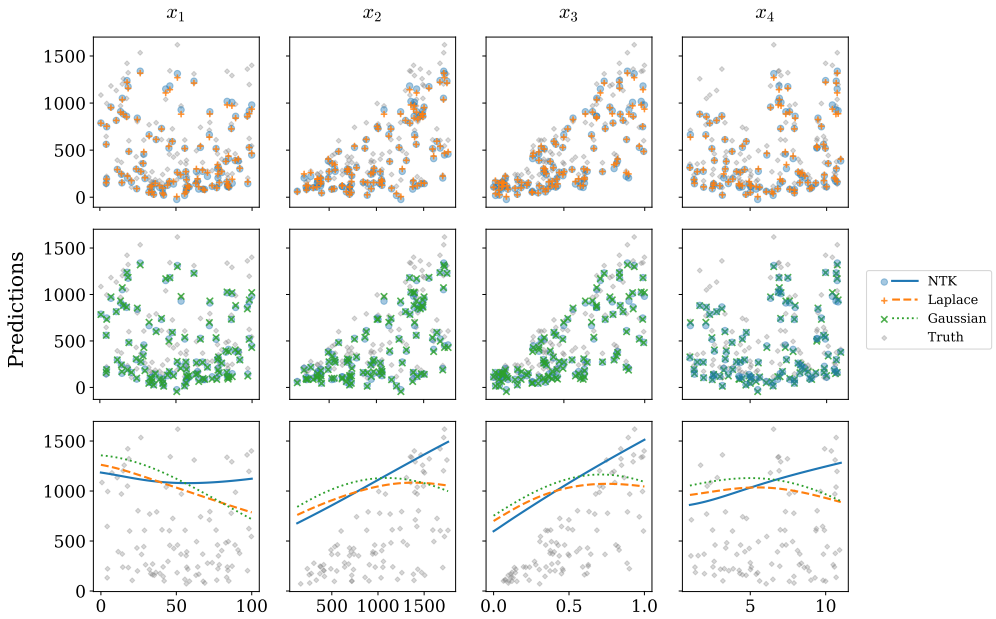

Figure C.4: Predictions for noisy Friedman 2 in $\mathbb{S}^3$ for $D = 2$ with $X_{rescale}$ and $\mathbf{y}_{rescale}$. The rows of the figure are laid out as in Figure C.3.

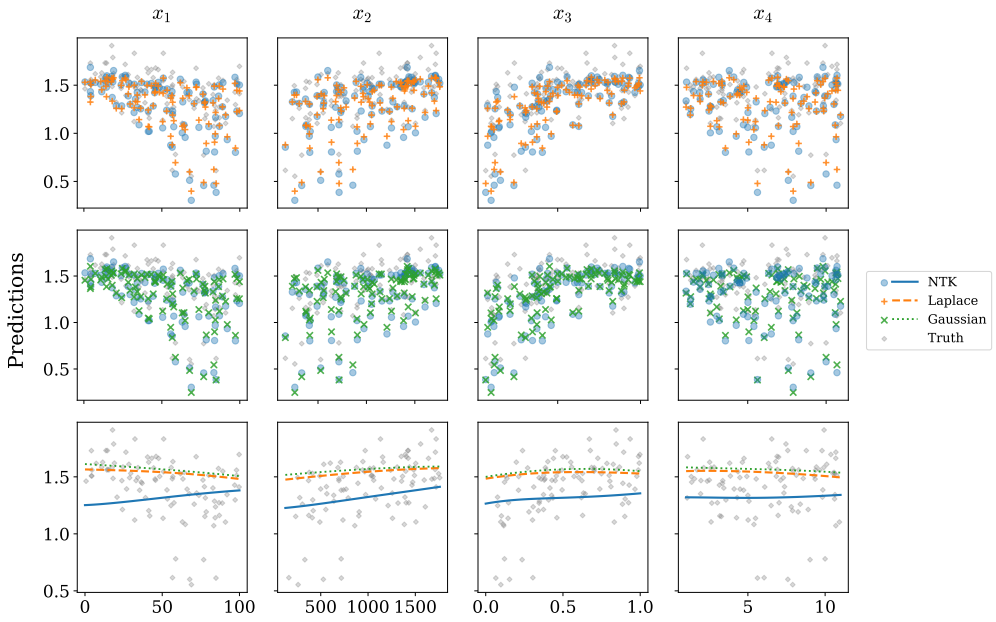

Figure C.5: Predictions for noisy Friedman 3 in $\mathbb{R}^4$ for $D = 2$ with $X_{rescale}$ and $\mathbf{y}_{rescale}$. The rows of the figure are laid out as in Figure C.3.

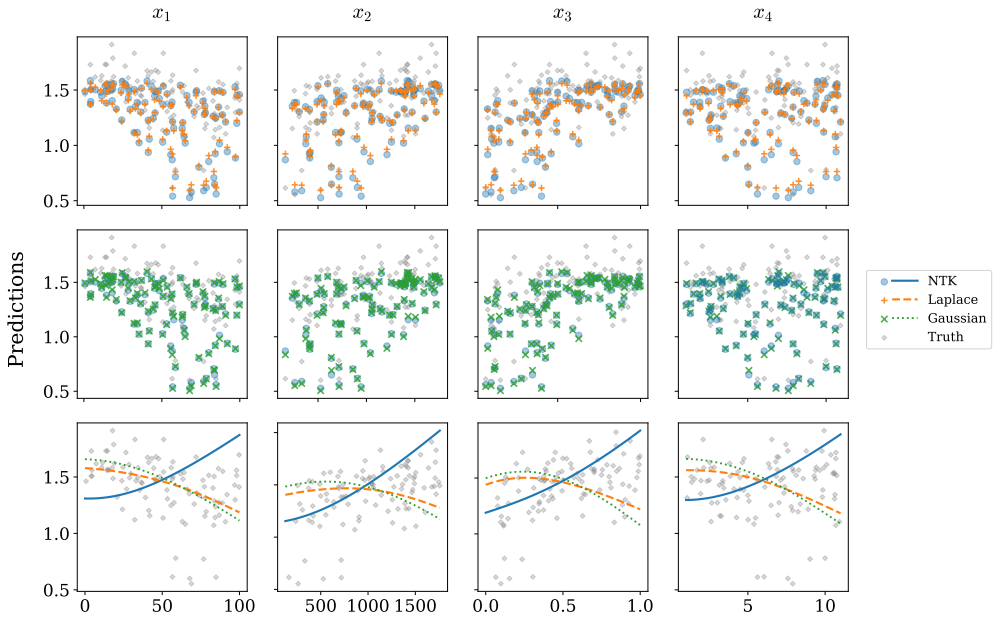

Figure C.6: Predictions for noisy Friedman 3 in $\mathbb{S}^3$ for $D = 2$ with $X_{rescale}$ and $\mathbf{y}_{rescale}$. The rows of the figure are laid out as in Figure C.3.

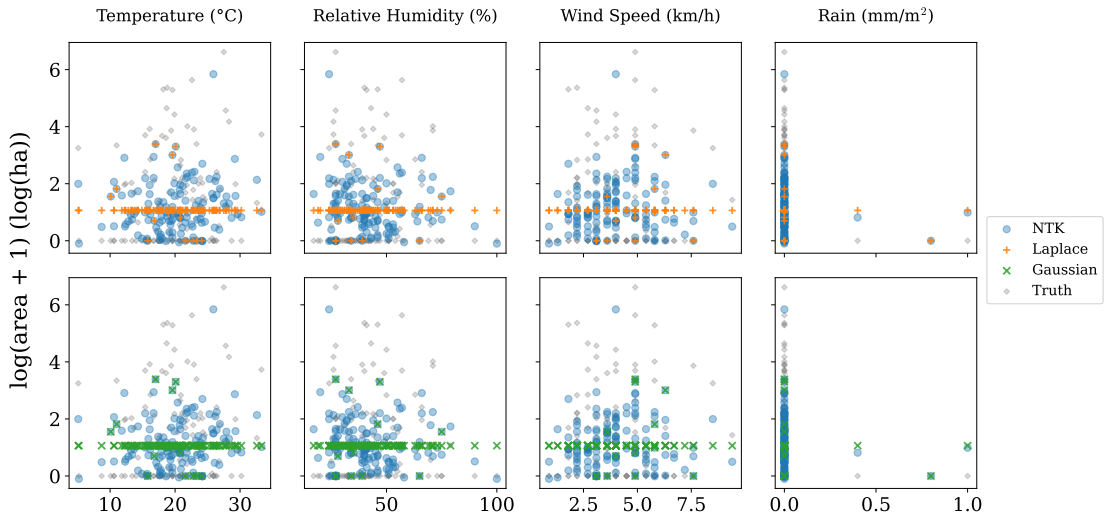

Figure C.7: Fire area predictions over $\mathbb{S}^3$ without white noise term with NTK $D = 10$. Inputs are shown in $\mathbb{R}^4$ and output is log-transformed for visualization. This is interesting since the NTK seems to do a better job of aligning the predictions to the ground truth in comparison to the GPs fit with a white noise term in Figure 6.2.