REPRODUCING KERNEL HILBERT SPACES AND FEATURE LEARNING IN NEURAL NETWORKS

NOAH MARSHALL

Department of Mathematics and Statistics
FACULTY OF SCIENCE
McGill University, Montreal

JANUARY 2023

A thesis submitted to McGill University
in partial fulfillment of the
requirements of the degree of
Master of Science

ABSTRACT

Abstract

Neural networks allow one to learn a set of semantically meaningful features in order to classify or regress to observed samples. Given this, neural networks may be viewed as the composition of a learned set of features with a linear function. It may also be shown that reproducing kernels are associated with a feature functions. In this thesis, we explore the utility of this connection. We give an introduction to reproducing kernel Hilbert spaces (RKHS) from a feature learning perspective. We review the literature connecting neural networks to RKHS. Finally, we introduce the neural feature kernel and show how it can be used to approximate an RKHS norm of a neural network, and how one may estimate its Mercer decomposition. We then show that the Mercer decomposition provides an importance score to the features of a neural network, giving insight into the predictions made by a network.

ABRÉGÉ

Abstract

Les réseaux neuronaux permettent l'apprentissage des caractéristiques saillantes des données observées afin d'effectuer la classification ou la régression. Vus de cette manière, ces réseaux sont la composition d'un ensemble de caractéristiques apprises avec une fonction linéaire. On peut également montrer que les noyaux reproduisants sont associés aux fonctions qui produisent des caractéristiques. Nous donnons une introduction aux espaces de Hilbert à noyau reproduisant (RKHS) sous la perspective de l'apprentissage de caractéristiques. Nous passons en revue la littérature qui relie les réseaux neuronaux aux RKHS. Enfin, nous présentons le noyau de caractéristiques neuronales et montrons comment il peut être utilisé pour approximer une norme RKHS d'un réseau neuronal et comment on peut estimer sa décomposition de Mercer. Nous montrons ensuite que la décomposition de Mercer fournit un score pour chacun des caractéristiques d'un réseau neuronal, donnant un aperçu des prédictions produises par un réseau.

LIST OF FIGURES

CONTENTS

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my incredible wife, Manpreet. She is the bedrock of my life, and none of this would have been possible without her love and support. I'm forever grateful for how she disrupted her life in order to allow me this opportunity.

I would also like to thank my supervisor, Dr. Adam Oberman, for the many useful meetings, literature recommendations, and ideas scribbled on blank printer paper. I would like to thank Dr. Tiago Salvador, who always knew the next question to ask, or the next experiment to run. I would like to thank my friend and fellow student, KC Tsiolis, for the many great discussions and his invaluable help in editing this thesis. I must also thank my lab mates; I'm honoured to work with such a collection of bright people. I would like to thank Dr. Warren Hare and Dr. Rebecca Tyson for preparing me for the transition to graduate school. Rebecca, I'll always be grateful for the opportunity to work with you during the chaos of the coronavirus pandemic.

I owe my penchant for learning to my parents, particularly my Mom. She has made education a priority since day one and has always been relentlessly supportive. I've got to thank my sister, Quinn, as well for being my best friend since she was born. Finally, I would like to thank all of my friends. Whether playing D&D, playing squash, or simply getting a drink, these moments make life all the more enjoyable.

I owe who and where I am to everyone who has supported me. Thank you all.

CHAPTER 1

INTRODUCTION

Broad interest in machine learning from the general public, popular culture, governments, and business has exploded in recent years [21], with the beginning of this explosion corresponding roughly to the breakthrough success of neural networks in the 2012 ImageNet Large Scale Visual Recognition Challenge [78, 52]. This interest has created a positive feedback loop in which the development of new deep learning technologies has enabled the discovery of new techniques leading to the further success of neural networks [69, 57, 16, 72]. Aided by these tools, neural networks have contributed to significant advances in natural language processing [86], drug design and discovery [71], robotics [84], and protein folding [48], among many others. Machine learning is a very broad term referring generally to the set of methods and algorithms used to recognize patterns from data in an automated fashion. Machine learning using neural network based methods is broadly referred to as deep learning. The term deep learning specifically refers to learning with neural networks of many layers; however, it is common to refer to any neural network method as a deep learning method as modern neural networks are typically large and complex.

Much of the success of neural networks stems from their ability to learn representations of data which can then be used for downstream tasks. By learning a semantically meaningful representation of the data, a neural network can learn relationships between the data and encode those relationships in a vector space. The subfield of deep learning that explicitly aims to embed the input data into some vector space such that the representation is semantically meaningful is known as representation or feature learning. Neural networks are uniquely capable among machine learning methods in their ability to learn

meaningful features from a wide variety of data. This is discussed further in Section 1.3.

However, neural networks are not alone in their use of features. Kernel machines — learned functions that lie in a reproducing kernel Hilbert space (RKHS, see Chapter 2) — implicitly compare the similarity of features extracted from an input to the features of known examples. However, these features are not learned as they depend on the kernel itself, which must be specified by the practitioner. The fact that both kernel machines and neural networks use features to represent their input data suggests a connection between them. Indeed, the connection between neural networks and kernel machines is far from new and has been known since at least 1994 [63]. There has been a recent resurgence of interest in this connection, which promises to give insight into the training dynamics [47], generalization behaviour [9], and expressivity [25] of neural networks among other things.

As neural networks are relatively new and complicated models, much work remains to be done to fully understand their behaviour. One aspect of neural networks that has surprised researchers is generalization ability. Classical statistical learning theory suggests that complex, highly parameterized models such as neural networks will predict poorly when presented with new examples as a consequence of being over-reliant on a limited set of known examples (see Section 1.1) [39, p. 221]. Yet, in practice, neural networks generalize quite well [94]. This apparent mismatch between theory and reality is an important question for the deep learning community to address. It appears likely that a connection between neural networks and kernel machines can be used in order to address this knowledge gap. There exist useful bounds on the complexity of the RKHS associated with a particular kernel machine which can be used to estimate the worst-case generalization error [60]. When classical theory is not useful — such as when working with highly complex models — it has been shown that kernel machines can exhibit the same behaviour as neural networks [9]. That this behaviour — overfitting the training data while also generalizing strongly — is evident in kernel machines suggests that the large body of RKHS theory will be useful in understanding the behaviour of neural networks.

Overall, there is a strong interest in better understanding the link between neural networks and kernel machines.

In this thesis, we argue that a neural network should be viewed as a learned feature function φ_θ which allows us to study the kernel and associated RKHS $\mathbb{H}$ formed by $\langle \varphi_\theta(x), \varphi_\theta(z) \rangle$, which we term the feature kernel. By taking this view, we may use the existing RKHS theory to study the function space $\mathbb{H}$ and its properties. We give background on relevant RKHS theory and connect it to our view of neural networks by introducing it from a perspective of the features induced by kernel functions. Additionally, we survey the existing literature exploring the connection between kernels and neural networks including: neural networks of infinite width [63, 56, 90] including the neural tangent kernel [47], overparameterized models [8, 9, 10], along with a collection of other approaches [25, 95, 38, 97, 53, 32]. Finally, we apply RKHS theory to evaluate the norms of neural networks in the RKHS induced by their feature kernel. Following this, we show how a spectral decomposition of the feature kernel may be performed to yield eigenvalues that serve as importance scores for a set of features. This is followed by numerical experiments illustrating and confirming the veracity of our theory.

1.1 Learning Theory

The general form of the learning problem consists of learning how to associate pairs of data $(x, y) \sim \mathcal{D}$ sampled from a joint distribution $\mathcal{D}$. Here, $x \in \mathcal{X}$ is an observation and $y \in \mathcal{Y}$ is an associated response that will depend on the task at hand. For instance, in a regression context it is likely that $\mathcal{Y} = \mathbb{R}$, or for k-class classification $\mathcal{Y} = \{1, \ldots, k\}$. Examples of a learning problem include: classifying objects in a photo, responding appropriately to a text-based inquiry, or predicting how toxic a molecule is likely to be. In the case of image classification, x is given by the photo's pixel values and y is the true label of the photo (according to some human labeller).

Often, problems will be categorized as supervised or unsupervised. In supervised

learning, the response variable associated with each observation is known; the response variable is unknown in unsupervised learning. The response variable may be unavailable because it is impossible to get. Perhaps one is trying to classify photos of unknown animal species. Alternatively, obtaining the response could be prohibitively costly. Hiring radiologists to classify a hundred thousand images of potential breast cancer tumours would be awfully expensive. In this work, we will typically focus on supervised learning objectives.

In an ideal world, a learner would have perfect knowledge of the data generating distribution $\mathcal{D}$. However, in practice, we must work with an empirical estimation $\hat{\mathcal{D}}$ derived from a dataset of observed samples. Because of this estimation, there is a gap in the predictive performance of the best model on $\hat{\mathcal{D}}$ versus the same model on the true distribution $\mathcal{D}$. A model trained on $\hat{\mathcal{D}}$ which is nonetheless able to make accurate predictions on data generated from $\mathcal{D}$ is said to generalize. The generalization of a model is measured via the gap in performance between the model evaluated on $\hat{\mathcal{D}}$ and on $\mathcal{D}$. Generalization often depends on the complexity of the learned model. RKHS provide a set of tools to both measure and control the complexity of the functions they contain. Once RKHS are introduced in more detail, we examine their relevance to machine learning in Section 2.5.

For simplicity, learning theory will be described in the context of supervised binary classification. In this setting, we have a dataset of samples $S = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathcal{X}$, $y_i \in \{0, 1\}$, and each pair (x_i, y_i) is drawn from some joint distribution $\mathcal{D}$. The goal is to learn a function f, which can identify which class a given data point x is most likely to belong to. Given a candidate function f, we may measure how often it produces incorrect results via the risk functional — also known as the 0-1 loss.

Definition 1.1.1 (Risk Functional)**.** *Given a predictor f and a distribution of pairs of samples $\mathcal{D}$, the risk functional is given by*

$$\mathcal{R}[f] = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{1}_{f(x)\neq y}\right]. \tag{1.1}$$

The minimizer of (1.1) is known as the Bayes optimal classifier

$$f_{bayes}(x) = \underset{c\in[0,1]}{\operatorname{argmax}}\ p_{\mathcal{D}}(y = c|x), \tag{1.2}$$

where $p_{\mathcal{D}}$ indicates the reliance on $\mathcal{D}$. Of course, we do not know $\mathcal{D}$ and must make do with the empirical risk functional.

Definition 1.1.2 (Empirical Risk Functional)**.** *Given a predictor f and a dataset of n samples $S = \{(x_i, y_i)\}_{i=1}^n$, the empirical risk functional is given by*

$$\hat{\mathcal{R}}[f] = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{f(x_i)\neq y_i}. \tag{1.3}$$

Note that for a fixed f, the expectation of the empirical risk is equal to the true risk

$$\mathcal{R}[f] = \mathbb{E}_{S\sim\mathcal{D}^m}\left[\hat{\mathcal{R}}[f]\right].$$

The glaring issue with the above risk functionals is that they are highly discontinuous and thus difficult to minimize. Therefore, a surrogate functional that upper bounds the empirical risk functional is minimized in its place. This surrogate is known as a loss functional.

Definition 1.1.3 (Empirical Loss Functional)**.** *Given a predictor f, a loss function $c : \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}^+$ such that $c(y, y) = 0$, and a dataset of n samples $S = \{(x_i, y_i)\}_{i=1}^n$ the empirical loss functional is defined as*

$$\widehat{\mathcal{L}}[f] = \frac{1}{n}\sum_{i=1}^{n} c(f(x_i), y_i). \tag{1.4}$$

The loss function is nearly always required to be differentiable and convex as well.

Now, with an appropriate objective to measure the viability of a candidate function, the search space is restricted to a hypothesis set of functions $\mathcal{H}$. Thus, the objective is to find the function $f^* \in \mathcal{H}$ which minimizes the empirical loss functional (1.4)

$$f^* := \arg\min_{f \in \mathcal{H}} \widehat{\mathcal{L}}[f]. \tag{1.5}$$

Having found such an f^*, we turn our attention to studying the effect of the approximation error induced by the estimation of $\mathcal{D}$ with a finite dataset. This difference is known as the generalization gap and is defined as

$$|\mathcal{R}[f^*] - \hat{\mathcal{R}}[f^*]|. \tag{1.6}$$

Bounds on Equation 1.6 typically start by measuring the complexity of the hypothesis class $\mathcal{H}$. Although there are many such measures, we will focus on Rademacher complexity.

Definition 1.1.4 (Empirical Rademacher Complexity)**.** *Given a family of functions $\mathcal{H}$ mapping $\mathcal{X} \to [a, b] \subset \mathbb{R}$ and a dataset of n points $S = \{x_i\}_{i=1}^n$ the empirical Rademacher complexity of $\mathcal{H}$ with respect to S is defined as*

$$\hat{\mathfrak{R}}_S(\mathcal{H}) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i h(x_i) \right], \tag{1.7}$$

where $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_n]^T$ and each σ_i is a Rademacher random variable — a random value which uniformly takes the values ± 1.

Definition 1.1.5 (Rademacher Complexity)**.** *The Rademacher complexity of a family of functions $\mathcal{H}$ over samples of size $n \in \mathbb{N}$ is the expectation of the empirical Rademacher complexity* (1.7)

$$\mathfrak{R}_n(\mathcal{H}) = \mathbb{E}_{S \sim \mathcal{D}^n} \left[\hat{\mathfrak{R}}_S(\mathcal{H}) \right]. \tag{1.8}$$

The Rademacher complexity of a function class measures the ability of the function class to fit randomly labelled binary data. The intuition behind Rademacher complexity is

that functions capable of fitting randomly labelled data are likely to overfit the training data. The fact that the Rademacher complexity of a hypothesis class can be used to bound the generalization gap supports this intuition.

Theorem 1.1.6 (Error Bounds via Rademacher Complexity)**.** *Let $\mathcal{H}$ be a family of functions taking values in $\{\pm 1\}$. Let $\mathcal{D}$ be the distribution of samples over the input space $\mathcal{X}$. Then, for any $\delta > 0$ with probability at least $1 - \delta$ over a sample S of size n we have*

$$\mathcal{R}(h) \leq \hat{\mathcal{R}}_S(h) + \mathfrak{R}_n(\mathcal{H}) + \sqrt{\frac{\log \frac{1}{\delta}}{2n}} \tag{1.9a}$$

$$\mathcal{R}(h) \leq \hat{\mathcal{R}}_S(h) + \hat{\mathfrak{R}}_S(\mathcal{H}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2n}} \tag{1.9b}$$

Proof. The proof is given by [60, p.33]. □

As will be seen in Section 2.5, there exists known bounds of the Rademacher complexity of an RKHS. As neural networks are capable of perfectly fitting random noise [94], they have high Rademacher complexity making the above bound essentially vacuous when learning with a neural network. Formulating a neural network as a function in an RKHS could allow one to bound the generalization gap of a neural network using the bounds described later in Section 2.5.

1.2 Neural Networks

Neural networks are general purpose, highly parameterized functions used to approximate an unknown function through data. Neural networks are called such as they were initially inspired by the neurons of the human brain — they were first called *artificial* neural networks. They work to approximate functions through repeated linear and non-linear transforms of their input data. There are many variants of neural networks designed for computer vision [55], natural language processing [44, 87], playing games [85], and much more. One of the simplest variants of a neural network is known as a multi-layer perceptron

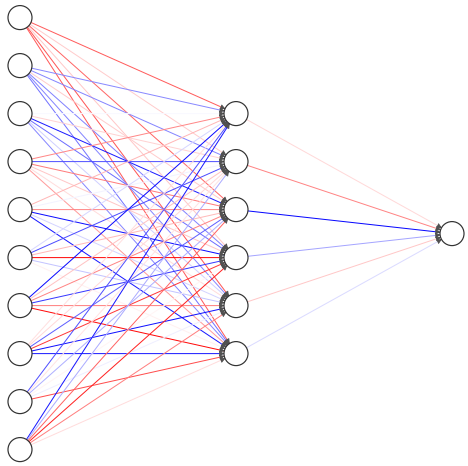


Figure 1.1: *A graphical representation of a multi-layer perceptron at initialization (Equation 1.10). The nodes are called neurons and the edges represent the weights connecting the nodes. The edges are coloured according to randomly chosen weights with dark blue representing highly negative weights and dark red representing highly positive weights. The second to last (and, in this case, middle) layer corresponds to the feature layer. Note that the activation functions are not shown but are applied element-wise to the sum of weights entering any particular node.*

— named after the Perceptron [59] — which consists of multiple iterations of matrix multiplication followed by the element-wise application of a non-linear, almost everywhere differentiable function σ — known as the activation function — to every layer but the very last. Traditionally σ has been taken to be $\sigma(x) = \tanh(x)$ or $\sigma(x) = (1 + \exp(-x))^{-1}$; however, modern work tends to use $\sigma(x) = \max(0, x)$ or slight variants of such. In principle, any almost everywhere differentiable, non-linear function will suffice and the activation function is chosen empirically. The equation for an MLP is

$$f_\theta(x) = W_L\sigma(\dots\sigma(W_2\sigma(W_1x + b_1) + b_2)\cdots) + b_L, \tag{1.10}$$

where θ represents the parameters of the neural network and is taken to be the collection of every element of the weights W_i and biases b_i for $i = 1, \dots, L$. It is typical to write a neural network, particularly in a classification context, as a feature function φ_θ followed by a linear classifier

$$f_\theta(x) = W_L\varphi_\theta(x) + b_L. \tag{1.11}$$

The fitting of the parameters θ of a neural network is typically done via variants of the gradient descent algorithm. In the machine learning literature, this optimization is known as training. The stochastic gradient descent (SGD) algorithm is the most popular variant used to train neural networks due to its simplicity and its effectiveness [77].

One appealing property of neural networks is that under certain assumptions, they can approximate any sufficiently well-behaved function [24, 6]. This result is known as universal approximation. Universal approximation combined with reliable optimization tools and well-maintained libraries of code to implement and train neural networks has resulted in deep learning being a highly active field of research that has achieved remarkable performance across many disciplines.

1.3 Feature Learning

Feature or representation learning refers to the problem of learning a map $\varphi : \mathcal{X} \to \mathbb{R}^d$ such that $\varphi(x)$ encodes the semantics of x in a meaningful way. Feature learning has become a subfield in its own right within the machine learning community, even having its own conference, ICLR, the International Conference on Learning Representations. Due to their expressivity and architecture, neural networks are naturally suited to such a learning objective. Their construction allows one to view them as progressively combining ever more intricate features layer-by-layer until the final layer, where the features are applied to the task at hand. This is the view expressed in Equation 1.11. This algorithmic learning of features transfers the effort of creating features away from experts handcrafting features [31] and towards the design of algorithms and more powerful computers. This allows algorithmic feature learning to scale much faster than expert-driven feature engineering while increasing the flexibility and expressive power of the learned features. It should be noted that feature learning is not unique to deep learning. For instance, word2vec uses a contrastive loss in order to solve a matrix factorization problem, giving a word representation depending on its context within a corpus of text [58].

The general implementation of feature learning is conceptually quite simple. One wishes to learn representations such that a mathematical notion of similarity (such as distance or cosine similarity) agrees with statements of similarity that a human may make about the same data. For instance, we may wish to learn a representation of objects such that an orange cat is more similar to a grey cat than a dog while also being more similar to the dog than to Swiss cheese.

Features may be extracted from generative models trained to sample from the data distribution [43, 88, 22, 66, 36, 95]. However, these approaches likely solve a harder problem than is necessary since representations need not capture low-level details such as pixel colour. Indeed, representations might be better formed by encapsulating the higher-level aggregate features such as shapes or objects.

Alternatively, the models may be trained via a pre-text task that is hoped to produce features that will be useful in downstream tasks [27]. Such tasks might include: image inpainting [70], colourization [54], predicting rotations [35], or predicting image patch locations [65]. However, the choice of pre-text task is important, and a pre-text task which is too dissimilar to the true objective can produce unsatisfactory features. Moreover, a pre-text task must be defined for various applications, which may be time-consuming and difficult. For example, at least to a lay deep learning practitioner, it is not obvious what pre-text task might produce good representations of chemical compounds.

Contrastive learning has emerged as a popular and now common framework [20, 67, 18, 37, 68, 5, 42, 17] to learn representations. It acts as a sort of general pre-text task where the challenge of defining the task is reduced to the challenge of determining which samples are similar (or dissimilar). The first contrastive learning paper sought to learn features where samples from the same class are close, and samples from different classes are distant [20]. This general idea still forms the core of most contrastive learning algorithms. However, one can replace the idea of the same and different classes with that of positive and negative pairs. A positive pair is a pair of samples the practitioner

expects to be represented similarly, while a negative pair is one the practitioner expects to be represented dissimilarly. Formulating contrastive learning in terms of positive and negative pairs allows one to algorithmically produce these pairs. For instance, one can take a positive pair to be a sampled image and a heavily modified version of that image (modified perhaps by blurring, cropping, adjusting the colour, etc.). The process of creating a positive pair through modification is known as data augmentation. The negative pair would simply be the original image along with a separate randomly sampled image. This approach, laid out in Chen et al. [18], remarkably allows the learning problem to be unsupervised. This formulation allows the method beyond the need for human-labelled datasets. Methods exist in which negative pairs are not required [37, 93], but require some other objective so that the representations do not collapse — if every representation is the same, similarity is always satisfied. Recent models have used text and image pairs to perform contrastive learning [73, 74, 64]. Making use of extensive datasets of captioned images, the authors use the text and image as positive pairs. Having learned good representations, the authors show that these representations can be used to help generate new images given a text prompt. It is remarkable that contrastive learning algorithms can produce unsupervised models with performance nearly on par with their supervised counterparts. This suggests something foundational about the concept of similarity to the learning process.

Another key benefit of feature learning is that once a good representation has been found, the same representation may then be used for many other downstream tasks. The process of generalizing across contexts is known as transfer learning. It is believed that representation learning algorithms have an advantage in transfer learning because they learn representations that capture underlying attributes of the data [11]. A simple example is as follows: a representation function φ_θ is trained to produce representations for photos of monster trucks and learns that the presence of a wheel is a useful attribute of the data. This representation would then be useful in tasks involving wheeled objects more

generally.

Further empirical support for the utility of feature learning is provided by the concept of foundation models [14]. A foundation model refers to a model trained with a tremendous amount of data and computation that is meant to be adapted to downstream tasks. A large amount of data is thought to force the model to learn good representations, which can then be used in other tasks with little to no additional training. The fact that foundation models can be used effectively suggests that neural networks can be trained to provide useful representations of data which can then be used in numerous tasks.

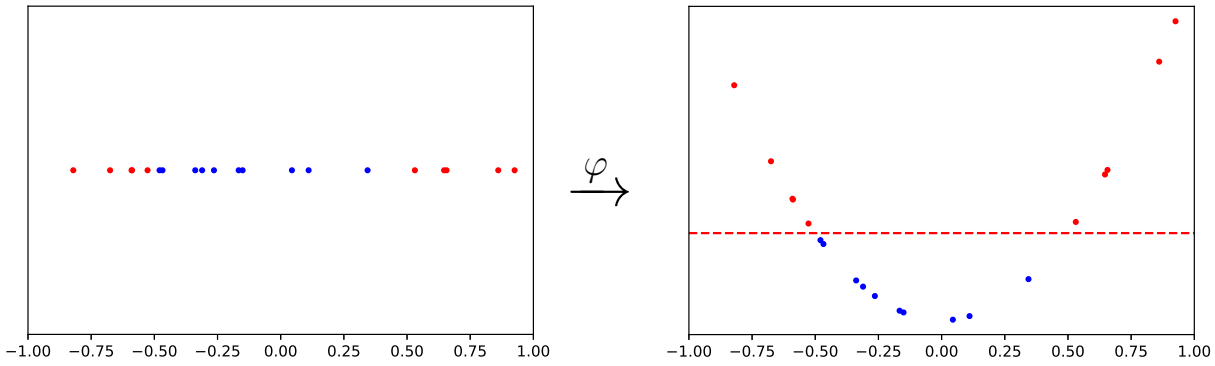
CHAPTER 2

REPRODUCING KERNEL HILBERT SPACES

2.1 HISTORICAL BACKGROUND

The study of reproducing kernel Hilbert spaces (RKHS) first arose in the early twentieth century as a result of Hilbert's study of Fredholm integral equations. [13, 82, 33]. The study of these spaces seems to have happened in parallel with the works of Zaremba [92] on boundary value problems and Mercer's study of positive definite functions within the study of integral equations [33].

The importance of this work was realized in the thesis of Bergmann [12] and later expanded through the various works of Aronszajn [2, 3, 4]. In the mid-twentieth century, Aizerman realized the broad applicability of kernels to machine learning; he pointed out that kernels enable what has become known as the kernel trick — the ability to replace an inner product with a kernel evaluation in order to make use of rich feature spaces induced by the kernel without added computational cost [1]. However, it was arguably the support vector machine (SVM) taking full advantage of the kernel trick that popularized the use of RKHS in machine learning [30, 15, 23]. Kernel theory was computationally attractive because a reproducing kernel function allows one to indirectly compute the inner product between a possibly infinite-dimensional representation of the kernel's arguments. The representations are induced by the choice of kernel. The inner product may be computationally costly or impossible to compute directly, yet the kernel allows this inner product to be calculated nonetheless. This representation can provide a richer set of features than the inputs themselves. For example, the indicator function $\mathbb{1}_{[-0.5,0.5]}$ is not learnable by a linear function. Yet, in a representation associated with the kernel $\mathcal{K}(x, z) = (1 + xz)^2$ it is (see Figure 2.1).



(a) *Indicator function,* $\mathbb{1}_{[-0.5,0.5]}$ **(b)** *Feature space,* $\mathcal{V}_{feat}$

Figure 2.1: *The indicator function* $\mathbb{1}_{[-0.5,0.5]}$ *is not linearly separable in* $\mathbb{R}$ (***left***)*; however, it becomes linearly separable when* $\mathbb{R}$ *is embedded with the feature function* $\varphi(x) = [1, x, x^2]$ (***right***)*. The feature function corresponds to the kernel* $\mathcal{K}(x, z) = \langle \varphi(x), \varphi(z) \rangle = (1 + xz)^2$*, known as the 2nd degree polynomial kernel.*

From a theoretical viewpoint, RKHS are attractive partially due to their associated norm, allowing one to regularize — loosely, to smooth out — an approximating function and partially because, as we will see, evaluation functionals applied to functions in an RKHS are bounded. This property ensures that we may use these function norms to find a convergent series of functions that continue to approximate training data without converging to uninformative solutions, as can happen with other function norms (see Section 2.4). While nowadays, SVMs have fallen somewhat out of fashion in the research community, being substituted for neural networks, kernel theory is very much an active area of research. In Chapter 3, we further detail modern work connecting kernel theory with neural networks.

2.2 REPRODUCING KERNEL HILBERT SPACES

This chapter will introduce the necessary background to understand the basic concepts of reproducing kernel Hilbert spaces, beginning more generally with the concept of inner product and Hilbert spaces. We then introduce reproducing kernels and RKHS before expanding on some of their fundamental properties. We frame our presentation of this

theory as much as possible around the induced feature spaces associated with each kernel.

Reproducing kernel Hilbert spaces are, first of all, Hilbert spaces. Hilbert spaces generalize the properties of Euclidean space which we use to model the world of everyday life. For this reason, Hilbert spaces have intuitive and simple properties while remaining a very rich and powerful concept. It is natural to try to work in a Hilbert space whenever an inner product and vector structure is available. With inner products come norms. Norms measure the size of an element within an inner product space, and in the context of an inner product on functions, the norm can serve as a measure of complexity. Thus in many machine learning contexts, we will seek to regularize an approximating function by controlling its norm in some Hilbert space.

Definition 2.2.1 (Inner Product Space)**.** *An inner product space $\mathcal{V}$ is a vector space equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{V}} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ such that*

1. $\langle f, f \rangle_{\mathcal{V}} \geq 0$ *for all* $f \in \mathcal{V}$ *and where* $\langle f, f \rangle_{\mathcal{V}} = 0$ *iff* $f = 0$
2. $\langle f, g \rangle_{\mathcal{V}} = \langle g, f \rangle_{\mathcal{V}}$ *for all* $f, g \in \mathcal{V}$
3. $\langle f, \alpha g + h \rangle_{\mathcal{V}} = \alpha \langle f, g \rangle_{\mathcal{V}} + \langle f, h \rangle_{\mathcal{V}}$ *for all* $f, g, h \in \mathcal{V}$ *and all* $\alpha \in \mathbb{R}$.

Inner products provide a natural way to measure the similarity of two points in a space. Consider the inner product $\langle u, v \rangle_{\mathcal{V}}$ between two vectors u, v with unit length. Then $\langle u, v \rangle_{\mathcal{V}} = \cos(\theta)$ where θ represents the angle in $\mathcal{V}$ between u and v. Thus the inner product provides a measure of alignment between the two vectors in the space $\mathcal{V}$ with: $\langle v, v \rangle_{\mathcal{V}} = 1$ indicating the vectors are perfectly aligned, $\langle v, v \rangle_{\mathcal{V}} = 0$ indicating they are orthogonal, and $\langle v, v \rangle_{\mathcal{V}} = -1$ indicating they are essentially opposites. This notion of similarity through inner products is a very important notion throughout machine learning [83, 18, 73, 81].

Note that inner products give rise to a norm $\|v\|_{\mathcal{V}} = \sqrt{\langle v, v \rangle_{\mathcal{V}}}$. Using this norm, we can define a Cauchy sequence as a sequence $\{f_n\}_{n=1}^{\infty}$ with all $f_n \in \mathcal{V}$ such that for all $\epsilon > 0$

there exists an $N \in \mathcal{N}$ where $\|f_m - f_n\|_{\mathcal{V}} < \epsilon$ for all $n, m \geq N$. Intuitively, a function with a small norm is relatively "flat", while a function with a large norm is quite oscillatory.

Definition 2.2.2 (Hilbert Space)**.** *A Hilbert space $\mathbb{H}$ is an inner product space where every Cauchy sequence $\{f_n\}_{n=1}^{\infty}$ converges to some element $f^* \in \mathbb{H}$.*

Example 2.1 (Square Integrable Functions)**.** The space of square integrable functions $L_2[0,1]$ is a prototypical example of a Hilbert space that is not $\mathbb{R}^d$. This space consists of all functions $f : [0,1] \to \mathbb{R}$ such that

$$\int_0^1 f^2(x)dx < \infty.$$

The inner product associated with $L_2[0,1]$ is given by

$$\int_0^1 f(x)g(x)dx, \quad \text{for all } f, g \in L_2[0,1].$$

In mathematics and computing, the term kernel is quite overloaded. Here, we use the term to refer generally to a function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ of two variables to the real line. Here $\mathcal{X}$ is some data space that we do not assume to have any structure. Images, word sequences, and probability distributions are all examples of valid data spaces. We will also use the term kernel to refer to a positive semidefinite kernel (see Definition 2.2.4) when the context is clear.

If we wish to view the kernel function as an inner product in some (possibly infinite) dimensional space, it is then worth asking which functions mapping two variables to the real line have this interpretation. The answer will lie in the class of positive semidefinite kernels.

Definition 2.2.3 (Positive Semidefinite Matrix)**.** *A matrix $M \in \mathbb{R}^{n \times n}$ is called positive semidefinite (PSD) if for all $\alpha \in \mathbb{R}^n$ we have*

$$\alpha^T M \alpha \geq 0.$$

Definition 2.2.4 (Positive Semidefinite Kernel Function)**.** *A symmetric, bivariate function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive semidefinite (PSD) if for all $n \in \mathbb{N}$ and all subsets $\{x_i\}_{i=1}^n \subset \mathcal{X}$ the matrix given by $\boldsymbol{K}_{ij} := \mathcal{K}(x_i, x_j)$ is PSD. The matrix $\boldsymbol{K}$ is known as the Gram matrix.*

Example 2.2 (Linear Kernel)**.** The simplest example of a PSD kernel function on $\mathbb{R}^d$ is that of the linear kernel. The linear kernel function $\mathcal{K}(x, z) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is simply given by

$$\mathcal{K}(x, z) = \langle x, z \rangle.$$

We can verify positive semidefiniteness by letting $\{x_i\}_{n=1}^n$ be a arbitrary collection of points in $\mathbb{R}^d$ and considering the matrix $\boldsymbol{K}_{ij} := \langle x_i, x_j \rangle$. Letting α be any vector in $\mathbb{R}^n$, we see

$$\alpha^T \boldsymbol{K} \alpha = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle = \| \sum_{i=1}^n \alpha_i x_i \|_2^2 \geq 0,$$

and conclude that $\mathcal{K}$ is PSD.

Example 2.3 (Feature Kernel)**.** Given any function $\varphi : \mathcal{X} \to \mathcal{V}_{feat}$ where $\mathcal{V}_{feat}$ has an inner product we may define a kernel $\mathcal{K}(x, z) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ by

$$\mathcal{K}(x, z) = \langle \varphi(x), \varphi(z) \rangle_{\mathcal{V}_{feat}}. \tag{2.1}$$

It is worth noting that the kernel is not uniquely associated with the feature function φ. For instance, if $U \in \mathbb{R}^{d \times d}$ is an orthogonal matrix and $\mathcal{V}_{feat} = \mathbb{R}^d$ then $\mathcal{K}(x, z) = \langle \varphi(x), \varphi(z) \rangle = \langle U\varphi(x), U\varphi(z) \rangle$. A kernel of this type is known as a feature kernel.

The following property of PSD kernels is generally useful and worth noting.

Proposition 2.2.5 (Cauchy-Schwarz Property of Kernels)**.** *With PSD kernel $\mathcal{K}$ and two arbitrary points x_1, x_2, we have*

$$\mathcal{K}^2(x_1, x_2) \leq \mathcal{K}(x_1, x_1)\mathcal{K}(x_2, x_2). \tag{2.2}$$

Proof. The proof follows from considering the determinant of the 2×2 Gram matrix. □

Having introduced the notion of Hilbert spaces and kernels, we are quite close to justifying the name of reproducing kernel Hilbert spaces. Given any PSD function $\mathcal{K}$ on $\mathcal{X} \times \mathcal{X}$, we will be able to define a Hilbert space of functions $\mathbb{H}$ such that

$$\langle \mathcal{K}(\cdot, x), f \rangle_{\mathbb{H}} = f(x), \quad \text{for all} f \in \mathbb{H}, x \in \mathcal{X}. \tag{2.3}$$

This identity (2.3) is known as the reproducing property.

Theorem 2.2.6. *Given any PSD kernel function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there is a unique Hilbert space such that the reproducing property 2.3 is satisfied. This Hilbert space is referred to as the reproducing kernel Hilbert space with kernel $\mathcal{K}$.*

It is worth taking the time to prove Theorem 2.2.6 as the proof is constructive and useful in Chapter 4. As we will see later on, it is beneficial to be able to characterize functions in an RKHS in terms of the associated unique kernel. We now show how to build an RKHS from a PSD kernel. To construct our RKHS, we will first need to construct a Hilbert space. This process consists of first building an inner product space and then completing it. We will then show that our Hilbert space has the reproducing property.

Proof. Consider the collection of functions $\tilde{\mathbb{H}}$ with the form

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i \mathcal{K}(\cdot, x_i),$$

for some collection of points $\{x_i\}_{i=1}^n \subset \mathcal{X}$ with $n \in \mathbb{N}$ and some weight vector $\alpha \in \mathbb{R}^n$. It is clear that $\tilde{\mathbb{H}}$ forms a vector space. Given any pair of functions $f, \bar{f} \in \tilde{\mathbb{H}}$ where $f(\cdot) = \sum_{i=1}^{n} \alpha_i \mathcal{K}(\cdot, x_i)$ and $\bar{f}(\cdot) = \sum_{i=1}^{\bar{n}} \beta_i \mathcal{K}(\cdot, \bar{x}_i)$ we define the inner product as

$$\langle f, \bar{f} \rangle_{\tilde{\mathbb{H}}} := \sum_{i=1}^{n} \sum_{j=1}^{\bar{n}} \alpha_i \beta_j \mathcal{K}(x_i, \bar{x}_j). \tag{2.4}$$

We wish our inner product to be independent of the representation of our functions, as there may be multiple equivalent representations. Note that

$$\left\langle f, \bar{f} \right\rangle_{\tilde{\mathbb{H}}} = \sum_{j=1}^{\bar{n}} \beta_j f(\bar{x}_j) = \sum_{i=1}^{n} \alpha_i \bar{f}(x_i),$$

so we need not be concerned with the particular representation of either function in regards to our inner product.

We'll now verify that (2.4) is a valid inner product according to Definition 2.2.1. Clearly, it satisfies symmetry and linearity. By positive definiteness of $\mathcal{K}$, $\langle f, f \rangle_{\tilde{\mathbb{H}}} \geq 0$. It remains to show that $\langle f, f \rangle_{\tilde{\mathbb{H}}} = 0$ if and only if $f = 0$. By the reproducing property as well as Proposition 2.2.5, we have

$$|f(x)|^2 = |\langle \mathcal{K}(\cdot, x), f \rangle_{\tilde{\mathbb{H}}}|^2 \leq \mathcal{K}(x, x) \langle f, f \rangle_{\tilde{\mathbb{H}}},$$

and $\langle f, f \rangle_{\tilde{\mathbb{H}}} = 0$ implies $f = 0$.

It is then possible to complete the space $\tilde{\mathbb{H}}$, which we will then refer to as $\mathbb{H}$. Finally, we show uniqueness of the space. Let $\mathbb{G}$ be another RKHS with kernel $\mathcal{K}$ so that $\mathcal{K}(\cdot, x) \in \mathbb{G}$ for all x. This implies the $\mathbb{H} \subseteq \mathbb{G}$. We can write $\mathbb{G} = \mathbb{H} + \mathbb{H}^{\perp}$. Let $g \in \mathbb{H}^{\perp}$ be an arbitrary function. Then, since $\mathcal{K}(\cdot, x) \in \mathbb{H}$ for all x we have $g(x) = \langle \mathcal{K}(\cdot, x), g \rangle_{\mathbb{G}} = 0$. Thus $\mathbb{H}^{\perp} = \{0\}$ and $\mathbb{H} = \mathbb{G}$. □

The proof of Theorem 2.2.6 suggests two important properties of RKHS. Namely, their reproducing property and the kernel function's role in providing a basis for the RKHS. These properties are fundamental enough that we may define an RKHS in terms of these properties.

Definition 2.2.7 (Reproducing Kernel Hilbert Space 1)**.** *Let $\mathcal{X}$ be a non-empty set and $\mathbb{H}$ a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. Then $\mathbb{H}$ is called a reproducing kernel Hilbert space with inner product $\langle \cdot, \cdot \rangle_{\mathbb{H}}$ and norm $\|\cdot\|_{\mathbb{H}} = \sqrt{\langle \cdot, \cdot \rangle_{\mathbb{H}}}$ if there exists a PSD kernel function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that*

1. $\langle \mathcal{K}(x, \cdot), f \rangle_{\mathbb{H}} = f(x), \quad$ *for all* $x \in \mathcal{X}$

2. $\mathbb{H} = \overline{\text{span}\{\mathcal{K}(\cdot, x) : \forall x \in \mathcal{X}\}}$, *for* $\overline{A}$ *denoting the completion of a set* A

Definition 2.2.7 helps characterize which functions f might belong to some RKHS $\mathbb{H}$. Roughly, any function $f \in \mathbb{H}$ must be at least as smooth as the kernel (where smoothness is measured by the RKHS norm), given that it is either a linear combination of partially evaluated kernels or the limit of such a combination. The reproducing property shows that any kernel may be written as a feature kernel (2.1) where $\varphi(x) := \mathcal{K}(x, \cdot)$ and $\mathcal{V}_{feat} := \mathbb{H}$. We see

$$\mathcal{K}(x, z) = \langle \mathcal{K}(x, \cdot), \mathcal{K}(z, \cdot) \rangle_{\mathbb{H}}.$$

An alternative definition of an RKHS — and one that is particularly relevant to learning (see Section 2.4) — is as a Hilbert space of functions where all evaluation functionals $\text{eval}_x[f] = f(x)$ are bounded (and therefore continuous). In that case, we may appeal to the Riesz Representation Theorem for a characterization of the kernel.

Theorem 2.2.8 (Riesz Representation Theorem)**.** *Let* L *be a bounded, linear functional on a Hilbert space* $\mathbb{H}$*. Then there exists a unique* $g \in \mathbb{H}$ *such that* $L[f] = \langle f, g \rangle_{\mathbb{H}}$ *for all* $f \in \mathbb{H}$*. We call* g *the representer of* L*.*

Definition 2.2.9 (Reproducing Kernel Hilbert Space 2)**.** *A reproducing kernel Hilbert space* $\mathbb{H}$ *is a Hilbert space of functions* $f : \mathcal{X} \to \mathbb{R}$ *such that for each* $x \in \mathcal{X}$ *the evaluation functional* eval_x *is bounded. That is*

$$\text{eval}_x[f] \leq M \|f\|_{\mathbb{H}} \quad M \in \mathbb{R}.$$

It can be shown that Definitions 2.2.7 and 2.2.9 are equivalent (see [89, p. 391]).

2.3 Mercer's Theorem

Clearly, a lot is going on inside of an RKHS. These spaces have nicely behaved functions and are characterized nicely by positive semidefinite kernel functions. The one-to-one correspondence between PSD kernels and RKHS ensures that we can work within an RKHS whenever we have an appropriate kernel. It turns out that *any* PSD kernel may be written as an inner product as in Example 2.3. In fact, infinitely many features can form a given kernel. Yet, there is one particularly important feature decomposition that yields a natural interpretation of RKHS as providing an embedding of our data domain $\mathcal{X}$ into ℓ^2.

Let the space $\mathcal{X}$ be measurable with respect to the non-negative measure μ. Then consider the function space $L^2(\mathcal{X}, \mu)$, simply L^2 for short. Given a symmetric PSD kernel, we can then define an integral operator $T_{\mathcal{K}}$ on L^2 as

$$T_{\mathcal{K}}[f](x) = \int_{\mathcal{X}} \mathcal{K}(x, z) f(z) d\mu(z). \tag{2.5}$$

If the kernel satisfies the following condition

$$\int_{\mathcal{X}} \int_{\mathcal{X}} \mathcal{K}^2(x, z) d\mu(x) d\mu(z) < \infty, \tag{2.6}$$

known as the Hilbert-Schmidt condition, then the operator (2.5) is bounded on L^2. See that

$$\begin{aligned} \|T_{\mathcal{K}}[f]\|_{L^2}^2 &= \int_{\mathcal{X}} \left(\int_{\mathcal{X}} \mathcal{K}(x, z) d\mu(x) \right)^2 d\mu(z) \\ &\leq \|f\|_{L^2} \int_{\mathcal{X}} \int_{\mathcal{X}} \mathcal{K}^2(x, z) d\mu(x) d\mu(z). \end{aligned}$$

These such operators are known as Hilbert-Schmidt operators, which may be seen to generalize a matrix. Indeed, if $\mathcal{X} = \{x_i\}_{i=1}^m$ and $\mu(x_i) = 1$ for all $i = 1, \ldots, m$ then

$$T_{\mathcal{K}}[f](x) = \sum_{i=1}^{m} \mathcal{K}(x, x_i) f(x_i),$$

and the operator reduces to matrix multiplication. We may use eigenfunctions and eigen-

values of the integral operator in order to decompose the kernel.

Theorem 2.3.1 (Mercer's Theorem). *If $\mathcal{X}$ is compact, $\mathcal{K}$ is PSD, continuous, and satisfies the Hilbert-Schmidt condition then there exists a sequence of eigenfunctions $\{\varphi_i\}_{i\in\mathbb{N}}$ that form an orthonormal basis for $L^2(\mathcal{X},\mu)$ and non-negative eigenvalues $\{\lambda_i\}_{i\in\mathbb{N}}$ such that*

$$T_{\mathcal{K}}[\varphi_i] = \lambda_i \varphi_i, \tag{2.7}$$

and

$$\mathcal{K}(x,z) = \sum_{i=1}^{\infty} \lambda_i \varphi_i(x)\varphi_i(z), \tag{2.8}$$

where the infinite series converges uniformly and absolutely.

We pause briefly to ask: what would one call a young eigensheep? Clearly, it is a lamb, duh [75]. Continuing, we see that any kernel $\mathcal{K}$ may be seen to be a feature kernel (2.1) by the mapping of $\mathcal{X}$ into ℓ^2 defined by $\varphi : \mathcal{X} \to \ell^2$

$$\varphi(x) := \left[\sqrt{\lambda_1}\varphi_1(x), \sqrt{\lambda_2}\varphi_2(x), \sqrt{\lambda_3}\varphi_3(x), \dots\right]^T,$$

where $\{(\lambda_i,\varphi_i)\}_{i=1}^{\infty}$ are the eigenvalues and eigenfunctions guaranteed by Mercer's theorem.

In Section 4.2, we show how to estimate the eigenvalues and eigenfunctions associated with the integral operator (2.5) from data. The ability to approximate these quantities through data allows one to gain insight into the properties of the RKHS associated with a kernel when solving the integral operator eigenvalue problem (2.7) is intractable, as is the case with the feature kernel (2.1) defined by the features learned by a neural network.

2.3.1 *Approximation of Kernels by Neural Networks*

Note that Theorem 2.3.1 implies that given any appropriate kernel and some $\epsilon > 0$ there exists some $d \in \mathbb{N}$ such that $|\langle \varphi(x), \varphi(z)\rangle_{\mathbb{R}^d} - \mathcal{K}(x,z)| < \epsilon$. Thus any kernel may be learned

by a neural network of width d up to any desired ϵ. However, we focus on studying the kernel associated with learned features rather than learning features associated with a known kernel.

2.4 Why Use an RKHS?

What makes an RKHS an appropriate and useful space in a machine learning context? One way to define an RKHS is as a Hilbert space of functions such that the evaluation functional is bounded (Definition 2.2.9). This important characterization implies that convergence in an RKHS implies pointwise convergence. Indeed this is easy to see using the Cauchy-Schwarz inequality. Let $f_j \to f^*$ be a sequence of functions converging in the RKHS $\mathbb{H}$. Then

$$\begin{aligned} |f_j(x) - f^*(x)| &= |\langle R_x, f_j\rangle_{\mathbb{H}} - \langle R_x, f^*\rangle_{\mathbb{H}}| \\ &= |\langle R_x, f_j - f^*\rangle_{\mathbb{H}}| \\ &\leq \|R_x\|_{\mathbb{H}}\|f_j - f^*\|_{\mathbb{H}} \to 0, \end{aligned}$$

where R_x is the representer of evaluation guaranteed by Theorem 2.2.8.

In the context of statistical learning, this is an essential property. We will be seeking a sequence of iterates in our hypothesis space that converge to some ideal function f^*. However, we are typically limited to finding a function $\hat{f}$ which agrees with our ideal function on a finite set of training examples $S = \{(x_i, f^*(x_i))\}_{i=1}^n$. Were our hypothesis space not an RKHS, our iterates may converge in hypothesis space while disagreeing over the training set rendering us incapable of learning f^* using the norm associated with the hypothesis space. This situation is illustrated in the following example.

Example 2.4. Let our hypothesis space be $L^2[0,1]$ — which is not an RKHS — equipped with the usual norm. Assume that $f^*(x) = 0$ for all x and consider the iterates $f_n(x) =$

$x^n, n \in \mathbb{N}$. Then, $\|f_n\|^2_{L^2[0,1]} = \int_0^1 x^{2n} dx = \frac{1}{2n+1}$, so we see that $f_n \to f^*$ in $L^2[0,1]$. Yet $f_n(1) = 1$ for all $n \in \mathbb{N}$, so $\{f_n\}_{n\in\mathbb{N}}$ does not converge pointwise to f^*.

2.5 LEARNING IN AN RKHS

We now discuss learning in an RKHS. We give some bounds on the Rademacher complexity of RKHS and discuss the use of RKHS to regularize the problem. These topics allow for theoretical guarantees on the bound of the generalization gap (1.6), as these bounds give an explicit description of the Rademacher complexity of a bounded subset of an RKHS. We then discuss how to evaluate the RKHS norm of a function which leads us to the representer theorem. Finally, we discuss more modern work drawing analogies between learning with neural networks and learning in an RKHS.

Theorem 2.5.1 (Rademacher Complexity Based on Kernels). *Let $\mathcal{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PSD kernel associated with RKHS $\mathbb{H}$. Let $S \subseteq \{x : \mathcal{K}(x,x) \leq r^2\}$ be a sample of size n and $\mathbb{H}_\Lambda = \{\langle w, \mathcal{K}(x,\cdot)\rangle_{\mathbb{H}} : \|w\|_{\mathbb{H}} \leq \Lambda\}$ for some $\Lambda \geq 0$. Then*

$$\mathfrak{R}_n(\mathcal{H}) \leq \frac{\Lambda\sqrt{\mathrm{Tr}(\boldsymbol{K})}}{n} \leq \sqrt{\frac{r^2\Lambda^2}{n}}, \tag{2.9}$$

where $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ is the Gram matrix associated with S.

Proof. The proof is given by [60, p.118]. □

Theorem 2.5.1 in combination with Theorem 1.1.6 allows us to bound the generalization gap associated to a function $f \in \mathbb{H}_\Lambda$. It is then of interest to control the RKHS norm of a learned function. This capacity control procedure is known as regularization.

2.5.1 *Regularization*

RKHS theory provides a natural framework to regularize the learning problem. The intuition complementing the theoretical benefits may be thought of in analogy to Occam's

razor — the simplest sufficient solution is likely the best. One way regularization may be achieved is through the regularized empirical loss minimization problem.

$$\min_{f\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} L(f(x_i), y_i) + \lambda R(f), \tag{2.10}$$

where the loss functional L over the hypothesis space $\mathcal{H}$ measures goodness of fit to the data and R is a regularizer used to penalize overfitting and incorporate prior knowledge—such as smoothness of the minimizer. The tuning parameter λ controls the trade-off between minimizing the sometimes conflicting objectives. Typically, R is taken to be convex since when L is also convex, this ensures that there exists a global minimum and any minimum found is global rather than local. We may take $\mathcal{H}$ to be an RKHS $\mathbb{H}$ and R to be the associated norm squared, giving

$$\min_{f\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} L(f(x_i), y_i) + \lambda\|f\|_{\mathbb{H}}^2. \tag{2.11}$$

Minimizing $\|f\|_{\mathbb{H}}^2$ serves to simplify f while still ensuring good agreement with the data, and thus small values of the loss. In general, the objective (2.11) is intractable due to the need to calculate the norm in $\mathbb{H}$ where such spaces are often large or infinite dimensional. However, we will see that this difficulty may be avoided with the representer theorem 2.5.2.

2.5.2 *Evaluating the RKHS norm*

A practical problem that naturally arises is computing the norm of a function f in the RKHS $\mathbb{H}$. Recall that by Definition 2.2.7 for any $f \in \mathbb{H}$ there is some $\alpha \in \mathbb{R}^m$ and collection $\{x_i\}_{i=1}^m \subset \mathcal{X}$ such that

$$f(x) = \sum_{i=1}^{m} \alpha_i \mathcal{K}(x, x_i). \tag{2.12}$$

Then

$$\begin{aligned}\|f\|_{\mathbb{H}}^2 &= \langle f, f\rangle_{\mathbb{H}} \\ &= \left\langle \sum_{i=1}^m \alpha_i \mathcal{K}(x, x_i), \sum_{j=1}^m \alpha_j \mathcal{K}(x, x_j) \right\rangle_{\mathbb{H}} \\ &= \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j \mathcal{K}(x_i, x_j) \\ &= \alpha^T \boldsymbol{K} \alpha,\end{aligned} \tag{2.13}$$

where $\alpha = [\alpha_1, \ldots, \alpha_m] \in \mathbb{R}^m$ and $\boldsymbol{K}$ is the Gram matrix associated with $\{x_i\}_{i=1}^m$. The issue in using (2.13) is that generally the collection of points $\{x_i\}_{i=1}^m$ is not known requiring one to integrate over the entire input space $\mathcal{X}$. This problem is addressed through the famous representer theorem originally due to Kimeldorf and Wahba [49] and [50].

Link this with representer form

Theorem 2.5.2 (Representer Theorem)**.** *Let R be a strict, monotonic increasing function and $\mathbb{H}$ be an RKHS of functions $f : \mathcal{X} \to \mathcal{Y}$ associated with the kernel $\mathcal{K}$. Then for the dataset $S = \{(x_i, y_i)\}_{i=1}^n$ each minimizer of the regularized risk with the loss functional L*

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n L(f(x_i), y_i) + \lambda R(\|f\|_{\mathbb{H}}) \tag{2.14}$$

has a representation in terms of the data points such that

$$f(x) = \sum_{i=1}^n \alpha_i \mathcal{K}(x, x_i). \tag{2.15}$$

We will say functions in an RKHS of the form (2.15) are in representer form.

Proof. Assume that we are working with $\hat{R}(\|f\|_{\mathbb{H}}) = R(\|f\|_{\mathbb{H}}^2)$. Note that $\hat{R}$ is strictly monotonic on $[0, \infty)$ if and only if R is. Now, we decompose any $f \in \mathbb{H}$ into the part spanned by $\{\mathcal{K}(\cdot, x_i)\}_{i=1}^n$ and this set's orthogonal complement leading to

$$f(x) = \sum_{i=1}^n \alpha_i \mathcal{K}(x, x_i) + f_\perp(x).$$

Here $\alpha_i \in \mathbb{R}$ and $f_\perp \in \mathbb{H}$ is such that $\langle f_\perp \mathcal{K}(x_i, \cdot)\rangle_{\mathbb{H}} = 0$ for all $i = 1, \ldots, n$. It then remains to show that minimizers of (2.14) have $f_\perp(x) = 0$. First see that for $\{x_j\}_{j=1}^n$

$$\begin{aligned} f(x_j) &= \langle f(\cdot), \mathcal{K}(\cdot, x_j)\rangle_{\mathbb{H}} \\ &= \langle \sum_{i=1}^n \alpha_i \mathcal{K}(\cdot, x_i), \mathcal{K}(\cdot, x_j)\rangle_{\mathbb{H}} + \langle f_\perp(\cdot), \mathcal{K}(x_j, \cdot)\rangle_{\mathbb{H}} \\ &= \sum_{i=1}^n \alpha_i \mathcal{K}(x_j, x_i). \end{aligned}$$

So, the value of the loss function does not depend on $f_\perp$. Also, note that

$$\hat{R} = R(\|s\|_{\mathbb{H}}^2 + \|f_\perp\|_{\mathbb{H}}^2) \geq R(\|s\|_{\mathbb{H}}^2).$$

Thus, when α_i are fixed (2.14) is minimized when $f_\perp = 0$.

□

The representer theorem guarantees that solutions to RKHS regularized problems have representations in terms of points in our dataset. This gives the ability to compute norms of minimizing functions. The ability to represent optimal solutions to (2.14) as linear combinations of the kernel function partially evaluated on points within the training set reduces the minimization problem from a search over an infinite dimension space to a search for coefficients $\alpha = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^n$.

Additionally, the representer theorem allows us to interpret the behaviour of a function f in representer form. Because the kernel function is an inner product in an RKHS, we see that $f(x)$ is a weighted combination of inner products in this RKHS. That is, $f(x)$ produces a weighted score of how similar x is to points in the training set in terms of the features associated with $\mathcal{K}$. In Section 4.1, we take inspiration from the representer theorem to show how to evaluate the norm of a neural network in terms of the RKHS associated with the feature kernel (2.1).

2.6 KERNEL MACHINES

Functions in an RKHS applied in the context of machine learning are often referred to as kernel machines.

Definition 2.6.1 (Kernel Machine)**.** *A kernel machine is a function f in the RKHS $\mathbb{H}$ associated with a kernel $\mathcal{K}$. Given a dataset $S = \{x_i\}_{i=1}^n \subset \mathcal{X}$ a kernel machine is a function in representer form*

$$f(x) = \sum_{i=1}^{n} \alpha_i \mathcal{K}(x_i, x). \tag{2.16}$$

If we view $\mathcal{K}(x, z) = \langle \varphi(x), \varphi(z) \rangle_{\mathcal{V}_{feat}}$ for some feature function $\varphi : \mathcal{X} \to \mathcal{V}_{feat}$ we may view a kernel machine as a two layer neural network

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha_i \langle \varphi(x_i), \varphi(x) \rangle_{\mathcal{V}_{feat}} \\ &= \left\langle \sum_{i=1}^{n} \alpha_i \varphi(x_i), \varphi(x) \right\rangle_{\mathcal{V}_{feat}} \\ &= \langle w, \varphi(x) \rangle_{\mathcal{V}_{feat}}, \end{aligned} \tag{2.17}$$

where the first layer is fixed to be $\varphi(x)$ and only w is learnable. Care must be taken to learn $w \in \mathcal{V}_{feat}$. Generally, we may take $\mathcal{V}_{feat} \subseteq \ell^2$.

One may also view neural networks as kernel machines. We espouse this view in Chapter 4, where we argue the benefits of viewing neural networks as kernel machines with respect to the feature kernel (2.1). An alternate view shows that neural networks trained with gradient descent using an infinitesimal step size are also kernel machines [28]. The particular kernel is given by the neural tangent kernel (see Section 3.1) integrated along the path of the parameters during optimization. However, the exact kernel given is practically tough to evaluate. Additionally, the coefficients of the partially evaluated kernels depend on the input to the kernels, which is atypical.

CHAPTER 3

SHORT SURVEY OF KERNELS AND NEURAL NETWORKS

3.1 INFINITE WIDTH NEURAL NETWORKS

The first connection to be made between neural networks and kernels was made by Radford Neal in his 1994 thesis [63]. Neal considers neural networks f_θ of the form (1.10) with $L = 2$ and $\sigma(x) = \tanh(x)$. For cleaner notation, we will drop the θ subscript denoting that f_θ is a parameterized function.

$$\begin{aligned} f(x) &= \frac{W}{\sqrt{H}} \tanh(Ux + a) + b \\ &:= \frac{W}{\sqrt{H}} h(x) + b, \quad h(x) := \tanh(Ux + a) \end{aligned} \tag{3.1}$$

Where $f : \mathbb{R}^I \to \mathbb{R}^O$ and the hidden layer has H units. Thus, $U \in \mathbb{R}^{I \times H}$ and $W \in \mathbb{R}^{O \times H}$. The weights U_{ij} and biases a_i are initialized according to mean zero Gaussians with standard deviations σ_u and σ_a. Additionally, the weights W_{ki} and biases b_k are initialized according to mean zero Gaussians with standard deviations σ_w and σ_b.

Considering the k-th output of the network initialized as above, we see that

$$f_k(x) = \frac{1}{\sqrt{H}} \sum_{i=1}^{H} W_{ki} h(x)_i + b_k \tag{3.2}$$

The expectation of each term in the sum is $\mathbb{E}[W_{ki} h_i(x)] = \mathbb{E}[W_{ki}]\mathbb{E}[h_i(x)] = 0$ due to independence of the initializations. The variance of the terms is $\mathbb{E}[(W_{ki} h_i(x))^2] = \mathbb{E}[W_{ki}^2]\mathbb{E}[h_i(x)^2] = \sigma_w^2 \mathbb{E}[h_i(x)^2] < \infty$ which is finite as $h(x)$ is bounded by 1. Define $V := \mathbb{E}[h_i(x)^2]$ which is equal for all i. Accordingly, we may apply a central limit theorem to see that as $H \to \infty$ $f_k(x)$ become Gaussian with zero mean and variance $\sigma_b^2 + \sigma_w^2 V$.

Consider now the joint distribution of the outputs f_k for a number of inputs $\{x^{(i)}\}_{i=1}^n$. An identical argument to the above shows that this joint converges to mean zero, multivariate Gaussian with covariance

$$\begin{aligned}\mathbb{E}[f_k(x^{(p)})f_k(x^{(q)})] &= \sigma_b^2 + \frac{1}{H}\sum_{i=1}^{H}\mathbb{E}[h_i(x^{(p)})h_i(x^{(q)})] \\ &= \sigma_b^2 + \sigma_w^2\mathbb{E}[h_i(x^{(p)})h_i(x^{(q)})]\end{aligned} \tag{3.3}$$

It is interesting to note that any two distinct outputs of the neural network described above are independent despite using the same hidden layer features. This is a consequence of the fact that for Gaussian random variables, having zero covariance implies independence.

Distributions over functions where the distribution of the function values at any finite collection of points is multivariate Gaussian are known as Gaussian processes. Gaussian processes and kernels are intimately linked through the covariance function of the process. Observations are thus connected through their covariance, defined by the network. This covariance function is a kernel connecting the gaussian process — and thus scaled, infinitely wide, 2-layer neural networks — to the theory of RKHS. Covariance is symmetric and positive definite and thus defines a valid kernel; this is shown formally through the Loève representation theorem [13, p. 65]. Neal then uses this connection to Gaussian processes to perform Bayesian inference over neural network priors. While Neal's results apply only to neural networks with one hidden layer, identical results were later obtained by Lee et al. [56] for networks of many layers. A follow-up by Williams gives some methods for computing with infinite width neural networks [90]. Inspired by this work, Cho and Saul [19] propose a class of kernels meant to mimic neural networks.

Recently, one of the developments that has most excited researchers is the discovery of the Neural Tangent Kernel (NTK) [47]. As the name might suggest, the neural tangent kernel of a neural network $f_\theta : \mathcal{X} \to \mathbb{R}$ with layers scaled as in (3.1) is given by $\langle \nabla_\theta f_\theta, \nabla_\theta f_\theta \rangle$.

By using the above equivalence of infinitely wide neural networks to Gaussian processes, the authors are able to show that the NTK converges in probability to a deterministic limit as the neural network approaches infinite width. The distribution of the NTK for particular networks at initialization is studied by [45]. Additionally, the authors show that performing gradient descent with an infinitely small step size on a parameterized function with respect to a loss function is equivalent to performing gradient descent in function space with respect to the tangent kernel. Using this deterministic NTK, the authors can study the training dynamics of a neural network under gradient descent with respect to the square loss. For a dataset $\mathcal{D}$ with $|\mathcal{D}| = m$ the square loss is given by

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y_i - f_\theta(x_i))^2$$

The authors use this connection to help explain the behaviour of the network during training, namely, that the parameters will evolve along a line in the direction of the first kernel principal component with respect to the NTK (for more information on kernel principal component analysis see [80, p. 429]). A remarkable consequence of this connection between kernel and cost gradient descent is that the optimization is guaranteed to converge to a global minimum in this infinite width limit. This is in contrast to general gradient descent results, which only guarantee convergence to a local minimum. A practical consequence is that for finite-width neural networks, by Taylor expanding f_θ around θ, one may view the linearized neural network at θ trained with the square loss as performing kernel ridge regression with the NTK kernel [91, 47]. This suggests that the linearization of the neural network can also be studied and approximated from the perspective of kernel methods.

A common, critical assumption in all of these works is that the last layer of the neural network is linear, i.e., we do not apply a non-linear activation on top of the final linear transform. This assumption is necessary to apply a central limit theorem and thus obtain convergence of the network's outputs to Gaussian variables. Additionally, convergence

in the context of central limit theorems is known to happen quite slowly ($\mathcal{O}(\sqrt{n})$ where n is the layer width), so it is not clear that results relying on infinite width limits are likely to approximate the behaviour of practical neural networks sufficiently; experiments performed in these papers use networks of width $\approx 10,000$ for best results which is exceedingly rare in practice. As Neal himself says, "[in the infinite width limit] the contributions of individual hidden units are all negligible, and consequently, these units do not represent hidden features that capture important aspects of the data" [63, p.49].

3.2 FINITE WIDTH NEURAL NETWORKS

In many cases, it seems more natural to view a neural network from the feature learning perspective as in Equation 1.11, where a neural network learns meaningful features associated with the data. Recall also that we may view a kernel machine as a 2-layer neural network (2.17) where the feature function is fixed and induced by the kernel. Authors taking this perspective then study kernels related to a neural network feature function φ_θ.

For instance, Zhang et al. [95] introduce the neural Fisher kernel to extract relevant features from a neural network. The Fisher kernel measures the similarity of two data points in terms of the Fisher score and the Fisher information matrix [46]. The authors then use an approximation to the neural Fisher kernel to get compact and informative representations of the data. The neural Fisher kernel incorporates gradient information with respect to the model's parameters in a similar fashion to the NTK. This gradient information can allow for more informative features to be extracted from the model. Indeed, the authors are motivated to extract features from generative models, which are known to produce relatively poor features in general [96].

However, when good quality features are available, it is simpler to study the feature kernel associated with the neural network (see Example 2.3). Additionally, neural networks do not include gradient information in their output. Thus, it is more practical to study the neural feature kernel.

3.2.1 *Neural Feature Kernel*

Let $f_\theta(x) = W_L\varphi_\theta(x) + b_L$, where f_θ is a neural network. Then, the neural feature kernel associated with f_θ is

$$\mathcal{K}(x, z) = \langle \varphi_\theta(x), \varphi_\theta(z) \rangle . \tag{3.4}$$

It must be noted that the feature kernel is also known as the conjugate kernel in some literature. However, given our view of kernels and neural networks as models which work with features at their core, the term feature kernel is preferred. This term has been used previously when focusing on the features that neural networks produce [40].

The feature kernel is frequently studied — often incidentally — in theoretical analyses of contrastive learning. This should not be surprising as contrastive learning essentially seeks to enforce similarity (or dissimilarity) relations between representations of data, and kernels quantify exactly these types of relations. For example, the authors of [38] show that minimizing a particular variant of the InfoNCE loss [68] using data augmentation to create positive and negative samples yields a representation which approximately decomposes a particular kernel which measures how likely one input is to be an augmentation of the other. The authors use this analysis to show that these learned features can separate sufficiently well-clustered data. Similarly, [97] show that contrastive learning under the right conditions learns a feature function corresponding to a kernel which gives the dot product between the latent variables which have generated the observed inputs. This allows the authors to prove that the features learned are equal to the latent variables up to permutations. Similar results are seen in [53].

However, the connection between the feature kernel and neural networks can be made very explicitly. Daniely, Frostig, and Singer [25] argue for a dual view of neural networks and kernels to study the effects of random parameter initialization in neural networks. The main results show that the neural feature kernel is approximately equal to a compositionally defined kernel consisting of the repeated application of inner products

followed by dual activation functions. A dual activation function $\hat{\sigma}$ is defined as

$$\hat{\sigma} = \mathbb{E}_{(X,Y)\sim N_\rho}\left[\sigma(X)\sigma(Y)\right]$$

where σ is an activation function and N_ρ is a centred multivariate Gaussian with covariance $\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$. The design of this kernel is clearly meant to mimic neural networks. This compositional kernel is referred to as the dual kernel. Using this dual kernel, the authors show that the set of functions expressible using the dual kernel is approximately equal to the set of functions expressible with the neural network with randomly initialized weights, implying that with the right initialization, one need not train φ_θ and can simply train the last layer instead. This idea has shown empirical promise [79]. Additionally, this view provides an RKHS that may be used to study the functions learnable by the neural network near initialization. Note that the results suggest that weights do not change with training but rather that the functions expressible with the neural network do not change drastically. These findings also appear to agree with some NTK results suggesting that there are invariants associated with the weights given the appropriate initialization.

Inspired by this work, Fan and Wang [32] study the spectra of the feature kernel and the NTK at initialization in finite width neural networks where layer width increases linearly with sample size. They show that the feature kernel spectrum may be found by iterating the Marchenko-Pastur distribution across the hidden layers. Similarly, the NTK spectrum can be found using a recursive, fixed-point iteration relating to the Marchenko-Pastur distribution. The authors then study these kernels empirically over the course of training. They see that isolated principal components emerge from the eigenvalues bulk, which are predictive of the training labels. These results, on their face, appear to disagree with [25] as these principle components appear to be related to useful features that the network has learned. Indeed, to our knowledge, although randomly initialized and untrained features are surprisingly effective [34, 79], any network's predictions improve with training.

3.3 Overparameterized Learning

Zhang et al. [94] provided empirical evidence that neural networks trained to have zero classification risk in training do not have significant problems with overfitting. This is despite the same networks being capable of fitting randomly labelled data. This observation would appear to disagree with generalization bounds based on Rademacher complexity of the hypothesis class. Neural networks are able to fit randomly labelled data implying they have high Rademacher complexity suggesting that neural networks will generalize poorly; however, given the empirical success of neural networks, this does not appear to be an issue in practice. This extremely interesting observation has intrigued the machine learning community since its inception.

Recently, it was discovered that kernel machines and even linear models could exhibit this same behaviour when the number of parameters greatly exceeds the number of available data [9, 7]. Given that relatively well understood models can share this perplexing behaviour with neural networks, it was suggested that understanding this behaviour in kernel machines would give insight into the behaviour driving generalization in neural networks [9, 8]. This idea is lent further support with the view of kernel machines as neural networks with a fixed representation layer (see (2.17)). As the models have many times more parameters than available data, this line of research is known as the study of overparameterized models.

A proposed explanation for this behaviour suggests that in a hypothesis space rich enough to contain many functions which perfectly fit the training data, it will be possible to select one such function with favourable generalization properties [10, 8]. The selection of this ideal function must then depend on the inductive biases associated with the training method. For instance, it was shown that kernel machines in representer form have particularly good generalization abilities [9]. It is then shown that the inductive bias produced through optimization using SGD initialized at 0 is compatible with finding a minimum RKHS norm solution. Recall that such solutions are in representer form

(Theorem 2.5.2).

The notion of double descent expands further on this hypothesis. Take the number of parameters in a model as a measure of model complexity. Beginning from an empty model as parameters are added and the model retrained, double descent suggests that classical learning theory will hold with the model first underfitting, then finding a 'sweet spot', followed by the model overfitting up until the point where the model is capable of perfectly fitting the training data. This point is called the interpolation threshold and models which perfectly fit the training data are called interpolating models. Past the interpolation threshold, as more parameters are added to the model, the generalization gap again begins to narrow, suggesting that as the set of learnable interpolating models increases, inductive biases are more able to select functions which generalize [10]. The practitioner often produces such an inductive bias themselves. When training a neural network, it is common practice to periodically test model performance on a held-out test set and to retain the model with the best performance. This practice ensures that the final model will be the one with the best observed test performance rather than simply the model obtained at the end of training. Double descent is illustrated in Figure 3.1.

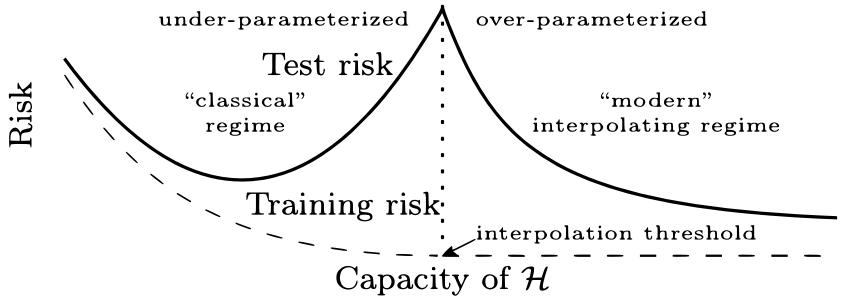


Figure 3.1: *(figure reproduced from [8], licensed under CC BY 4.0) An illustration of double descent. Starting from the left, the risk of the model decreases as a function of model capacity. As model capacity increases further, a threshold is reached where the model begins to overfit. This behaviour is encapsulated in classical learning theory. Overfitting worsens until the interpolation threshold is reached. Past the interpolation threshold, the model risk again decreases with model capacity. Double descent suggests this is due to the larger set of interpolating models, some of which generalize well and are selected for. This behaviour has been suggested by more modern research.*

Understanding overparameterization in kernel machines, with their abundance of

theory, promises to give insight into the generalization behaviour of neural networks. However, this area of research is young, and many questions remain open. Moreover, classical learning theory already provides generalization bounds for functions in an RKHS. In this thesis, we do not focus on the overparameterization regime; however, given that reproducing kernel theory is useful in both classical and overparameterized regimes, it is clear that it is important to better understand the connections between neural networks and kernel machines.

CHAPTER 4

STUDY OF THE NEURAL FEATURE KERNEL

Upon defining a satisfactory kernel $\mathcal{K}$ — as we have seen in previous sections — the practitioner has implicitly defined a (non-unique) feature map $\varphi : \mathcal{X} \to \mathcal{V}_{feat}$ such that $\langle \varphi(x), \varphi(z) \rangle_{\mathcal{V}_{feat}} = \mathcal{K}(x, z)$. Noticing this, we view the problem in reverse from a feature learning perspective where one seeks to *learn* a feature map φ_θ from a dataset $S = \{(x_i, y_i)\}_{i=1}^n$ using a neural network parameterized by θ. Upon successfully learning a feature map, we may study the associated feature kernel. Namely

$$\mathcal{K}(x, z) := \langle \varphi_\theta(x), \varphi_\theta(z) \rangle_{\mathcal{V}_{feat}}, \tag{4.1}$$

where $\mathcal{V}_{feat} := \overline{\text{span}\{\varphi_\theta(x) : x \in \mathcal{X}\}} \subseteq \mathbb{R}^d$ (see Example 2.3). We refer to this kernel as the neural feature kernel or simply the feature kernel when the context is clear. Since $\mathcal{V}_{feat} \subseteq \mathbb{R}^d$ the inner product in $\mathcal{V}_{feat}$ is taken to be the usual dot product on $\mathbb{R}^d$.

Having learned φ_θ, we compose the features with a linear function to allow the neural network to be deployed on a downstream task. The resulting function can be viewed in the form of Equation 1.11 giving

$$f_\theta(x) = w^T \varphi_\theta(x), \tag{4.2}$$

where $f_\theta : \mathcal{X} \to \mathbb{R}$. A bias term is not included as it may be incorporated into the weights w.

4.1 EVALUATING THE RKHS NORM OF A NEURAL NETWORK

Recall Equation 2.17, where it was shown that kernel machines may be viewed as neural networks. Similarly, for an appropriate choice of w, the neural network f_θ may be viewed as a kernel machine lying in the RKHS $\mathbb{H}$ associated with $\mathcal{K}$. Specifically, $f_\theta \in \mathbb{H}$ if and only if $w \in \mathcal{V}_{feat}$. Given $w \in \mathcal{V}_{feat}$, we could write $w = \sum_{j=1}^m a_j \varphi_\theta(x_j)$ for some collection of points $A = \{x_j\}_{j=1}^m \subset \mathcal{X}$ with $m \in \mathbb{N}$. With access to A, we could compute the RKHS norm of f_θ as shown in Section 2.5.2. However, we do not, in general, have knowledge of A. Instead, we seek to approximate f_θ in representer form (2.15) by projecting w into $\mathcal{V}_n := \text{span}\{\phi_\theta(x) | x \in S\} \subset \mathcal{V}_{feat}$. Thus reducing the projection problem to least squares regression.

Define F_θ to be the $d \times n$ matrix with columns given by $\{\varphi_\theta(x) : x \in S\}$. We seek

$$b^* = \arg\min_{b \in \mathbb{R}^n} \|w - F_\theta b\|_2^2. \tag{4.3}$$

In practice, numerical instabilities can make solving (4.3) difficult, so a small regularization term is added. This leads to the objective

$$b^* = \arg\min_{b \in \mathbb{R}^n} \|w - F_\theta b\|_2^2 + \lambda \|b\|_2^2, \tag{4.4}$$

which may be solved using an abundance of least squares algorithms. Once we have solved for b^*, we may define both

$$\hat{w} = \sum_{i=1}^n b_i^* \varphi_\theta(x_i), \tag{4.5a}$$

$$\hat{f}(x) = \hat{w}^T \varphi_\theta(x). \tag{4.5b}$$

From there, it is straightforward to define the Gram matrix $\boldsymbol{K}_{ij} = \mathcal{K}(x_i, x_j)$ for all pairs $(x_i, x_j) \in S \times S$ and evaluate the norm of the projected function $\hat{f}$

$$\|\hat{f}\|_{\mathbb{H}} = \sqrt{b^{*T}\boldsymbol{K}b^*}.$$

Proposition 4.1.1. *Given a function $f(x) = w^T\varphi_\theta(x)$ and $\hat{f}(x)$ as in Equation 4.5b the absolute difference between the norms $\|f\|_{\mathbb{H}}$ and $\|\hat{f}\|_{\mathbb{H}}$ is bounded by a constant depending on the feature kernel and the representation of the coefficients in $\mathcal{V}_{feat}$.*

Proof. If $w \notin \mathcal{V}_{feat}$ then $f \notin \mathbb{H}$ and $\|f\|_{\mathbb{H}}$ is taken to be infinite.

Now assume $w \in \mathcal{V}_{feat}$ and write $w = \sum_{j=1}^m a_j\varphi_\theta(x_i)$ for some collection of points $A = \{x_i\}_{i=1}^m \subset \mathcal{X}$. Let $C := A \cup S$ and $p = |C|$. Then, extend the coefficients $\{a_i\}_{i=1}^m, \{b_i^*\}_{i=1}^n$ by defining $\alpha, \beta \in \mathbb{R}^p$ such that for $k = 1, \dots, p$

$$\alpha_k = \begin{cases} a_k & x_k \in A \\ 0 & x_k \notin A \end{cases}$$

$$\beta_k = \begin{cases} b_k^* & x_k \in S \\ 0 & x_k \notin S \end{cases}$$

Then $f(x) = \sum_{k=1}^p \alpha_k\varphi_\theta(x_k)^T\phi(x)$ and $\hat{f}(x) = \sum_{k=1}^p \beta_k^*\varphi_\theta(x_k)^T\phi(x)$. Thus,

$$\begin{aligned} |\|f\|_{\mathbb{H}} - \|\hat{f}\|_{\mathbb{H}}| &\leq \|f - \hat{f}\|_{\mathbb{H}} \\ &= \|w - \hat{w}\|_2 \\ &= \sqrt{(\alpha - \beta)^T\boldsymbol{G}(\alpha - \beta)} \\ &\leq \|\alpha - \beta\|_2\sqrt{\|\boldsymbol{G}\|_{op}}, \end{aligned}$$

where $\boldsymbol{G}_{ij} := \mathcal{K}(x_i, x_j)$ for all pairs $(x_i, x_j) \in C \times C$.

□

In the following section, we show that $\boldsymbol{G}$ approximates the integral operator (2.5) defined in Section 2.3. As a consequence $\|\boldsymbol{G}\|_{op} \approx \|\boldsymbol{K}\|_{op}$. We also suggest how one could

estimate and control $\|\boldsymbol{G}\|_{op}$.

4.1.1 *Multi-Output Neural Networks*

In a k-class classification setting, one typically will regress to e_c, the standard basis vector of a data point with label c. For k-dimension regression, we regress to each dimension of the target. In these cases, the neural network takes the form

$$f_\theta(x) = W^T \varphi_\theta(x) = \begin{bmatrix} w_1^T \varphi_\theta(x) \\ w_2^T \varphi_\theta(x) \\ \vdots \\ w_k^T \varphi_\theta(x) \end{bmatrix},$$

in which case we may view our network as k stacked subnetworks and apply the above theory to each of the k subnetworks.

4.2 ESTIMATING MERCER'S DECOMPOSITION OF THE FEATURE KERNEL

We now describe a method of estimating the Mercer decomposition guaranteed by Mercer's Theorem 2.3.1 for the neural feature kernel $\mathcal{K}$ (4.1) with associated RKHS $\mathbb{H}$. The approach holds for kernels generally and was first introduced in [76] and later, apparently independently, by [61] for general operators on RKHS. The method is closely related to the Nyström method, which is often used to speed up computation involving the Gram matrix in traditional kernel methods [29]. The approach was used in the context of neural networks by [46]; however, they justify their approach as Monte Carlo estimation of (2.5) rather than through the theory of operators as is done here.

In general, solving the eigenvalue problem for the integral operator $T_{\mathcal{K}}$ (see (2.5)) involves solving a Fredholm integral equation. This is a highly non-trivial problem, made ever more intractable by the high dimensional data domains typical of machine learning

applications. However, it turns out that we may link $T_{\mathcal{K}}$ with the Gram matrix given by our dataset through the singular value decomposition of linear operators.

Recall the singular value decomposition of a compact linear operator $A : H \to F$ where H, F are Hilbert spaces (see [62, 76, 61]). Let $\{\lambda_i\}_{i=1}^{\infty}, \{u_i\}_{i=1}^{\infty}$ be the eigenvalues and eigenfunctions of A^*A and $\{v_i\}_{i=1}^{\infty}$ be the eigenfunctions of AA^*. Then A may be decomposed as

$$A = \sum_{i=1}^{\infty} \sqrt{\lambda_i} v_i u_i^*, \tag{4.6}$$

where

$$\begin{aligned} A^*Au_i &= \lambda_i u_i \\ AA^*v_i &= \lambda_i v_i \\ Au_i &= \sqrt{\lambda_i} v_i \\ A^*u_i &= \sqrt{\lambda_i} u_i. \end{aligned}$$

The goal then, is to define appropriate operators in order to relate the integral operator $T_{\mathcal{K}}$ and the normalized empirical Gram matrix $\boldsymbol{K}_{ij} := \frac{\mathcal{K}(x_i, x_j)}{n}$ associated with a dataset $\{x_i\}_{i=1}^n$ [76]. To this end, it is helpful to define an operator $T_{\mathbb{H}} : \mathbb{H} \to \mathbb{H}$ equal to the restriction of $T_{\mathcal{K}}$ to $\mathbb{H}$

$$T_{\mathbb{H}}[f](z) := \int_{\mathcal{X}} \langle f, \mathcal{K}(\cdot, x) \rangle_{\mathbb{H}} \mathcal{K}(x, z) d\rho(x). \tag{4.7}$$

Also define an analog of $T_{\mathbb{H}}$ where $\rho(x)$ is estimated via the empirical distribution $\hat{\rho} = \frac{1}{n} \sum_{i=1}^n \delta(x_i)$

$$T_n[f](z) := \frac{1}{n} \sum_{i=1}^{n} \langle K(\cdot, x_i), f \rangle_{\mathbb{H}} K(x_i, z) \tag{4.8}$$

From here, we will connect our four objects of study $\boldsymbol{K}, T_n, T_{\mathbb{H}}, T_{\mathcal{K}}$. We will show that

$\boldsymbol{K} = T_n^*$ and $T_{\mathcal{K}} = T_{\mathbb{H}}^*$. Then, it will be shown that $T_n \to T_{\mathbb{H}}$ as $n \to \infty$ and bounds will be given.

Proposition 4.2.1. *Let R_n be the sampling or restriction operator $R_n : \mathbb{H} \to \mathbb{R}^n$ associated with sample $\{x_i\}_{i=}^n$ where*

$$R_n[f] := [f(x_1), f(x_2), \ldots, f(x_n)]^T. \tag{4.9}$$

Then $\boldsymbol{K} = R_n R_n^*, T_n = R_n^* R_n$ and $K^* = T_n$ with respect to the normalized euclidean inner product $\langle \cdot, \cdot \rangle_n := \frac{\langle \cdot, \cdot \rangle_{\mathbb{R}^n}}{n}$. This inner product is also known as the sampling inner product.

Proof. The adjoint R_n^* with respect to $\langle \cdot, \cdot \rangle_n$ can be seen to be

$$R_n^*[(y_1, \ldots, y_n)](z) = \frac{1}{n} \sum_{i=1}^n y_i \mathcal{K}(x_i, z) \tag{4.10}$$

as

$$\begin{aligned} \langle f, R_n^*(y_1, \ldots, y_n) \rangle_{\mathbb{H}} &= \langle R_n f, (y_1, \ldots, y_n) \rangle_n \\ &= \frac{1}{n} \sum_{i=1}^n y_i f(x_i) \\ &= \frac{1}{n} \sum_{i=1}^n y_i \langle \mathcal{K}(x_i, \cdot), f \rangle_{\mathbb{H}}. \end{aligned}$$

Then, we may see that T_n is equal to $R_n^* R_n$

$$\begin{aligned} R_n^* R_n[f] &= R_n^*[f(x_1), \ldots, f(x_n)] \\ &= \frac{1}{n} \sum_{i=1}^n f(x_i) \mathcal{K}(x_i, z) \\ &= \frac{1}{n} \sum_{i=1}^n \langle f, \mathcal{K}(x_i, \cdot) \rangle_{\mathbb{H}} \mathcal{K}(x_i, z) \\ &= T_n[f](z), \end{aligned}$$

where the last equality is due to the reproducing property (2.3). And $\boldsymbol{K}$ is equal to $R_n R_n^*$ when viewed as an operator from $\mathbb{R}^n \to \mathbb{R}^n$

$$\begin{aligned} R_n R_n^*[(y_1, \ldots, y_n)] &= R_n \left[\frac{1}{n} \sum_{i=1}^n y_i \mathcal{K}(x_i, z) \right] \\ &= \frac{1}{n} \left[\sum_{i=1}^n y_i \mathcal{K}(x_i, x_1), \ldots, \sum_{i=1}^n y_i \mathcal{K}(x_i, x_n) \right]^T. \end{aligned}$$

Finally, $\boldsymbol{K}^* = (R_n R_n^*)^* = R_n^* R_n = T_n$.

□

Proposition 4.2.2. *Let $R_{\mathbb{H}} : \mathbb{H} \to L^2$ be the inclusion map then $T_{\mathcal{K}} = R_{\mathbb{H}} R_{\mathbb{H}}^*$, $T_{\mathbb{H}} = R_{\mathbb{H}}^* R_{\mathbb{H}}$ and $T_{\mathcal{K}}^* = T_{\mathbb{H}}$.*

Proof. Similar to the previous case □

Thus, via Proposition 4.2.1 along with operator SVD, we see that if $\hat{u}^{(j)}, \hat{v}^{(j)}$ are the j-th eigenvector and eigenfunction of $\boldsymbol{K}$ and T_n associated with the singular value λ_j then

$$\hat{v}^{(j)}(z) = \frac{1}{n\sqrt{\lambda_j}} \sum_{i=1}^n \hat{u}_i^{(j)} \mathcal{K}(z, x_i) \tag{4.11}$$

where $\hat{u}_i^{(j)}$ is the i-th component of $\hat{u}^{(j)}$. Additionally, $\hat{u}^{(j)}, \hat{v}^{(j)}$ must be normalized according to the sampling and $\mathbb{H}$ inner products, respectively. Similarly, via Proposition 4.2.1 along with operator SVD, if $u^{(j)}, v^{(j)}$ are the j-th eigenfunctions of $T_{\mathcal{K}}$ and $T_{\mathbb{H}}$ associated with the singular value λ_j then

$$v^{(j)}(z) = \sqrt{\lambda_i} u^{(j)}(z), \quad \text{for } \rho\text{-almost all } x \in \mathcal{X}, \tag{4.12}$$

where $u^{(j)}, v^{(j)}$ are normalized in L^2 and $\mathbb{H}$.

It remains to be shown that the eigenvalues and eigenfunctions of T_n approach those of $T_{\mathbb{H}}$ as n increases. It is shown in Theorem 7 of Rosasco, Belkin, and Vito [76] that T_n and

$T_{\mathbb{H}}$ converge in Hilbert-Schmidt norm at a $\sqrt{n}$ rate. This theorem is reproduced here for convenience.

Theorem 4.2.3 (Theorem 7 of Rosasco, Belkin, and Vito [76]). *With probability at least* $1-\delta$

$$\|T_{\mathbb{H}} - T_n\|_{HS} \leq \sqrt{\frac{8\kappa^2 \log(2/\delta)}{n}}, \tag{4.13}$$

where $\kappa = \sup_{x\in\mathcal{X}} \mathcal{K}(x,x)$.

Theorem 12 of Rosasco, Belkin, and Vito [76] shows that the eigenfunction extension (4.11) can provide a faithful approximation to the true eigenfunctions of $T_{\mathcal{K}}$ with convergence in $L^2(\mathcal{X},\rho)$ norm. Again, this theorem is reproduced for convenience.

Theorem 4.2.4 (Theorem 12 of Rosasco, Belkin, and Vito [76]). *Let* m *be the sum of the multiplicities of the first* N *eigenvalues of* $T_{\mathcal{K}}$ *ordered such that*

$$\lambda_1 > \lambda_2 > \ldots, > \lambda_m > \lambda_{m+1}$$

and P_N *be the orthogonal projection onto the span of the corresponding eigenfunctions. Let* k *be the rank of* $\boldsymbol{K}$ *and* $\hat{u}_1,\ldots,\hat{u}_k$ *be the corresponding eigenvectors to the nonzero eigenvalues of* $\boldsymbol{K}$ *in decreasing order. Finally, let* $\hat{v}_1,\ldots,\hat{v}_k$ *be the corresponding extensions given by* (4.11)*. Then, with probability at least* $1-\delta$ *if*

$$n > \frac{128\kappa^2 \log(2/\delta)}{(\lambda_m - \lambda_{m+1})^2},$$

then

$$\sum_{i=1}^{m} \|\hat{v}_i - P_N\hat{v}_i\|_{L^2}^2 + \sum_{i=m+1}^{k} \|P_N\hat{v}_i\|_{L^2}^2 \leq \frac{32\kappa^2 \log(2/\delta)}{(\lambda_m - \lambda_{m+1})^2 n}. \tag{4.14}$$

Finally, these results combined show that the j-th eigenfunction $u^{(j)}$ of the kernel integral operator T_k can be estimated over the dataset by the j-th eigenvector of $\boldsymbol{K}$.

Furthermore, by combining equations (4.11) and (4.12), we see that this estimation can be extended to the entire domain $\mathcal{X}$ as

$$u^{(j)}(z) \approx \frac{1}{n\lambda_j} \sum_{i=1}^{n} \hat{u}_i^{(j)} \mathcal{K}(x_i, z), \tag{4.15}$$

and in the case of the feature kernel

$$u^{(j)}(z) \approx \frac{1}{n\lambda_j} \sum_{i=1}^{n} \hat{u}_i^{(j)} \varphi_\theta(x_i)^T \varphi_\theta(z). \tag{4.16}$$

Remark 4.1. The bounds in Theorems 4.2.3 and 4.2.4 do not depend on the dimension of the data. This is wonderful news for a machine learning practitioner who often works in very high dimensions. In the case of the feature kernel, the dimension of the feature representation influences the bounds only through κ. This can be eliminated by first normalizing the features.

With these tools in hand, we can approximate Mercer's decomposition of any kernel from data. This is illustrated and empirically verified through a simple example.

Example 4.1 (Polynomial Kernel)**.** Consider the kernel $\mathcal{K}(x, z) = (1 + xz)^2$ where $\mathcal{X} = [-1, 1]$ and μ is the uniform measure over $\mathcal{X}$. Then, it is known that the integral operator

$$T_{\mathcal{K}}[f](x) = \int_{\mathbb{R}} (1 + xz)^2 f(z) \frac{dz}{2}$$

has the (L^2 normalized) eigenfunctions and eigenvalues given by [89, p. 397]

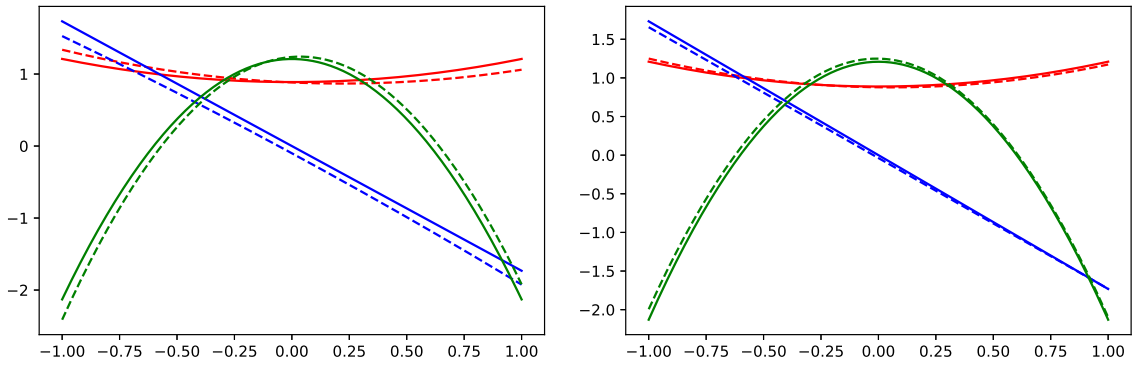
$$\varphi_1 = \frac{1}{1.06}(0.94 + 0.34x^2), \qquad \lambda_1 = 1.12 \tag{4.17a}$$

$$\varphi_2 = \frac{-1}{0.58}x, \qquad \lambda_2 = 0.67 \tag{4.17b}$$

$$\varphi_3 = \frac{1}{0.28}(0.34 - 0.94x^2), \qquad \lambda_3 = 0.08 \tag{4.17c}$$

In order to empirically verify the theory, we sample $n = 10$ and $n = 100$ points uniformly

from $[-1, 1]$ and form and decompose the Gram matrix before plotting the estimated and true eigenfunctions in Figure 4.1.



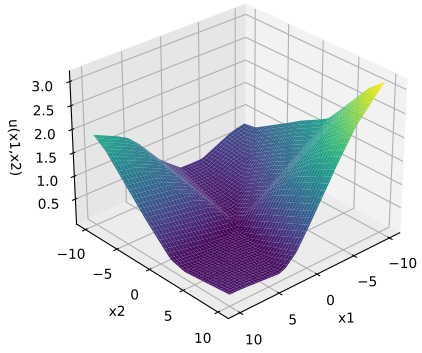
(a) $n = 100$ **(b)** $n = 1000$

Figure 4.1: *The estimated (**dashed**) and true eigenfunctions (**solid**) of the polynomial kernel from Example 4.1. We see that the empirical estimation agrees with the theory and improves with the sample size. Red, blue, and green lines correspond to φ_1, φ_2, and φ_3 respectively.*

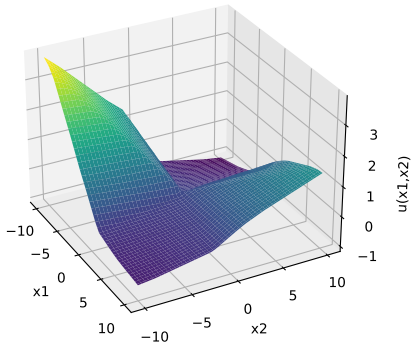
Example 4.2 (Neural Feature Kernel)**.** We now a train a neural network to classify data generated by the XOR function. The XOR function $\mathrm{xor} : [-10, 10]^2 \to \{\pm 1\}$ is given by

$$\mathrm{xor}([x_1, x_2]) = \begin{cases} 1 & (x_1 \geq 0 \,\text{and}\, x_2 < 0) \,\text{or}\, (x_1 < 0 \,\text{and}\, x_2 \geq 0) \\ -1 & (x_1 > 0 \,\text{and}\, x_2 > 0) \,\text{or}\, (x_1 \leq 0 \,\text{and}\, x_2 \leq 0) \end{cases}$$

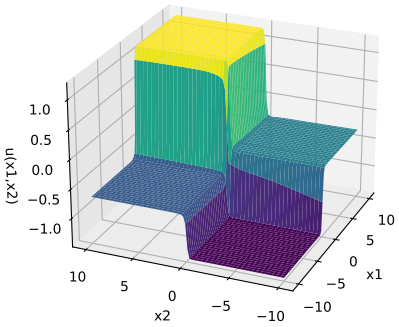
$1,000$ sample points are generated uniformly from $[-10, 10]^2$ and given labels according to XOR. The neural network is a simple 3-layer MLP as in Section 1.2, with a feature dimension $d = 10$. The trained network achieves 100% accuracy on a 100 sample test set. We repeat the experiment using both the ReLU and hyperbolic tangent activation functions. In Figure 4.2, we plot the first two Mercer features. It is interesting to note the remarkable differences between ReLU and tanh networks. Inspecting the functions, it appears that the first Mercer features respond highly when $\mathrm{xor}(x) = 1$ and the second Mercer features respond highly when $\mathrm{xor}(x) = -1$.



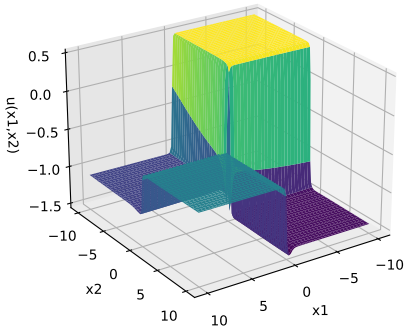
(a) $u^{(1)}$ *for the ReLU network.* $\lambda_1 = 132.45$.



(b) $u^{(2)}$ *for the ReLU network.* $\lambda_2 = 76.58$.



(c) $u^{(1)}$ *for the tanh network.* $\lambda_1 = 2.862$.



(d) $u^{(2)}$ *for the tanh network.* $\lambda_2 = 2.70$.

Figure 4.2: *The first two Mercer feature functions for both the ReLU and the tanh networks. Note the remarkable difference between functions with different activations; however, across networks, it appears that the first Mercer feature responds highly when* $\mathrm{xor}(x) = 1$ *and the second Mercer feature responds highly when* $\mathrm{xor}(x) = -1$.

4.2.1 *Relationship Between Neural Network Features and Mercer Eigenfunctions*

For simplicity, assume that $\mathcal{X} = \mathbb{R}^D$ and we have trained a neural network to produce features $\varphi_\theta : \mathbb{R}^D \to \mathbb{R}^d$. Consider the neural feature kernel. Then the eigenfunctions of T_K guaranteed by Mercer's theorem are estimated by (4.16). We thus have a set of non-zero eigenvalues $\{\lambda_i\}_{i=1}^k$ and a set of feature functions $u^{(j)}, j = 1, \ldots, k$ whose outputs we may form into a vector giving a new representation of data in $\mathcal{X}$.

$$\varphi_M(z) = [u^{(1)}(z), \ldots, u^{(k)}(z)] \tag{4.18}$$

We'll refer to these features as the Mercer features. It should be noted that eigenfunctions are unique only up to sign. For this reason, one may sometimes wish to flip the sign of a given $u^{(j)}$. The key benefit of these features is that they are given an importance ranking from the associated eigenvalues. The expressivity of the Mercer features is the same as the learned features φ_θ. This may be seen by establishing a linear relationship between the two sets of features. Let

$$b_j = \frac{1}{n}\sum_{i=1}^{n} u_i^{(j)} \varphi_\theta(x_i), \quad j = 1, \ldots, k$$

$$b_j = \mathbf{0}, \quad j = k+1, \ldots, d$$

and define the matrices $B = [b_1, \ldots, b_d] \in \mathbb{R}^{k \times d}$ and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_k)$ then

$$\varphi_M(z) = \Lambda^{-1} B^T \varphi_\theta(z).$$

Thus, linear models built on top of either φ_M or φ_θ will be equally as expressive. See that

$$\begin{aligned} f(z) &= w^T \varphi_M(z) \\ &= w^T \Lambda^{-1} B^T \varphi_\theta(z) \\ &= v^T \varphi_\theta(z), \quad \text{for some } w, v \in \mathbb{R}^k \end{aligned}$$

In Section 4.2.3, we perform some experiments on three benchmark datasets showing that the ordering of the eigenvalues associated with the the Mercer features corresponds to their usefulness in downstream tasks.

4.2.2 *Differentiating the Mercer Eigenfunctions*

The extended Mercer eigenfunctions or Mercer feature functions (4.16) are differentiable with respect to the input z. Having learned neural network features φ_θ and found the associated Mercer feature functions, this differentiability may be used to find the argmin or argmax of the Mercer features. This could lend insight into how the neural network is forming the features. For example, given $x \in \operatorname{argmax}_{z\in\mathcal{X}} |u^{(1)}(z)|$ an inspection of x yields insight into what aspects of the data provoke the highest response from the most informative Mercer feature. The absolute value must be taken as eigenfunctions are unique only up to sign.

The gradient of a Mercer feature $u^{(j)}$ may be found as follows. Define the Jacobian $J(z)$ as usual with $J_{ij}(z) = \frac{d\varphi_i(z)}{dz_j}, i \in [d], j \in [D]$. Then

$$\nabla u^{(j)}(z) = \frac{1}{n\lambda_j} \sum_{i=1}^{n} \hat{u}_i^{(j)} J(z)^T \varphi_\theta(x_i) \tag{4.19}$$

The Jacobian is easy to compute using commonly found autograd functionality in deep learning libraries [69, 57]. This makes it straightforward to optimize the eigenfunction using gradient ascent type algorithms.

4.2.3 *Eigenvalues Provide an Importance Score*

We seek to demonstrate that the eigenvalues associated with the Mercer decomposition of the feature kernel act as an importance score for the Mercer features. Let $\varphi_\theta : \mathbb{R}^D \to \mathbb{R}^d$ be a learned representation function and $\mathcal{K}(x, z) = \varphi_\theta^T(x)\varphi_\theta$ be the associated neural feature kernel. Given access to a dataset $S = \{x_i\}_{i=1}^n$, we approximate the Mercer decomposition as described above resulting in a collection of Mercer features $\{u^{(i)}\}_{i=1}^k$ and their associated eigenvalues $\{\lambda_i\}_{i=1}^k$. Define

$$\varphi_M^\alpha(z) = [u^{(1)}(z), \ldots, u^{(\alpha)}(z)], \quad 1 \le \alpha \le k, \tag{4.20}$$

to be the Mercer feature vector truncated to only the first α components. Similarly, define

$$\varphi_M^{\beta}(z) = [u^{(k)}(z), \ldots, u^{(k-\beta)}(z)], \quad 1 \leq \beta \leq k, \tag{4.21}$$

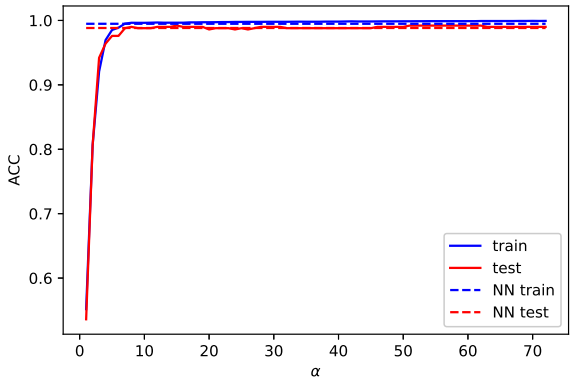
to be the Mercer feature vector truncated to the first β components.

We may then compose linear classifiers with the truncated feature functions $\varphi_M^{\alpha}, \varphi_M^{\beta}$ resulting finally with models in the form of Equation 1.11

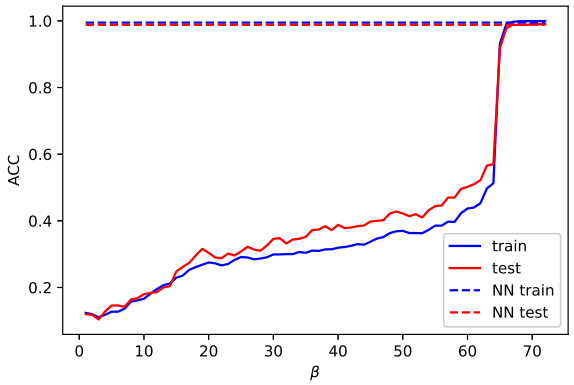
$$f_M^{\alpha}(z) = w^T \varphi_M^{\alpha}(z) \tag{4.22a}$$

$$f_M^{\beta}(z) = w^T \varphi_M^{\beta}(z). \tag{4.22b}$$

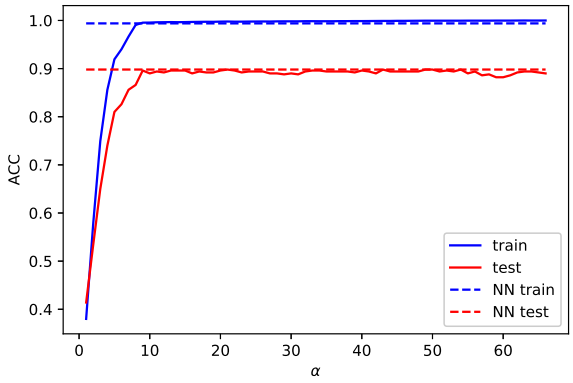
It should be expected that using φ_M^{α} should provide good performance even for low values of α as the purportedly most informative features are preferentially included. On the other hand, using φ_M^{β} will give a poor performance for low values of β as the purportedly least informative features are included first. We test this theory using a ResNet-18 model [41] on three datasets: MNIST [26] , CIFAR-10 [51], and ImageNet [78]. When building the Gram matrix for each dataset, we first take a random subset of $10,000$ samples from the original dataset. This avoids the decomposition of an otherwise very large Gram matrix while obtaining a high-quality approximation. The results are available in Figure 4.3. We see that eigenvalues rank the utility of the associated Mercer features. The f_M^{α} models quickly match the accuracy of the full neural network. However, f_M^{β} models do not achieve parity with the neural networks until the highest ranked features are included in the model. This provides empirical evidence that the eigenvalues are valuable tools in judging the importance of these features. This gives insight into the workings of a neural network as one could inspect highly ranked Mercer features.



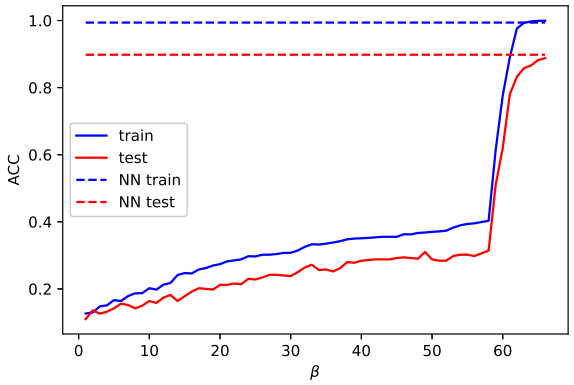
(a) f_M^{α} *on MNIST.*



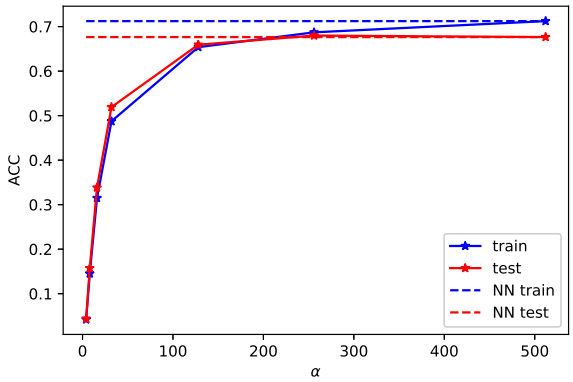
(b) f_M^{β} *on MNIST.*



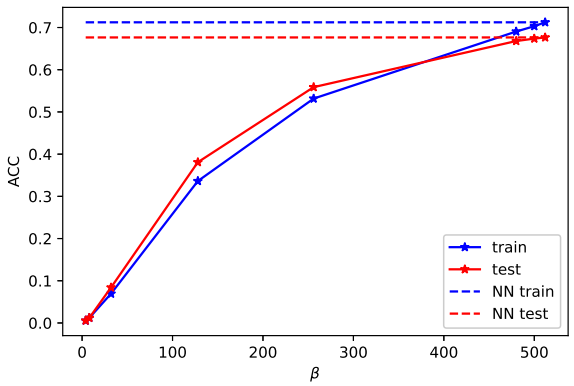
(c) f_M^{α} *on CIFAR.*



(d) f_M^{β} *on CIFAR.*



(e) f_M^{α} *on ImageNet.*



(f) f_M^{β} *on ImageNet.*

Figure 4.3: *We see that eigenvalues rank the utility of the associated Mercer features. The* f_M^{α} *models quickly match the accuracy of the full neural network. However,* f_M^{β} *models do not achieve parity with the neural networks until the highest ranked features are included in the model. We see that the ImageNet trained model has learned features whose value decays slower. It seems likely that the difficulty of the ImageNet task forced the model to learn more complex features leading to a fuller use of the 512 dimensional feature space. A Mercer feature was included in this experiment if its associated eigenvalue was significant to two digits. Due to computational constraints only a subset of possible* k *values were evaluated for the ImageNet models. These are shown with a star.*

CHAPTER 5

CONCLUSION

In this thesis, we argued that feature learning provides a lens with which to view both neural networks and kernel machines. To support this view, we introduce the theory of reproducing kernel Hilbert spaces from a feature learning perspective, paying particular attention to the induced feature spaces associated with any kernel. Neural networks are viewed as the composition of a learned feature function with a linear model. Understanding neural networks from an RKHS perspective could allow for improved generalization bounds or better insight into the nature of neural networks. We examine some recent work connecting kernels to neural networks and contrast these works with our view. Finally, we apply our developed theory of RKHS and kernels from a feature learning perspective by introducing the neural feature kernel. This allows us to approximate the norm of a neural network in the RKHS associated with the neural feature kernel, which may be used to apply generalization bounds. Additionally, we describe a method to find eigenvalues and eigenfunctions of the neural feature kernel. This gives a set of features with a utility score, essentially describing their usefulness in the task at hand. There is much room for future work looking into potential applications for these features. Another possible direction of study is using batched approximations of the neural feature kernel to shape the learned RKHS during the training process.

BIBLIOGRAPHY

[1] Mark A. Aizerman. "Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning". In: *Automation and Remote Control* 25 (1964), pp. 821–837.

[2] N. Aronszajn. "Theory of reproducing kernels". In: *Transactions of the American Mathematical Society* 68.3 (1950), pp. 337–404. DOI: 10.1090/s0002-9947-1950-0051437-7. URL: https://doi.org/10.1090/s0002-9947-1950-0051437-7.

[3] N. Aronszajn and K. T. Smith. "Characterization of Positive Reproducing Kernels. Applications to Green's Functions". In: *American Journal of Mathematics* 79.3 (July 1957), p. 611. DOI: 10.2307/2372565. URL: https://doi.org/10.2307/2372565.

[4] Nachman Aronszajn. "Reproducing and pseudo-reproducing kernels and their application to the partial differential equations of physics". In: ().

[5] Philip Bachman, R Devon Hjelm, and William Buchwalter. "Learning Representations by Maximizing Mutual Information Across Views". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/ddf354219aac374f1d40b7e760ee5bb7-Paper.pdf.

[6] A.R. Barron. "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Transactions on Information Theory* 39.3 (1993), pp. 930–945. DOI: 10.1109/18.256500.

[7] Peter L. Bartlett et al. "Benign overfitting in linear regression". In: *Proceedings of the National Academy of Sciences* 117.48 (Apr. 2020), pp. 30063–30070. DOI: 10.1073/pnas.1907378117. URL: https://doi.org/10.1073/pnas.1907378117.

[8] Mikhail Belkin. *Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation*. 2021. DOI: 10.48550/ARXIV.2105.14368. URL: https://arxiv.org/abs/2105.14368.

[9] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. *To understand deep learning we need to understand kernel learning*. 2018. DOI: 10.48550/ARXIV.1802.01396. URL: https://arxiv.org/abs/1802.01396.

[10] Mikhail Belkin et al. "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32 (July 2019), pp. 15849–15854. DOI: 10.1073/pnas.1903070116. URL: https://doi.org/10.1073/pnas.1903070116.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. *Representation Learning: A Review and New Perspectives*. 2014. DOI: 10.48550/ARXIV.1206.5538. URL: https://arxiv.org/abs/1206.5538.

[12] Stefan Bergmann. "Über die Kernfunktion eines Bereiches und ihr Verhalten am Rande. I." In: *crll* 1933.169 (1933), pp. 1–42. DOI: 10.1515/crll.1933.169.1. URL: https://doi.org/10.1515/crll.1933.169.1.

[13] Alain Berlinet and Christine Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer US, 2004. DOI: 10.1007/978-1-4419-9096-9. URL: https://doi.org/10.1007/978-1-4419-9096-9.

[14] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2021. DOI: 10.48550/ARXIV.2108.07258. URL: https://arxiv.org/abs/2108.07258.

[15] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*. ACM Press, 1992. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[16] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[17] Mathilde Caron et al. *Unsupervised Learning of Visual Features by Contrasting Cluster Assignments*. 2020. DOI: 10.48550/ARXIV.2006.09882. URL: https://arxiv.org/abs/2006.09882.

[18] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *arXiv preprint arXiv:2002.05709* (2020).

[19] Youngmin Cho and Lawrence Saul. "Kernel Methods for Deep Learning". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio et al. Vol. 22. Curran Associates, Inc., 2009. URL: https://proceedings.neurips.cc/paper/2009/file/5751ec3e9a4feab575962e78e006250d-Paper.pdf.

[20] S. Chopra, R. Hadsell, and Y. LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 539–546 vol. 1. DOI: 10.1109/CVPR.2005.202.

[21] Michael Chui et al. *The state of AI in 2021*. 2021. URL: https://www.mckinsey.com/business-functions/quantumblack/our-insights/global-survey-the-state-of-ai-in-2021.

[22] Adam Coates and Andrew Y. Ng. "Learning Feature Representations with K-Means". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 561–580. DOI: 10.1007/978-3-642-35289-8_30. URL: https://doi.org/10.1007/978-3-642-35289-8_30.

[23] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/bf00994018. URL: https://doi.org/10.1007/bf00994018.

[24] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/bf02551274. URL: https://doi.org/10.1007/bf02551274.

[25] Amit Daniely, Roy Frostig, and Yoram Singer. "Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/abea47ba24142ed16b7d8fbf2c740e0d-Paper.pdf.

[26] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[27] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. "Unsupervised Visual Representation Learning by Context Prediction". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Dec. 2015. DOI: 10.1109/iccv.2015.167. URL: https://doi.org/10.1109/iccv.2015.167.

[28] Pedro Domingos. *Every Model Learned by Gradient Descent Is Approximately a Kernel Machine*. 2020. DOI: 10.48550/ARXIV.2012.00152. URL: https://arxiv.org/abs/2012.00152.

[29] Petros Drineas and Michael W. Mahoney. "On the Nystrom Method for Approximating a Gram Matrix for Improved Kernel-Based Learning". In: *Journal of Machine Learning Research* 6.72 (2005), pp. 2153–2175. URL: http://jmlr.org/papers/v6/drineas05a.html.

[30] Harris Drucker et al. "Support Vector Regression Machines". In: *Advances in Neural Information Processing Systems*. Ed. by M.C. Mozer, M. Jordan, and T. Petsche. MIT

Press, 1996. DOI: 10.5555/2998981.2999003. URL: https://proceedings.neurips.cc/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf.

[31] David Kristjanson Duvenaud. "Automatic Model Construction with Gaussian Processes". PhD thesis. 2014.

[32] Zhou Fan and Zhichao Wang. *Spectra of the Conjugate Kernel and Neural Tangent Kernel for linear-width neural networks*. 2020. DOI: 10.48550/ARXIV.2005.11879. URL: https://arxiv.org/abs/2005.11879.

[33] "Functions of positive and negative type, and their connection with the theory of integral equations". In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 83.559 (Nov. 1909), pp. 69–70. DOI: 10.1098/rspa.1909.0075. URL: https://doi.org/10.1098/rspa.1909.0075.

[34] Claudio Gallicchio and Simone Scardapane. *Deep Randomized Neural Networks*. 2020. DOI: 10.48550/ARXIV.2002.12287. URL: https://arxiv.org/abs/2002.12287.

[35] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. *Unsupervised Representation Learning by Predicting Image Rotations*. 2018. DOI: 10.48550/ARXIV.1803.07728. URL: https://arxiv.org/abs/1803.07728.

[36] Will Grathwohl et al. *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models*. 2018. DOI: 10.48550/ARXIV.1810.01367. URL: https://arxiv.org/abs/1810.01367.

[37] Jean-Bastien Grill et al. "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 21271–21284. URL: https://proceedings.neurips.cc/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf.

[38] Jeff Z. HaoChen et al. *Provable Guarantees for Self-Supervised Deep Learning with Spectral Contrastive Loss*. 2021. DOI: 10.48550/ARXIV.2106.04156. URL: https://arxiv.org/abs/2106.04156.

[39] Trevor Hastie, Robert Tibshirani, and J H Friedman. *The elements of statistical learning*. en. 2nd ed. Springer series in statistics. New York, NY: Springer, Dec. 2009.

[40] Bobby He and Mete Ozay. "Feature Kernel Distillation". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=tBIQEvApZK5.

[41] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[42] Kaiming He et al. "Momentum Contrast for Unsupervised Visual Representation Learning". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 9726–9735. DOI: 10.1109/CVPR42600.2020.00975.

[43] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (July 2006), pp. 504–507. DOI: 10.1126/science.1127647. URL: https://doi.org/10.1126/science.1127647.

[44] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[45] Zhengmian Hu and Heng Huang. "On the Random Conjugate Kernel and Neural Tangent Kernel". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 4359–4368. URL: https://proceedings.mlr.press/v139/hu21b.html.

[46] Tommi Jaakkola and David Haussler. "Exploiting Generative Models in Discriminative Classifiers". In: *Advances in Neural Information Processing Systems*. Ed. by M. Kearns, S. Solla, and D. Cohn. Vol. 11. MIT Press, 1998. URL: https://proceedings.neurips.cc/paper/1998/file/db1915052d15f7815c8b88e879465a1e-Paper.pdf.

[47] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks". In: (2018). DOI: 10.48550/ARXIV.1806.07572. URL: https://arxiv.org/abs/1806.07572.

[48] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (July 2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2. URL: https://doi.org/10.1038/s41586-021-03819-2.

[49] George Kimeldorf and Grace Wahba. "Some results on Tchebycheffian spline functions". In: *Journal of Mathematical Analysis and Applications* 33.1 (Jan. 1971), pp. 82–95. DOI: 10.1016/0022-247x(71)90184-3. URL: https://doi.org/10.1016/0022-247x(71)90184-3.

[50] George S. Kimeldorf and Grace Wahba. "A Correspondence Between Bayesian Estimation on Stochastic Processes and Smoothing by Splines". In: *The Annals of Mathematical Statistics* 41.2 (Apr. 1970), pp. 495–502. DOI: 10.1214/aoms/1177697089. URL: https://doi.org/10.1214/aoms/1177697089.

[51] A. Krizhevsky and G. Hinton. "Learning Multiple Layers of Features from Tiny Images". In: *Master's thesis, Department of Computer Science, University of Toronto* (2009).

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[53] Julius von Kügelgen et al. "Self-Supervised Learning with Data Augmentations Provably Isolates Content from Style". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 16451–16467. URL: https://proceedings.neurips.cc/paper/2021/file/8929c70f8d710e412d38da624b21c3c8Paper.pdf.

[54] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. "Learning Representations for Automatic Colorization". In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 577–593. DOI: 10.1007/978-3-319-46493-0_35. URL: https://doi.org/10.1007/978-3-319-46493-0_35.

[55] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541. URL: https://doi.org/10.1162/neco.1989.1.4.541.

[56] Jaehoon Lee et al. *Deep Neural Networks as Gaussian Processes*. 2017. DOI: 10.48550/ARXIV.1711.00165. URL: https://arxiv.org/abs/1711.00165.

[57] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[58] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[59] Marvin Minsky and Seymour Papert. *Perceptrons*. London, England: MIT Press, Feb. 1969.

[60] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. en. Adaptive Computation and Machine Learning series. London, England: MIT Press, Aug. 2012.

[61] Mattes Mollenhauer et al. "Singular Value Decomposition of Operators on Reproducing Kernel Hilbert Spaces". In: *Advances in Dynamics, Optimization and Computation*. Springer International Publishing, 2020, pp. 109–131. DOI: 10.1007/978-3-030-51264-4_5. URL: https://doi.org/10.1007%2F978-3-030-51264-4_5.

[62] Valter Moretti. *Spectral theory and quantum mechanics*. en. 2nd ed. La Matematica per il 3+2. Basel, Switzerland: Springer International Publishing, Feb. 2018.

[63] Radford M. Neal. "Bayesian Learning for Neural Networks". In: 1995. URL: http://www.cs.toronto.edu/~radford/ftp/thesis.pdf.

[64] Alex Nichol et al. *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. 2021. DOI: 10.48550/ARXIV.2112.10741. URL: https://arxiv.org/abs/2112.10741.

[65] Mehdi Noroozi and Paolo Favaro. "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles". In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 69–84. DOI: 10.1007/978-3-319-46466-4_5. URL: https://doi.org/10.1007/978-3-319-46466-4_5.

[66] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu koray. "Neural Discrete Representation Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

[67] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2018. DOI: 10.48550/ARXIV.1807.03748. URL: https://arxiv.org/abs/1807.03748.

[68] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2018. DOI: 10.48550/ARXIV.1807.03748. URL: https://arxiv.org/abs/1807.03748.

[69] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[70] Deepak Pathak et al. "Context Encoders: Feature Learning by Inpainting". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2536–2544.

[71] Debleena Paul et al. "Artificial intelligence in drug discovery and development". In: *Drug Discovery Today* 26.1 (Jan. 2021), pp. 80–93. DOI: 10.1016/j.drudis.2020.10.010. URL: https://doi.org/10.1016/j.drudis.2020.10.010.

[72] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[73] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. DOI: 10.48550/ARXIV.2103.00020. URL: https://arxiv.org/abs/2103.00020.

[74] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. DOI: 10.48550/ARXIV.2204.06125. URL: https://arxiv.org/abs/2204.06125.

[75] Paul Renteln and Alan Dundes. "Foolproof : A Sampling of Mathematical Folk Humor". In: *Notices of the American Mathematical Society*. Vol. 52. AMS, 2005.

[76] Lorenzo Rosasco, Mikhail Belkin, and Ernesto De Vito. "On Learning with Integral Operators". In: *Journal of Machine Learning Research* 11.30 (2010), pp. 905–934. URL: http://jmlr.org/papers/v11/rosasco10a.html.

[77] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. DOI: 10.48550/ARXIV.1609.04747. URL: https://arxiv.org/abs/1609.04747.

[78] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[79] Andrew M. Saxe et al. "On Random Weights and Unsupervised Feature Learning". In: *ICML*. 2011, pp. 1089–1096. URL: https://icml.cc/2011/papers/551_icmlpaper.pdf.

[80] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001. ISBN: 0262194759.

[81] Amit Singhal. "Modern Information Retrieval: A Brief Overview." In: *IEEE Data Eng. Bull.* 24.4 (2001), pp. 35–43. URL: http://singhal.info/ieee2001.pdf.

[82] James Stewart. "Positive definite functions and generalizations, an historical survey". In: *Rocky Mountain Journal of Mathematics* 6.3 (Sept. 1976). DOI: 10.1216/rmj-1976-6-3-409. URL: https://doi.org/10.1216/rmj-1976-6-3-409.

[83] Xiaoyuan Su and Taghi M Khoshgoftaar. "A survey of collaborative filtering techniques". en. In: *Adv. Artif. Intell.* 2009 (Oct. 2009), pp. 1–19.

[84] Niko Sünderhauf et al. "The limits and potentials of deep learning for robotics". In: *The International Journal of Robotics Research* 37.4-5 (Apr. 2018), pp. 405–420. DOI: 10.1177/0278364918770733. URL: https://doi.org/10.1177/0278364918770733.

[85] Richard S Sutton and Andrew G Barto. *Reinforcement Learning*. en. Adaptive Computation and Machine Learning series. Cambridge, MA: Bradford Books, Feb. 1998.

[86] Amirsina Torfi et al. *Natural Language Processing Advancements By Deep Learning: A Survey*. 2020. DOI: 10.48550/ARXIV.2003.01200. URL: https://arxiv.org/abs/2003.01200.

[87] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[88] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research* 11.110 (2010), pp. 3371–3408. URL: http://jmlr.org/papers/v11/vincent10a.html.

[89] Martin J. Wainwright. *High-Dimensional Statistics*. Cambridge University Press, Feb. 2019. DOI: 10.1017/9781108627771. URL: https://doi.org/10.1017/9781108627771.

[90] Christopher Williams. "Computing with Infinite Networks". In: *Advances in Neural Information Processing Systems*. Ed. by M.C. Mozer, M. Jordan, and T. Petsche. Vol. 9. MIT Press, 1996. URL: https://proceedings.neurips.cc/paper/1996/file/ae5e3ce40e0404a45ecacaaf05e5f735-Paper.pdf.

[91] Blake Woodworth et al. "Kernel and Rich Regimes in Overparametrized Models". In: *Proceedings of Thirty Third Conference on Learning Theory*. Ed. by Jacob Abernethy and Shivani Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, 2020, pp. 3635–3673. URL: https://proceedings.mlr.press/v125/woodworth20a.html.

[92] Stanislaw Zaremba. "L'équation biharmonique et une class remarquable de functions fondamentales harmoniques". In: *Bulletin International de l'Acadmie des Sciences de Cracovie* (1907).

[93] Jure Zbontar et al. *Barlow Twins: Self-Supervised Learning via Redundancy Reduction*. 2021. DOI: 10.48550/ARXIV.2103.03230. URL: https://arxiv.org/abs/2103.03230.

[94] Chiyuan Zhang et al. *Understanding deep learning requires rethinking generalization*. 2016. DOI: 10.48550/ARXIV.1611.03530. URL: https://arxiv.org/abs/1611.03530.

[95] Ruixiang Zhang et al. *Learning Representation from Neural Fisher Kernel with Low-rank Approximation*. 2022. DOI: 10.48550/ARXIV.2202.01944. URL: https://arxiv.org/abs/2202.01944.

[96] Shengjia Zhao, Jiaming Song, and Stefano Ermon. "Learning Hierarchical Features from Deep Generative Models". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 4091–4099. URL: https://proceedings.mlr.press/v70/zhao17c.html.

[97] Roland S. Zimmermann et al. "Contrastive Learning Inverts the Data Generating Process". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 12979–12990. URL: https://proceedings.mlr.press/v139/zimmermann21a.html.