Kernel Approximation Methods for Speech Recognition

Avner May

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2018

ABSTRACT

Kernel Approximation Methods for Speech Recognition

Avner May

Over the past five years or so, deep learning methods have dramatically improved the state of the art performance in a variety of domains, including speech recognition, computer vision, and natural language processing. Importantly, however, they suffer from a number of drawbacks:

1. Training these models is a non-convex optimization problem, and thus it is difficult to guarantee that a trained model minimizes the desired loss function.

2. These models are difficult to interpret. In particular, it is difficult to explain, for a given model, why the computations it performs make accurate predictions.

In contrast, kernel methods are straightforward to interpret, and training them is a convex optimization problem. Unfortunately, solving these optimization problems *exactly* is typically prohibitively expensive, though one can use *approximation* methods to circumvent this problem. In this thesis, we explore to what extent kernel approximation methods can compete with deep learning, in the context of large-scale prediction tasks. Our contributions are as follows:

1. We perform the most extensive set of experiments to date using kernel approximation methods in the context of large-scale speech recognition tasks, and compare performance with deep neural networks.

2. We propose a feature selection algorithm which significantly improves the performance of the kernel models, making their performance competitive with fully-connected feedforward neural networks.

3. We perform an in-depth comparison between two leading kernel approximation strategies — random Fourier features [Rahimi and Recht, 2007] and the Nyström method [Williams and

Seeger, 2001] — showing that although the Nyström method is better at approximating the kernel, it performs worse than random Fourier features when used for learning.

We believe this work opens the door for future research to continue to push the boundary of what is possible with kernel methods. This research direction will also shed light on the question of when, if ever, deep models are needed for attaining strong performance.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my advisor Michael Collins for his years of guidance and valuable feedback. In 2013, when I first met with Mike, I knew very little about machine learning or speech recognition. He brought me from that day to this one. He did this in a number of ways:

1. *He immediately gave me a concrete problem to work on.* When I first spoke with Mike, he told me about a multi-class classification problem important for speech recognition, and told me to dive right in. This allowed me to get my feet wet right away, and begin to ask the questions necessary to tackle this problem. Little did I know, I would spend the next four years working on it.

2. *He was hands-on when necessary.* I remember sitting with Mike, working to debug my first implementation of SGD for acoustic model training. It's not often you hear about an advisor stepping through code with a student.

3. *He frequently provided guidance on promising directions to pursue.* At so many points during my PhD, Mike has listened to me go through a number of ideas, helping me think about them in new ways, and choose amongst them. He draws from his wealth of knowledge and experience to point me in the most interesting and promising directions. I still remember when he first pointed me to the original random Fourier feature paper.

4. *He always provided me with honest feedback.* One of the things I most appreciate about Mike is the very clear way in which he presents feedback. By holding my work to his very high standards, and telling me exactly where it currently falls short, he pushes me well beyond where I would be able to get on my own.

5. *He gave me the freedom to explore ideas.* Often during my PhD, I have spent time studying topics not directly necessary to advance my research, but which I was excited about. Mike

has graciously allowed me to do this, gently pulling me back to shore when necessary.

Thank you, Mike, for all the time, energy, and guidance you have given me over the years. You have been instrumental in my reaching this point, and our time together has set the intellectual foundation upon which my future work will build.

I would also like to thank my many collaborators during my time at Columbia. Brian Kingsbury was an invaluable resource. He was instrumental in getting me up and running with all the speech recognition datasets, as well as with all the code necessary to decode the acoustic models I trained. Daniel Hsu has been an important mentor, who helped considerably with the feature selection work in particular. He is always extremely generous with his time, and his insights help me understand the problems I am working on in new ways. Thanks to Fei Sha and his group at USC for our extended collaboration on using random Fourier features for acoustic modeling. It was extremely helpful to have another group with which to discuss ideas, and compare results. The work on Entropy Regularized Perplexity came from their group, and was important for improving the performance of our acoustic models.

During my PhD, I was fortunate to have the opportunity to intern at both Microsoft Research in Redmond, WA, and Google Research in New York City. At Microsoft, I worked with the speech recognition group, and was mentored by Jasha Droppo. At Google, I worked in Sanjiv Kumar's team, and was mentored by Jeffrey Pennington. Thanks to both groups for welcoming me and teaching me so much.

Prior to beginning my work with Mike, I did research for two years under Professor Augustin Chaintreau. I am deeply indebted to him for all the energy and time he gave me during those two years, and for introducing me to what it really means to do research. I was always impressed by his boundless energy, sharp intellect, and profound kindness and patience. During this time we collaborated closely with Silvio Lattanzi and Nitish Korula at Google Research, two brilliant researchers and kind people from whom I learned a lot.

A special thanks to my PhD committee—David Blei, Shih-Fu Chang, Michael Collins, Daniel Hsu, and Brian Kingsbury—for all the time and energy they have put into reviewing my work, and providing useful feedback and questions.

My time at Columbia would not have been the same without all the friends I made during my time here. Thanks to Karl Stratos for being a close friend and intellectual partner throughout my entire PhD. He is actually the one who first recommended I speak with Mike regarding advising, and for that I am forever indebted. Karl and I spent many hours talking about machine learning (through our “chevruta”), as well as about many other topics (lifestyle, fitness, nutrition, politics, religion, etc.), and I always appreciate hearing his perspective. I admire his love of learning, and endless dedication to self-improvement. Victor Soto has been my officemate and friend for over four years, and has brought much needed levity and companionship to my days. His laughter flows easily, brightening days that would otherwise be quite solitary. Thanks also to Sasha Rush and Yin-Wen Chang for their friendship and mentorship after I joined Mike’s research group, and for their patience with all my questions. Thanks to all my other friends and colleagues at Columbia: Mohammad Rasooli, Erica Cooper, Sarah Ita Levitan, Anna Prokofieva, Andrei Simion, Chris Kedzie, Noura Farra, Tom Effland, Daniel Bauer, Chris Riederer, Arthi Ramachandran, Mathias Lécuyer, and many others. This ride would have been a lot quieter, and nowhere near as fun, without you all.

I could not have made it to this point without the constant love and support of my entire family. Thanks to both of my sisters, Yael and Orly, for always being there, whether with a helping hand, an empathetic ear, or simply a hug. There are few things I enjoy more than going on runs with them and catching up. Thanks to my brother-in-law Jona and soon-to-be brother-in-law Zev for their friendship, great energy, and for making my sisters happy. And thank you to my niece, Aliza, and my nephew, Benji, for bringing me so much joy every time I see them. And most of all, thank you to my parents, Belly and Ernesto. It’s taken me 30 years to get to this point, and they have been by my side, and rooting for me, *every* step of the way. They have taught me by example the importance of putting my full energy behind all of my pursuits, and of always enjoying the ride. There is no way I could ever thank them enough. This work is dedicated to them.

Dedicated to my parents.

Chapter 1

Introduction

A basic computational problem is to compute an output $y \in \mathcal{Y}$ based on an input $x \in \mathcal{X}$. For example, x could correspond to a vector of real numbers, and y could correspond to the maximum element in that list. Programming languages provide a medium for precisely expressing the way in which y should be computed, given x. However, in many cases, it is very difficult to know, a priori, how to compute y, given x. For example, given a picture, represented by its pixels, how can we compute what is in the picture? Or given an audio recording, how can we predict what words were pronounced? In order to address these challenging scenarios, the field of supervised machine learning takes the following approach: gather as many examples (x_i, y_i) as possible, define a family of functions $\mathcal{F}$ from $\mathcal{X}$ to $\mathcal{Y}$, and find the function $f^* \in \mathcal{F}$ minimizing some notion of error on the examples you gathered. For example:

$$f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} L(f(x_i), y_i).$$

Here, $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a function assigning penalty $L(y', y)$ for predicting the label y' instead of the true label y. Note that if $\mathcal{Y}$ is a discrete set, this is called *classification*, whereas if $\mathcal{Y} \subseteq \mathbb{R}$, this is called *regression*. The goal of this *learning* process is to find a function $f^* : \mathcal{X} \to \mathcal{Y}$ which *generalizes* well to unseen data; in particular, we would like the expected penalty $\mathbb{E}_{X,Y}\left[L(f^*(X), Y)\right]$ on a random (X, Y) pair to be low.

For a variety of problems, a *linear* mapping between x and y is sufficient. Note that in the context of regression, linear models are defined as the functions of the form $f(x) = w^T x + b$ (for some $w \in \mathbb{R}^d$, $b \in \mathbb{R}$), while for binary classification ($\mathcal{Y} \in \{-1, +1\}$), linear models take the form

$f(x) = \text{sign}(w^T x + b)$, where $\text{sign}(z)$ is equal to $+1$ for $z \geq 0$, and -1 otherwise. Although the class of linear models might seem overly simplistic, it is quite important. One observation is that linear models can be made very powerful if the feature representation for x is sufficiently expressive.[1] However, for many problems, there are no obvious feature representations on top of which a linear function would perform well. For these problems, we must turn to *non-linear* methods.

A wide-variety of non-linear methods have been proposed over the years (e.g., decision trees, nearest neighbor methods, etc.). In this thesis, we will focus on two important and powerful families of models: kernel models, and deep neural networks (DNNs). A kernel model is one which makes predictions on a point x by comparing it with the points in the training set. It does so using expressions of the form $\sum_{i=1}^{N} \alpha_i k(x_i, x)$, where the *kernel* function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ can be thought of as a similarity measure between two points in $\mathcal{X}$. For example, for regression and binary classification, the families of functions considered are:

$$\begin{aligned} \mathcal{F}_{reg} &= \{f \mid f(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x) + b,\ \alpha_i \in \mathbb{R}\}, \\ \mathcal{F}_{class} &= \{f \mid f(x) = \text{sign}\Big(\sum_{i=1}^{N} \alpha_i k(x_i, x) + b\Big),\ \alpha_i \in \mathbb{R}\}. \end{aligned}$$

The functions in $\mathcal{F}_{reg}$ and $\mathcal{F}_{class}$ can be understood as functions in which all the training points x_i "vote" on what the label should be for a point x, and these votes are weighted by the similarity between x and x_i. The set of kernel functions k which are generally considered are those that correspond to a dot-product between points in some Hilbert space $\mathcal{H}$, which could be infinite dimensional. Specifically, $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$, where $\phi : \mathcal{X} \rightarrow \mathcal{H}$ maps a point x into the feature space $\mathcal{H}$. As a result, we can understand kernel methods as *linear* methods in $\mathcal{H}$. In particular, for

[1]For example, consider learning a binary classifier over a training set of points (x_i, y_i) where $y_i = 1$ if $|x_i| \geq 1$, and $y_i = -1$ otherwise. If we use $x'_i = [x_i, x_i^2] \in \mathbb{R}^2$ as the feature representation for the i^{th} training point, a linear model $y = \text{sign}(w^T x' + b) = \text{sign}(w_1 x + w_2 x^2 + b)$ would be able to perfectly model this relationship, using $w_1 = 0$, $w_2 = 1$, and $b = -1$.

$f \in \mathcal{F}_{reg}$:

$$\begin{aligned} f(x) &= \sum_{i=1}^{N} \alpha_i k(x_i, x) + b \\ &= \langle \sum_{i=1}^{N} \alpha_i \phi(x_i), \phi(x) \rangle_{\mathcal{H}} + b \\ &= \langle w, \phi(x) \rangle_{\mathcal{H}} + b, \text{ for } w = \sum_{i=1}^{N} \alpha_i \phi(x_i) \in \mathcal{H}. \end{aligned}$$

Deep neural networks, on the other hand, compute y through a combination of linear and non-linear transformations of x. One common approach is to transform x sequentially, alternating between linear and non-linear transformations. This can be expressed formally by defining the following class of neural network functions:

$$\mathcal{F} = \{f \mid f(x) = \sigma_R(W_R \cdot \sigma_{R-1}(...W_2 \cdot \sigma_1(W_1 \cdot x + b_1) + b_2...) + b_R)\},$$

where $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $b_i \in \mathbb{R}^{d_i}$, $R \in \mathbb{N}$, and the $\sigma_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ functions are called activation functions, and typically perform an element-wise non-linear transformation of their input. For example, the sigmoid activation function computes $\frac{exp(x)}{1+exp(x)}$, and the rectified linear unit (ReLU) activation function computes $max(0, x)$.

Classic results show that both kernel methods and DNNs are "universal approximators," meaning that they can approximate any real-valued continuous function with bounded support to an arbitrary degree of precision [Cybenko, 1989; Hornik *et al.*, 1989; Micchelli *et al.*, 2006]. Thus, some important questions are: Which class of methods performs better on real-world tasks? Which is more efficient, in terms of training time, test time, and in terms of memory requirements? Are there learning algorithms for each of these model families which are guaranteed to return the optimal $f^* \in \mathcal{F}$?

Training a kernel model corresponds to solving a convex optimization problem, and thus there exist techniques which find the optimal $f^* \in \mathcal{F}$. Unfortunately, these methods typically do not scale well to large datasets. In particular, with data sets of size N, the $\Theta(N^2)$ size of the matrix K of pairwise kernel values ($K_{ij} = k(x_i, x_j)$) makes training prohibitively slow, while the typical $\Theta(N)$ size of the resulting models [Steinwart, 2004] makes their deployment impractical. Thus, kernel methods are typically not applied to very large-scale problems, with millions of training points.

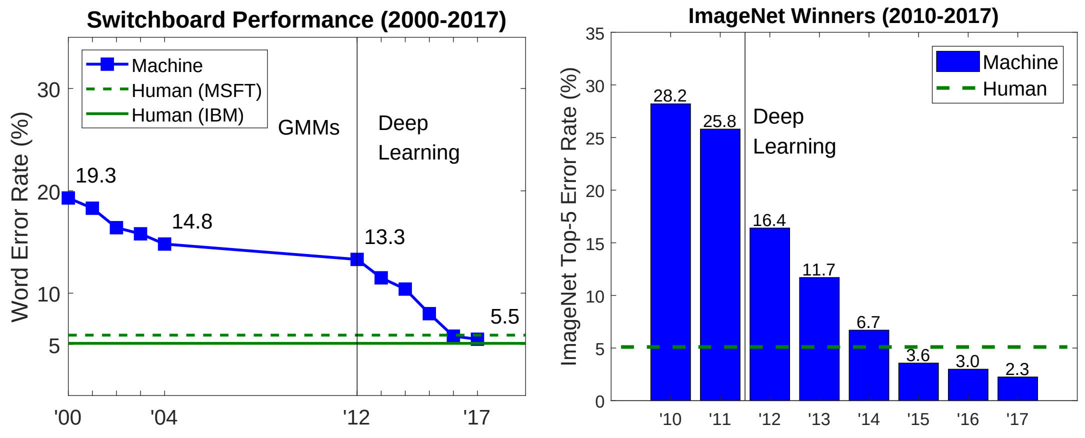

Figure 1.1: Impact of deep learning methods on state of the art performance in speech recognition and computer vision.

In contrast, DNNs are able to scale gracefully to very large datasets. They are generally trained using stochastic gradient methods, meaning that at each iteration of the algorithm, the parameters are updated using an unbiased estimate of the full gradient, which is obtained by computing the objective function on a random sample of training points. Unfortunately, the training objective for DNNs is non-convex, and thus it is generally impossible to guarantee that the model returned by the training algorithm is optimal. Nonetheless, in recent years, deep learning techniques have significantly advanced state of the art performance in various domains, including automatic speech recognition (ASR) [Seide *et al.*, 2011a; Hinton *et al.*, 2012; Mohamed *et al.*, 2012; Saon *et al.*, 2017; Xiong *et al.*, 2017], computer vision [Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014; He *et al.*, 2016], and natural language processing (NLP) [Mikolov *et al.*, 2013; Sutskever *et al.*, 2014; Andor *et al.*, 2016]. In Figure 1.1, we show the impact of deep learning methods on state of the art performance in speech recognition and computer vision. As you can see, in the past 5 years or so, deep learning methods have achieved impressive performance gains in both of these settings.[2]

The primary focus of this thesis is answering the following question:

Can kernel methods be scaled to compete with DNNs in
the large-scale settings in which DNNs currently dominate?

[2]Importantly, the Switchboard task is a relatively easy one; it consists of clear, unaccented speech between strangers, and all but four of the speakers in the test set are present in the training data.

In particular, we will focus on the *acoustic modeling* problem in speech recognition, which is the problem of modeling the pronunciation of the basic phonetic units of speech. Specifically, for a given frame of audio (typically corresponding to 25 ms), we must model the probability that the frame corresponds to a specific meaningful unit of speech. In the simplest setting, the set of units considered are called *phonemes*, which are the smallest units of sound which distinguish one word from another in a given language (for example, '/b/' and '/p/' correspond to different phonemes, because they distinguish the words "bark" and "park" from one another).[3] However, because the pronunciation of a phoneme is very affected by the phonemes that come before and after it, it is very beneficial to model phonemes *in context*. In order to address the very large number of context-dependent phoneme states (S^c if there are S phonemes, and a context window of size c is used), these states are clustered using decision trees [Hwang *et al.*, 1993; Young *et al.*, 1994]. These clustered states are called *senones*, and there are typically thousands of them, presenting a scalability challenge for training acoustic models.

Deep learning techniques have significantly advanced the state of the art in acoustic modeling, by modeling the probability $p(y|x)$ that an acoustic frame x (represented as a vector in $\mathbb{R}^d$ of acoustic features) corresponds to a senone y. In this thesis, we will scale kernel methods to this problem using *approximation* techniques, which help bypass the computational expense of solving the kernel method exactly. Much recent effort has been devoted to the development of approximations to kernel methods, primarily via the Nyström approximation [Williams and Seeger, 2001] and via random feature expansion (e.g., [Rahimi and Recht, 2007; Kar and Karnick, 2012]). These methods yield explicit feature representations on which linear learning methods can provide good approximations to the original non-linear kernel method. Specifically, they provide ways of generating representations $z(x) \in \mathbb{R}^D$ such that $\langle z(x), z(y) \rangle \approx k(x, y)$. By reducing the time and memory requirements to being *linear* in the size of the training set, these methods unlock the potential of applying kernel methods to truly large-scale tasks. However, there have been very few published attempts applying these methods to the challenging large-scale tasks on which deep learning techniques have truly shined (see Related Work section for discussion).

[3]Note that we distinguish phonemes from letters by using the '/' notation; 'b' is a letter, while '/b/' corresponds to the sound one makes when pronouncing the letter 'b.' Note that some letters can be pronounced multiple ways, making this an important distinction.

The primary contribution of this thesis is to demonstrate that kernel approximation methods can effectively compete with fully-connected feedforward neural networks on the acoustic modeling task. More specifically:

- We benchmark the performance of randomized kernel features relative to fully-connected DNNs on the acoustic modeling problem. Specifically, we use random Fourier features [Rahimi and Recht, 2007], and report results on four datasets.[4]

- We propose a novel feature selection method which can significantly improve the performance of a kernel model trained on a fixed number of random Fourier features. We show that using this technique, kernel methods effectively match the performance of feedforward DNNs across the four datasets.

- We perform an in-depth analysis comparing the performance of random Fourier features with the Nyström method for kernel approximation, in the large-scale setting. We compare these representations in terms of their kernel approximation error, their memory requirements, and their performance when used for learning.

This contribution is important for both practical and theoretical reasons. From a practical perspective, it suggests that randomized features can be competitive with deep learning methods on large-scale tasks. From a theoretical perspective, it adds to our understanding of DNNs and non-linear classification. There is a large open question of why DNNs work, which is being actively investigated from various directions, including optimization [Dauphin *et al.*, 2014; Choromanska *et al.*, 2015; Anandkumar and Ge, 2016; Agarwal *et al.*, 2017; Xie *et al.*, 2017; Pennington and Bahri, 2017], representational power and efficiency [Cybenko, 1989; Hornik *et al.*, 1989; Bengio and Lecun, 2007; Bianchini and Scarselli, 2014; Montúfar *et al.*, 2014; Ba and Caruana, 2014], and generalization performance [Bartlett, 1996; Neyshabur *et al.*, 2015; Zhang *et al.*, 2017; Arpit *et al.*, 2017]. The fact that a shallow architecture with random features can match DNNs on a task this large and challenging gives an important new perspective.

This thesis is organized as follows: Chapter 2 provides background on kernel methods, kernel approximation methods, deep neural networks, speech recognition, and acoustic modeling. We

[4]We use the IARPA Babel Program Cantonese (IARPA-babel101-v0.4c) and Bengali (IARPA-babel103b-v0.4b) limited language packs, a 50-hour subset of Broadcast News (BN-50) [Kingsbury, 2009], and TIMIT [Garofolo *et al.*, 1993].

review related work in Chapter 3. In Chapter 4, we present our work benchmarking the performance of random Fourier features relative to DNNs on four speech datasets. In Chapter 5 we present our feature selection algorithm, along with extensive experimental results using this method. In Chapter 6 we present our work comparing the Nyström method with random Fourier features. Lastly, we present our conclusions, and discuss directions for future work, in Chapter 7.

The work presented in Chapters 4 and 5 is a much extended version of the paper titled "Compact Kernel Models for Acoustic Modeling via Random Feature Selection" [May *et al.*, 2016]. These chapters also extend joint work with Lu *et al.* titled "A Comparison Between Deep Neural Nets and Kernel Acoustic Models for Speech Recognition" [Lu *et al.*, 2016]. This extended work has been posted publicly as a pre-print [May *et al.*, 2017], which is the primary document from which these chapters are adapted.

Chapter 2

Preliminaries

In this chapter, we provide background on kernel methods, kernel approximation, deep neural networks, and speech recognition. We begin by discussing the notation which we will use throughout this thesis. As further background, we include a number of important mathematical definitions in Appendix A (metric spaces, Hilbert spaces, Cauchy sequences, positive definite functions/matrices, etc.).

2.1 Notation

We will use the following notation throughout the thesis:

- $\mathbb{R}$ will denote the real numbers, $\mathbb{C}$ the complex numbers, and $\mathbb{N}$ the natural numbers.
- $[n]$ will denote the set $\{1, 2, \ldots, n\}$. If n is infinite, then $[n] = \mathbb{N}$.
- k will denote thousands, and M will denote millions (e.g., $100k$ will denote 100 thousand, and $2M$ will denote 2 million).
- We will use lower case letters to denote vectors (e.g., x), and we will use x_i to indicate the i^{th} element of the vector $x \in \mathbb{R}^d$. By default, vectors will be assumed to be column vectors.
- We will use capital letters to denote matrices (e.g., A), and use A_{ij} to indicate the element in the i^{th} row and j^{th} column of A.
- $[x_1, \ldots, x_n]$ will denote the $d \times n$ matrix with vector $x_i \in \mathbb{R}^d$ as the i^{th} column.

- $C = A \circ B$ will denote the Hadamard product between two matrices A and B, also known as the element-wise product ($C_{ij} = A_{ij}B_{ij}$).
- x^T will denote the transpose of a vector x, and A^T will denote the transpose of a matrix A.
- $\|A\|_F$ will denote the Frobenius norm of a matrix A, and $\|A\|_2$ will denote its spectral norm.
- 0_d will denote the d-dimensional zero vector, and **0** will denote the zero element in a vector space $\mathcal{X}$.
- $\mathbb{1}_{N,N}$ will denote the $N \times N$ identity matrix.
- $\mathcal{X}$ will denote the space of inputs, and $\mathcal{Y}$ will denote the space of outputs (typically equal to $\mathbb{R}$ or $[c]$ for some $c \in \mathbb{N}$).
- $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ will denote a kernel function, and $K \in \mathbb{R}^{N \times N}$ will denote the kernel matrix for a specific training set $\{x_1, \dots, x_N\} \subset \mathcal{X}$, with $K_{ij} = k(x_i, x_j)$.
- $\mathcal{H}$ will denote the feature space associated with a kernel function k, or more formally, its Reproducing Kernel Hilbert Space (see Section 2.4 for the definition).
- $\langle x, x' \rangle_{\mathcal{H}}$ will denote the inner-product in a space $\mathcal{H}$ between x and x'. If $\mathcal{H}$ is not specified, and $x, x' \in \mathbb{R}^D$, we can assume the standard dot-product $\langle x, x' \rangle = \sum_{i=1}^{D} x_i x_i'$ is used. In this case, we will often write this dot product as $x^T x'$. If $x, x' \in \mathbb{C}^D$, we use the standard complex dot-product: $\langle x, x' \rangle = \sum_{i=1}^{D} x_i \overline{x_i'}$, where $\overline{a}$ denotes the complex conjugate of any $a \in \mathbb{C}$.
- $\|x\|_{\mathcal{H}} = \sqrt{\langle x, x \rangle_{\mathcal{H}}}$ will denote the norm of a vector in a Hilbert space $\mathcal{H}$. If $\mathcal{H}$ is not specified, and $x \in \mathbb{R}^D$, we can use $\|x\|_1 = \sum_i |x_i|$ to denote the ℓ_1 norm of x, and $\|x\|_2 = \sqrt{\sum_i x_i^2}$ to denote the ℓ_2 norm of x, also known as the Euclidean norm.

2.2 Kernel methods

Kernel methods, broadly speaking, are a set of machine learning methods which learn to make predictions on unseen datapoints by considering their similarity to the points in the training set. In general, we define the similarity between two points in $\mathcal{X}$ through a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

Kernel models make predictions on unseen points x by making use of expressions of the form $h(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x) + b$; for regression, $h(x)$ is used directly as the prediction of the model $f(x)$, while for binary classification, $f(x) = \text{sign}(h(x))$. Thus, each training points x_i can influence the prediction of the model f on a point x, where this influence is weighted by the similarity between x_i and x. If the kernel function $k(x, x')$ corresponds to a dot-product $\langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$ in some feature space $\mathcal{H}$, then we can additionally interpret kernel models as linear models in this space:

$$\begin{aligned} h(x) &= \sum_{i=1}^{N} \alpha_i k(x_i, x) + b \\ &= \left\langle \sum_{i=1}^{N} \alpha_i \phi(x_i), \phi(x) \right\rangle_{\mathcal{H}} + b \\ &= \langle w, \phi(x) \rangle_{\mathcal{H}} + b, \text{ for } w = \sum_{i=1}^{N} \alpha_i \phi(x_i). \end{aligned}$$

Here, $\phi : \mathcal{X} \to \mathcal{H}$ is a feature map which sends a point in $\mathcal{X}$ to its corresponding point in $\mathcal{H}$. Importantly, for all positive definite kernels k, such a map exists.[1] Thus, kernel methods can be seen as a set of machine learning techniques which either explicitly (with ϕ) or implicitly (with k) map data from the input space $\mathcal{X}$ to some Hilbert space $\mathcal{H}$, in which a linear model is learned. We will now discuss the two primary ways of understanding kernel methods in more detail, corresponding to whether this mapping to $\mathcal{H}$ is explicit or implicit. In this discussion, we will use d to denote the dimension of $\mathcal{X}$, D to denote the dimension of $\mathcal{H}$ (which could be infinite), and N to denote the number of training points (x_i, y_i).

2.2.1 Primal formulation

In the first formulation (which we will call the "primal"), we consider an *explicit* mapping $\phi : \mathcal{X} \to \mathcal{H}$. We then learn a linear model directly on these representations:

$$w^* = \operatorname*{arg\,min}_{w \in \mathcal{H}, b \in \mathbb{R}} \sum_{i=1}^{N} L(\langle w, \phi(x_i) \rangle_{\mathcal{H}} + b, y_i) + R(w).$$

[1]In fact, there are many such maps, which are all equivalent (i.e., isometrically isomorphic), yet take very different forms. One such map is the mapping from x to the function $k(x, \cdot)$ in the Reproducing Kernel Hilbert Space for k (see Section 2.4). Another is given by Mercer's Theorem, which shows that such a map exists, where the dimension of the space $\mathcal{H}$ is countably infinite [mer, 1909]. Yet another is given by Bochner's Theorem, in the case of *shift-invariant* kernels (see Section 2.3.1).

Here, $R(w)$ is a regularization term which encourages simple models (typically, by penalizing the norm of w; for example, $R(w) = \frac{\lambda}{2}\|w\|_{\mathcal{H}}^2$), in order to improve generalization performance of w^*. Additionally, $L : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ is a generic loss function, which penalizes the model based on some function of $\langle w, \phi(x_i)\rangle_{\mathcal{H}} + b$ and the true label y_i. The set $\mathcal{Y}$ corresponds to $\mathbb{R}$ for regression, and $\{-1, +1\}$ for binary classification. For regression, the quadratic loss $L(z_i, y_i) = \frac{1}{2}(z_i - y_i)^2$ is typical, while for binary classification, the cross-entropy loss function $L(z_i, y_i) = \log\big((1 + \exp(-y_i z_i)\big)$ is common.

As an example, consider the following feature map $\phi : \mathbb{R}^2 \to \mathbb{R}^5$, applied to a point $x = [x_1, x_2] \in \mathbb{R}^2$:

$$\phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1). \tag{2.1}$$

A linear model trained on top of this feature representation would in effect be finding the optimal quadratic function for the given task. This demonstrates how learning a linear model over features generated through a non-linear map $\phi : \mathcal{X} \to \mathcal{H}$ corresponds to learning a non-linear model in the original space $\mathcal{X}$. This can imbue the model with a lot more power.

For a given map ϕ, we can define the corresponding kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ as $k(x, x') = \langle \phi(x), \phi(x')\rangle_{\mathcal{H}}$. For example, in the case of the quadratic map ϕ shown in Equation 2.1, the corresponding kernel function is

$$\begin{aligned} k(x, x') &= x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1x_2x_1'x_2' + 2x_1x_1' + 2x_1x_1' + 1 \\ &= (x^T x' + 1)^2 \end{aligned} \tag{2.2}$$

Importantly, defining $k(x, x')$ as a dot-product in some space $\mathcal{H}$ implies that k is a positive definite function, meaning that for any $c_1, \ldots, c_N \in \mathbb{R}$, and any $x_1, \ldots, x_N \in \mathcal{X}$, $\sum_{i,j=1}^N c_i c_j k(x_i, x_j) \geq 0$. This is easy to see, because $\sum_{i,j=1}^N c_i c_j k(x_i, x_j) = \langle \sum_{i=1}^N c_i \phi(x_i), \sum_{i=1}^N c_i \phi(x_i)\rangle_{\mathcal{H}} \geq 0$.

It is important to note that the computational expense of performing one epoch of stochastic gradient descent on the primal optimization problem discussed above is $O(ND)$ (Recall, N is the size of the training set, and D is the dimension of $\mathcal{H}$. Here we do not include the cost of computing $\phi(x)$.). This is fast as long as neither N or D is too large. Unfortunately, for a wide variety of of kernels, D is extremely large, or even infinite. In order to address this computational hurdle, we can instead work with the dual formulation of the optimization problem; this is discussed in the following section.

2.2.2 Dual formulation

In the second formulation (which we will call the "dual"), instead of defining the kernel function k in terms of an explicit map ϕ, we instead define the kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ directly. We require that k be a positive definite function. Generally, we can think of k as a similarity function, which will assign a high score to pairs of points that are similar (e.g., close in $\mathbb{R}^d$), and a low score to points that are different. For example, we list some common kernel functions in Table 2.1.

$$\begin{array}{rrcl}
\text{Gaussian kernel:} & k(x,x') & = & \exp\left(-\frac{\|x-x'\|_2^2}{2\sigma^2} \right) \\
\text{Laplacian kernel:} & k(x,x') & = & \exp\left(-\lambda \|x-x'\|_1 \right) \\
\text{Polynomial kernel (degree } r\text{):} & k(x,x') & = & (x^T x' + 1)^r
\end{array}$$

Table 2.1: A few example kernel functions.

As you can see, the kernel we defined in Equations 2.1 and 2.2 is an example of a degree 2 polynomial kernel. Notice that for all of these kernels, the kernel function can be computed in $O(d)$, where d is the dimension of $\mathcal{X}$.

We saw above how to perform optimization in the primal view of the problem. But how do we learn models using the dual view? For a large set of primal optimization problems of the form

$$w^* = \operatorname*{arg\,min}_{w \in \mathcal{H}, b \in \mathbb{R}} \sum_{i=1}^{N} L(\langle w, \phi(x_i) \rangle_{\mathcal{H}} + b, y_i) + R(w),$$

it is possible to reformulate the problem as an equivalent *dual* optimization problem, which only requires knowledge of the kernel matrix $K \in \mathbb{R}^{N \times N}$. For example, the SVM primal problem is:

$$\min_{w \in \mathcal{H}, b \in \mathbb{R}} C \sum_{i=1}^{N} \max\Big(0, 1 - y_i\big(\langle w, \phi(x_i) \rangle_{\mathcal{H}} + b\big)\Big) + \|w\|_{\mathcal{H}}^2.$$

The corresponding dual problem is:

$$\max_{\alpha_i \geq 0} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_k k(x_i, x_j)$$

$$\text{subject to } \alpha_i \in [0, C] \ \forall i, \text{ and} \sum_{i=1}^{N} \alpha_i y_i = 0.$$

After the optimal dual parameters α_i^* are found by solving the above optimization problem, the optimal bias term b^* can be computed by taking any vector x_j for which $0 < \alpha_j^* < C$ (these are called "support vectors"), and solving for b^* in the equation $1 = y_j(\sum_{i=1}^N \alpha_i^* y_i k(x_i, x_j) + b^*)$. Then, the model corresponding to these parameters is $f(x) = \text{sign}(\sum_{i=1}^N \alpha_i^* y_i k(x_i, x) + b^*)$.

Importantly, the dual optimization problem interacts with the datapoints x_i exclusively through the kernel function k. Importantly, both the primal and the dual optimization problems for kernel methods are convex. In the case where the kernel functions between two points can be computed quickly (e.g., in $O(d)$), and the dimension D of the primal space $\mathcal{H}$ is much bigger than N, it is more efficient to solve the dual (time at least quadratic in N) than to solve the primal (time at least linear in ND). This is called the "kernel trick," and it allows us to find the optimal linear classifier in $\mathcal{H}$, even if it is an infinite dimensional space. However, if N is very large, the dual formulation will be too expensive to solve as well, as it requires performing an optimization over the full $N \times N$ kernel matrix; simply computing the kernel matrix takes time (N^2d) (assuming kernel evaluations take $O(d)$). As an example, for a dataset with a million training points, the kernel matrix consumes four terabytes of memory if stored as single precision floats. This leads to the following question: What can we do in the case where both N and D are extremely large? For example, what if D is infinite, and N is in the millions? In this case, one can use kernel approximation methods, which we now discuss.

2.3 Kernel approximation

In the section, we discuss two important ways of doing kernel approximation: random Fourier features [Rahimi and Recht, 2007], and the Nyström method [Williams and Seeger, 2001]. These methods share the following goal: to construct low-dimensional representations $z(x) \in \mathbb{R}^D$ such that $z(x)^T z(x') \approx k(x, x')$. We can then simply train a linear model on top of these representations in order to attain an approximate solution to the exact kernel optimization problem. Training would thus require $O(ND)$ time per epoch,[2] which is much better than solving the dual problem when $D << N$. In particular, for a fixed D this runtime only grows *linearly* in N, making this appealing in the large-scale setting. We now provide overviews for how $z(x)$ is constructed, using either

[2]This runtime assumes that $z(x)$ can be computed in $O(D)$ for every x, which is generally not the case. However, even if computing $z(x)$ is more expensive than this, you can incur this as a one-time cost at the beginning of training, and store the representations $z(x)$ to disk (or in memory). If we assume computing $z(x)$ is $O(Dd)$, which is common, and these features are computed on the fly during training, then the total runtime for an epoch of training becomes $O(NDd)$.

random Fourier features, or the Nyström method.

2.3.1 Random Fourier features (RFF)

For the random Fourier features method [Rahimi and Recht, 2007], the theorem which allows for the construction of the representation $z(x)$ is called Bochner's Theorem. This is a classical result in harmonic analysis, and it allows us to approximate any positive-definite *shift-invariant* kernel k with *finite*-dimensional features. A kernel $k(x, x')$ is *shift-invariant* if and only if $k(x, x') = \hat{k}(x - x')$ for some function $\hat{k}\colon \mathbb{R}^d \to \mathbb{R}$. We now present Bochner's Theorem:

Theorem 1. *(Bochner's Theorem, adapted from [Rahimi and Recht, 2007]): A continuous shift-invariant kernel $k(x, x') = \hat{k}(x - x')$ on $\mathbb{R}^d$ is positive-definite if and only if $\hat{k}$ is the Fourier transform of a non-negative measure $\mu(\omega)$.*

Thus, for any positive-definite shift-invariant kernel $\hat{k}(\delta)$, we have that

$$\hat{k}(\delta) = \int_{\mathbb{R}^d} \mu(\omega) e^{-j\omega^T\delta}\, d\omega, \tag{2.3}$$

where

$$\mu(\omega) = (2\pi)^{-d} \int_{\mathbb{R}^d} \hat{k}(\delta) e^{j\omega^T\delta}\, d\delta \tag{2.4}$$

is the inverse Fourier transform[3] of $\hat{k}(\delta)$, and where $j = \sqrt{-1}$. By Bochner's theorem, $\mu(\omega)$ is a non-negative measure. As a result, if we let $Z = \int_{\mathbb{R}^d} \mu(\omega) d\omega$, then $p(\omega) = \frac{1}{Z}\mu(\omega)$ is a proper probability distribution, and we get that

$$\frac{1}{Z}\hat{k}(\delta) = \int_{\mathbb{R}^d} p(\omega) e^{-j\omega^T\delta}\, d\omega.$$

For simplicity, we will assume going forward that $\hat{k}$ is properly-scaled, meaning that $Z = 1$. Now, the above equation allows us to rewrite this integral as an expectation:

$$\hat{k}(\delta) = \hat{k}(x - x') = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^T(x-x')}\, d\omega = \mathbb{E}_\omega\left[e^{j\omega^T x} e^{-j\omega^T x'}\right]. \tag{2.5}$$

This can be further simplified as

$$\hat{k}(x - x') = \mathbb{E}_{\omega,b}\left[\sqrt{2}\cos(\omega^T x + b) \cdot \sqrt{2}\cos(\omega^T x' + b)\right],$$

[3]There are various ways of defining the Fourier transform and its inverse. We use the convention specified in Equations (2.3) and (2.4), which is consistent with [Rahimi and Recht, 2007].

Kernel name	$\boldsymbol{k(x,x')}$	$\boldsymbol{p(\omega)}$	**Density name**
Gaussian	$e^{-\lVert x-x'\rVert_2^2/2\sigma^2}$	$(2\pi(1/\sigma^2))^{-d/2}e^{-\frac{\lVert\omega\rVert_2^2}{2(1/\sigma)^2}}$	Normal$(0_d, \frac{1}{\sigma^2}\mathbb{1}_{d,d})$
Laplacian	$e^{-\lambda\lVert x-x'\rVert_1}$	$\prod_{i=1}^d \frac{1}{\lambda\pi(1+(\omega_i/\lambda)^2)}$	Cauchy$(0_d, \lambda)$

Table 2.2: Gaussian and Laplacian Kernels, together with their sampling distributions $p(\omega)$

where ω is drawn from $p(\omega)$, and b is drawn uniformly from $[0, 2\pi]$. See Appendix B for details on why this specific functional form is correct.[4] In Table 2.2, we list two popular (properly-scaled) positive-definite kernels with their respective inverse Fourier transforms $p(\omega)$.

This motivates a sampling-based approach for approximating the kernel function. Concretely, we draw $\{\omega_1, \omega_2, \ldots, \omega_D\}$ independently from the distribution $p(\omega)$, and $\{b_1, b_2, \ldots\ b_D\}$ independently from the uniform distribution on $[0, 2\pi]$, and then use these parameters to approximate the kernel, as follows:

$$\begin{aligned} k(x,y) &\approx \frac{1}{D}\sum_{i=1}^{D}\sqrt{2}\cos(\omega_i^T x + b_i)\cdot\sqrt{2}\cos(\omega_i^T y + b_i) \\ &= z(x)^T z(x'), \end{aligned}$$

where $z_i(x) = \sqrt{\frac{2}{D}}\cos(\omega_i^T x + b_i)$ is the i^{th} element of the D-dimensional random vector $z(x)$. This gives us the explicit (random) mapping $z : \mathbb{R}^d \to \mathbb{R}^D$ proposed by the random Fourier features method. It has the very nice property that for all indices $i \in [D]$, $\mathbb{E}_{\omega,b}\left[z_i(x)z_i(x')\right] = \frac{1}{D}k(x,x')$, and thus that $\mathbb{E}_{\omega,b}\left[z(x)^T z(x')\right] = k(x,x')$.

We can bound the probability that $z(x)^T z(x')$ is more than ϵ away from $k(x,x')$ as follows. First, define $X_i = \sqrt{\frac{2}{D}}\cos(\omega_i^T x + b_i)\sqrt{\frac{2}{D}}\cos(\omega_i^T x' + b_i)$ to be a random variable corresponding to random draws ω_i, b_i, and let $X = \sum_{i=1}^D X_i$. Noticing that $X_i \in [-\frac{2}{D}, \frac{2}{D}]$, that $\mathbb{E}_{\omega,b}\left[X\right] = k(x,x')$, and that $X = z(x)^T z(x')$ for the random representations $z(x), z(x')$, we can directly apply Hoeffding's inequality on X to prove the desired bound:

$$\mathbb{P}_{\omega,b}\left[|z(x)^T z(x') - k(x,x')| \geq \epsilon\right] \leq 2\exp\left(\frac{-D\epsilon^2}{8}\right). \tag{2.6}$$

[4]Another important thing to notice is that the integral in Equation 2.5 immediately gives us an explicit mapping ϕ from $\mathcal{X} = \mathbb{R}^d$ to the space of complex square-integrable functions $L_2(\mathbb{R}^d, p)$, where $\phi(x)$ is the function $f_x : \mathbb{R}^d \to \mathbb{C}$ defined as $f_x(\omega) = e^{\omega^T x}$. Thus, $\langle\phi(x), \phi(x')\rangle_{L_2} = \int_{\mathbb{R}^d} f_x(\omega)\overline{f_{x'}(\omega)}p(\omega)d\omega = \int_{\mathbb{R}^d} p(\omega)e^{j\omega^T(x-x')}d\omega = k(x,x')$, as desired. See Section D.4.1 in Appendix D for more details on how this Hilbert space is defined.

In the original work by Rahimi and Recht, the authors prove a much stronger result, bounding the probability that $z(x)^T z(x)$ is within ϵ of $k(x, x')$ for *all* pairs $x, x' \in \mathcal{X}$ *simultaneously* [2007]. Specifically, they show that if $D = \tilde{\Omega}(\frac{d}{\epsilon^2})$, then with high probability $z(x)^T z(x')$ will be within ϵ of $k(x, x')$ for *all* x, x' in some compact subset $\mathcal{M} \in \mathbb{R}^d$ of bounded diameter.[5] See Claim 1 of [Rahimi and Recht, 2007] for the more precise statement and proof of this result.

In their follow-up work, Rahimi and Recht prove a generalization bound for models learned using these random features [2008]. They show that with high-probability, the excess risk[6] assumed from using this approximation, relative to using the "oracle" kernel model (the exact kernel model with the lowest risk), is bounded by $O(\frac{1}{\sqrt{N}} + \frac{1}{\sqrt{D}})$ (see the main result of [Rahimi and Recht, 2008] for more details). Given that the generalization error of a model trained using exact kernel methods is known to be within $O(\frac{1}{\sqrt{N}})$ of the oracle model [Bartlett *et al.*, 2002], this implies that in the worst case, $D = \Theta(N)$ random features may be required in order for the approximated model to achieve generalization performance comparable to the exact kernel model. Empirically, however, fewer than $\Theta(N)$ features are typically needed in order to attain strong performance, as we will see in Chapters 4, 5, and 6, and as has been seen in existing work (e.g., [Yu *et al.*, 2015]).

2.3.2 Nyström method

Like random Fourier features, the Nyström method constructs a feature representation $z(x) \in \mathbb{R}^D$ such that $z(x)^T z(x') \approx k(x, x')$. However, the Nyström method takes an entirely different approach in order to construct this feature map. Instead of finding a way to approximate $k(x, x')$ well for any pair $x, x' \in \mathcal{X}$, the Nyström method approaches this problem from the perspective of low-rank matrix decomposition. Specifically, the Nyström method attempts to approximate the full kernel matrix K well (e.g., in terms of Frobenius or spectral norm), using a low-rank decomposition $\hat{K} = Z^T Z$, where the i^{th} column of Z will correspond to $z(x_i)$. Note that one can find the optimal such $Z \in \mathbb{R}^{D \times N}$ minimizing both $\|K - Z^T Z\|_F$ as well as $\|K - Z^T Z\|_2$ by taking the singular value decomposition (SVD) $K = U \Lambda U^T$ of the kernel matrix K (with the singular values sorted from largest to smallest along the diagonal of Λ); then, the optimal $Z^T = U_D \Lambda_D^{1/2}$, where Λ_D denotes the $D \times D$ diagonal matrix of the D largest singular values, and U_D denotes the first D

[5]We are using the $\tilde{\Omega}$ notation to hide logarithmic factors.

[6]The "risk" of a model is defined as its expected loss on unseen data.

columns of U (which correspond to the singular vectors of the largest singular values). There are two problems with this solution:

1. Computing the SVD of a $N \times N$ matrix takes $O(N^3)$, and is thus impractical for very large N, which is precisely the setting in which we are interested in using kernel approximation. Given that we can generally solve the dual kernel optimization problems in $O(N^3)$, there is no reason to prefer this method from an efficiency perspective.

2. From the above definition for $z(x)$, it is not clear how one would compute $z(x)$ for a point x which isn't in the training set.[7] This however, has an easy solution; if we let $A_D = U_D \Lambda_D^{-1/2}$, we get:

$$\begin{aligned} KA_D &= KU_D\Lambda_D^{-1/2} \\ &= (U\Lambda U^T)U_D\Lambda_D^{-1/2} \\ &= U_D\Lambda_D\Lambda_D^{-1/2} \\ &= Z^T. \end{aligned}$$

   Thus, if we let $Z^T = KA_D$, we can take $z(x) = A_D^T k_x = \Lambda_D^{-1/2} U_D^T k_x$, where $k_x = [k(x, x_1), \ldots, k(x, x_N)]^T$ is the vector of kernel evaluations between x and all training points x_i. Notice that this function $z(\cdot) \in span(k(x_1, \cdot), \ldots, k(x_N, \cdot)) \subset \mathcal{H}$, and thus a linear model trained on top of this representation automatically gives us a model of the form $\sum_{i=1}^N \alpha_i k(x_i, \cdot)$, which we will see in Section 2.4.1 must be the form of the optimal kernel model (by Representer Theorem). Note, however, that while this solves the problem of how to compute $z(x) \in \mathbb{R}^D$ for any point $x \in \mathcal{X}$, it does not address the computational issue, that computing A_D still requires taking the SVD of the full kernel matrix K.

The Nyström method addresses the first problem raised above, by taking inspiration from the solution to the second problem. The intuition behind the Nyström method is as follows: instead of constructing the representation $z(x)$ based on the SVD of the full kernel matrix K, consider instead the SVD of $K_{m,m}$, the kernel matrix corresponding to "landmark" points $\{\hat{x}_1, \ldots, \hat{x}_m\}$; note that normally, these landmark points are selected from the training set (e.g., uniformly at random),

[7]For a points x_i in the training set, simply let $z(x_i)$ be the i^{th} column of Z

but this need not be the case (e.g., [Zhang *et al.*, 2008]). Now, we can consider $z(x) = \hat{A}_D^T \hat{k}_x$, where $K_{m,m} = \hat{U}\hat{\Lambda}\hat{U}^T$ is the SVD of the landmark point kernel matrix, $\hat{A}_D = \hat{U}_D \hat{\Lambda}_D^{-1/2}$, and $\hat{k}_x = [k(x, \hat{x}_1), \ldots, k(x, \hat{x}_m)]^T$. Taking $Z_m = [z(\hat{x}_1), \ldots, z(\hat{x}_m)]$ as the $D \times m$ matrix with $z(\hat{x}_i)$ as the i^{th} column, gives us the optimal rank D decomposition $Z_m^T Z_m$ for $K_{m,m}$, as discussed above ($Z_m^T Z_m = K_{m,m}$ if $m = D$). However, we can take this $z(x) \in \mathbb{R}^D$ to be our representation for any point $x \in \mathcal{X}$, and this gives us a low rank decomposition $Z^T Z$ to the full kernel matrix K, where $Z = [z(x_1), \ldots, z(x_N)]$. This $z(x)$ is precisely the representation which the Nyström method constructs for the purposes of kernel approximation. Just like with random Fourier features, one can learn a linear model on top of these representations in order to approximately solve the kernel optimization problem.

One important thing to note is the cost of the Nyström method, both in terms of time and memory:

1. Time (SVD): Computing the SVD of the $m \times m$ kernel matrix takes $O(m^3)$. Note that when $D < m$, the SVD can be done faster using randomized SVD algorithms [Halko *et al.*, 2011], which would take $O(m^2 D)$ instead of $O(m^3)$.

2. Time (training): During training, we must compute the representation $z(x) = \hat{A}_D^T \hat{k}_x$ for every training point x. Assuming computing $k(x, x')$ takes time $O(d)$ for $x, x' \in \mathbb{R}^d$, this takes time $O(Nmd + NmD)$. This is because computing $\hat{k}_{x_i}\ \forall i$ takes time $O(Nmd)$, while computing the matrix multiplication $\hat{A}_D^T \hat{k}_{x_i}$ takes $O(NmD)$ ($O(mD)$ cost for each x_i). Note that this is a lot more expensive than the cost of random Fourier features, which take time $O(NDd) \leq O(NmD)$ to compute; importantly, for RFF this can be further reduced to $O(ND\log(d))$ using structured matrix multiplications [Le *et al.*, 2013; Yu *et al.*, 2015].

3. Memory: Storing the m landmark points requires storing md floats, while storing the $\hat{A}_D$ matrix requires mD storage. This brings the total memory requirement to $O(md{+}mD)$. Note that this is substantially more expensive than the storage costs of random Fourier features, which simply require $O(Dd) \leq O(md)$, and which can be further reduced to $O(D)$ [Le *et al.*, 2013; Yu *et al.*, 2015].

In Table 2.3, we summarize the computational costs of Nyström vs. random Fourier features, discussed above.

Method	**Time**	**Memory**
Nyström	$O(m^2D + Nmd + NmD)$	$O(md + mD)$
Random Fourier Features	$O(NDd)$	$O(Dd)$
Random Fourier Features (structured)	$O(ND\log(d))$	$O(D)$

Table 2.3: Cost of computing Nyström vs. random Fourier features (RFF), both in terms of time and memory. For RFF, we also report the costs of the more efficient implementation using structured matrices [Le *et al.*, 2013; Yu *et al.*, 2015].

There are a variety of ways of understanding the Nyström method, aside from the one explained above. They include:

1. As a projection onto a subspace.
2. As a solution to an optimization problem.
3. As a preconditioning method.
4. As a way to approximate the eigenvalues and eigenfunctions of the linear operator of the kernel function, using Monte Carlo approximation.

These are all explained in further detail in Appendix D.

2.4 Reproducing kernel Hilbert spaces (RKHS)

One way of understanding kernel methods is as optimization over a space of functions from $\mathcal{X}$ to $\mathbb{R}$. The space of functions which corresponds to a specific kernel is called its Reproducing Kernel Hilbert Space (RKHS). Given a symmetric positive definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, its corresponding RKHS is essentially defined as the set of all linear combinations of functions of the form $k(x, \cdot)$, for $x \in \mathcal{X}$. Note that here I am using the notation $k(x, \cdot)$ to correspond to the function $f_x : \mathcal{X} \to \mathbb{R}$ such that $f_x(x') = k(x, x')$.

There are several ways of formally defining an RKHS [Sejdinovic and Gretton, 2012]; in this section, however, we will focus on the Moore-Aronsajn construction of the RKHS corresponding to a positive definite kernel k [Aronszajn, 1950]. This construction has two main steps: first, we will define the "pre-RKHS" $\mathcal{H}_0$ as the set of all *finite* linear combinations of functions of the form $k(x, \cdot)$ for $x \in \mathcal{X}$. Note that this is a subset of $\mathbb{R}^{\mathcal{X}}$, the space of all functions from $\mathcal{X}$ to $\mathbb{R}$. We will then turn this into a *complete* space $\mathcal{H}$ by adding to $\mathcal{H}_0$ the set of all its limit points in $\mathbb{R}^{\mathcal{X}}$. We now go through both of these steps in detail.

We begin by defining the set $\mathcal{H}_0$, along with an inner product in this space:

$$\mathcal{H}_0 = \{f(\cdot) = \sum_{i=1}^{n} \alpha_i k(x_i, \cdot) \mid x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, n \in \mathbb{N}\}.$$

We now define the inner product between $f = \sum_{i=1}^{n} \alpha_i k(x_i, \cdot)$ and $g = \sum_{j=1}^{m} \beta_j k(\tilde{x}_j, \cdot)$ in $\mathcal{H}_0$ as:

$$\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(x_i, \tilde{x}_j).$$

Now, if we consider the function $\phi : \mathcal{X} \to \mathcal{H}_0$, defined as $\phi(x) = k(x, \cdot)$, we can see by the above definition that $\langle \phi(x), \phi(x') \rangle_{\mathcal{H}_0} = \langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}_0} = k(x, x')$. Thus, we have constructed a map ϕ to a space $\mathcal{H}_0$ in which $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}_0}$, allowing us to view optimization over an RKHS as searching for a linear model in this space.[8]

Note that the above definition of the inner product in $\mathcal{H}_0$ implies that the (squared) norm of a function $k(x, \cdot) \in \mathcal{H}_0$ is $\|k(x, \cdot)\|^2_{\mathcal{H}_0} = \langle k(x, \cdot), k(x, \cdot) \rangle_{\mathcal{H}_0} = k(x, x)$. Furthermore, using this definition we can show that the "reproducing property" holds in $\mathcal{H}_0$—namely, that for any $f = \sum_{i=1}^{n} \alpha_i k(x_i, \cdot) \in \mathcal{H}_0$, and any $x \in \mathcal{X}$, $\langle f, k(x, \cdot) \rangle = f(x)$. This can be seen easily:

$$\begin{aligned} \langle f, k(x, \cdot) \rangle &= \left\langle \sum_{i=1}^{n} \alpha_i k(x_i, \cdot), k(x, \cdot) \right\rangle_{\mathcal{H}_0} \\ &= \sum_{i=1}^{n} \alpha_i k(x_i, x) \\ &= f(x). \end{aligned}$$

If we are in the case where we have an explicit feature map $\phi : \mathcal{X} \to \mathcal{H}'$, we can view the

[8]Importantly, $\mathcal{H}_0$ is not a proper Hilbert space, though it can be extended to one, as we will discuss.

corresponding pre-RKHS $\mathcal{H}_0$ as follows:

$$\begin{aligned}
\mathcal{H}_0 &= \{f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) \mid x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, n \in \mathbb{N}\} \\
&= \{f(\cdot) = \sum_{i=1}^{n} \alpha_i \langle \phi(\cdot), \phi(x_i) \rangle_{\mathcal{H}'} \mid x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, n \in \mathbb{N}\} \\
&= \{f(\cdot) = \left\langle \phi(\cdot), \sum_{i=1}^{n} \alpha_i \phi(x_i) \right\rangle_{\mathcal{H}'} \mid x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, n \in \mathbb{N}\} \\
&= \{f_a(\cdot) = \langle \phi(\cdot), a \rangle_{\mathcal{H}'} \mid a = \sum_{i=1}^{n} \alpha_i \phi(x_i), x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, n \in \mathbb{N}\}.
\end{aligned}$$

Thus, each function in $\mathcal{H}_0$ can be associated with a unique element $a \in \mathcal{H}'$. Furthermore, we will now show that the dot-product between two elements in $\mathcal{H}_0$ corresponds to the dot-product between the corresponding points in $\mathcal{H}'$. For $f_a, f_b \in \mathcal{H}_0$, where $f_a = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i)$, $f_b = \sum_{j=1}^{m} \beta_j k(\cdot, \tilde{x}_j)$, and the corresponding points $a = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ and $b = \sum_{j=1}^{m} \beta_j \phi(\tilde{x}_j)$, we have:

$$\begin{aligned}
\langle f_a, f_b \rangle_{\mathcal{H}_0} &= \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(x_i, \tilde{x}_j) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j \langle \phi(x_i), \phi(\tilde{x}_j) \rangle_{\mathcal{H}'} \\
&= \left\langle \sum_{i=1}^{n} \alpha_i \phi(x_i), \sum_{j=1}^{m} \beta_j \phi(\tilde{x}_j) \right\rangle_{\mathcal{H}'} \\
&= \langle a, b \rangle_{\mathcal{H}'}
\end{aligned}$$

This shows that there is a very strong correspondence between the space $\mathcal{H}_0$ and the subspace of the feature space $\mathcal{H}'$ composed of all finite linear combinations of points $\phi(x)$ for $x \in \mathcal{X}$. In fact, these spaces are isometrically isomorphic, meaning that there exists a linear bijection ψ between the spaces, and this map preserves inner products ($\langle f, g \rangle_{\mathcal{H}_0} = \langle \psi(f), \psi(g) \rangle_{\mathcal{H}'}$). This map takes the expected form, mapping $f_a \in \mathcal{H}_0$ to $a \in \mathcal{H}'$.

In order to turn this pre-RKHS $\mathcal{H}_0$ into a proper Hilbert space $\mathcal{H} \supset \mathcal{H}_0$, we need to make it *complete*; this means that all Cauchy sequences in $\mathcal{H}$ must converge to points in $\mathcal{H}$. Specifically, we must add to $\mathcal{H}_0$ the points $f \in \mathbb{R}^{\mathcal{X}}$ for which there exists a Cauchy sequence $\{f_1, f_2, \ldots\} \in \mathcal{H}_0$ which converges pointwise to f. Given two such points, f and g, where f is the limit of the Cauchy-sequence $\{f_n\}$, and g is the limit of $\{g_n\}$, we define the dot product between f and g as the limit of the dot products of the sequences: $\langle f, g\rangle_{\mathcal{H}} = \lim_{n\to\infty}\langle f_n, g_n\rangle_{\mathcal{H}_0}$.

For more of the formal mathematical details behind this construction, see [Berlinet and Thomas-Agnan, 2003; Sejdinovic and Gretton, 2012].

2.4.1 Representer Theorem

One very important result regarding optimization over an RKHS is the Representer Theorem, which we present now:

Theorem 2. *(Representer Theorem, adapted from [Schölkopf* et al.*, 2001]): Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric positive definite kernel, and $\mathcal{H}$ its RKHS. Then, for any non-decreasing function $G : \mathbb{R} \to \mathbb{R}$, any loss function $L : (\mathcal{X} \times \mathcal{Y} \times \mathbb{R})^N \to \mathbb{R} \cup \{+\infty\}$, and any N labeled points $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, the optimization problem*

$$\arg\min_{f\in\mathcal{H}} G(\|f\|_{\mathcal{H}}) + L((x_1, y_1, f(x_1)), \ldots, (x_N, y_N, f(x_N)))$$

has a solution of the form $f^* = \sum_{i=1}^{N} \alpha_i k(x_i, \cdot)$. Furthermore, if G is a strictly increasing function, then any solution has this form.

Proof. Let $A = span(k(x_1, \cdot), \ldots, k(x_N, \cdot)) \subset \mathcal{H}$, and let $A^{\perp}$ be the orthogonal complement of A in $\mathcal{H}$. Thus, $\mathcal{H} = A \oplus A^{\perp}$, and any $f \in \mathcal{H}$ can be uniquely decomposed as $f = f_A + f_{A^{\perp}}$, for $f_A \in A$ and $f_{A^{\perp}} \in A^{\perp}$. It follows from the reproducing property, and the orthogonality between all points $f_{A^{\perp}} \in A^{\perp}$ with points $k(x_i, \cdot) \in A$, that for all $x_1, \ldots, x_N$,

$$\begin{aligned} f(x_i) &= \langle f, k(x_i, \cdot)\rangle \\ &= \langle f_A, k(x_i, \cdot)\rangle + \langle f_{A^{\perp}}, k(x_i, \cdot)\rangle \\ &= \langle f_A, k(x_i, \cdot)\rangle \\ &= f_A(x_i). \end{aligned}$$

Thus, $L((x_1, y_1, f(x_1)), \ldots, (x_N, y_N, f(x_N))) = L((x_1, y_1, f_A(x_1)), \ldots, (x_N, y_N, f_A(x_N)))$. Now, note that $\|f\|_{\mathcal{H}} = \sqrt{\|f_A\|^2_{\mathcal{H}} + \|f_{A^{\perp}}\|^2_{\mathcal{H}}} \geq \|f_A\|_{\mathcal{H}}$. Thus, it follows from G being non-decreasing that $G(\|f\|_{\mathcal{H}}) \geq G(\|f_A\|_{\mathcal{H}})$. It now immediately follows that for any $f = f_A + f_{A^{\perp}} \in \mathcal{H}$, with $f_A \in A$, $f_{A^{\perp}} \in A^{\perp}$, we have $G(\|f\|_{\mathcal{H}}) + L((x_1, y_1, f(x_1)), \ldots, (x_N, y_N, f(x_N))) \geq$

$G(\|f_A\|_{\mathcal{H}}) + L((x_1, y_1, f_A(x_1)), \dots, (x_N, y_N, f_A(x_N)))$, and thus f_A is at least as good as f with respect to this objective function we are minimizing. Now, if $f^* = f^*_A + f^*_{A^\perp} \in \mathcal{H}$ is a global minimizer of the objective function, it follows that $f^*_A = \sum_{i=1}^N \alpha_i k(x_i, \cdot)$ is also a global minimizer. This proves the first part of the theorem.

For the second part, we assume G is a strictly increasing function. In this case, all solutions $f^* = f^*_A + f^*_{A^\perp}$ must be of this form ($f^*_{A^\perp} = 0$), because if they weren't, then we would have $\|f^*\|_{\mathcal{H}} > \|f^*_A\|$, and thus $G(\|f^*\|_{\mathcal{H}}) > G(\|f^*_A\|)$. This would mean that f^*_A would attain a strictly lower value with respect to the objective than f^*, which contradicts f^* being a global minimizer. □

This is a very important result, which explains what we said in Section 2.2, regarding kernel methods trained on points $x_1, \dots, x_N \in \mathcal{X}$ producing models of the form $f(x) = \sum_{i=1}^N \alpha_i k(x_i, x)$. The Representer Theorem tells us that even if we were allowed to search through the much larger space of function $\mathcal{H}$, this would not help us attain better performance on our training objective function, because there will always be a solution in $span(\{k(x_i, x) \,|\, i \in [m]\})$ that is at least as good.

Another important consequence of the Representer Theorem is that if we consider the problem of learning a linear model on top of an explicit feature mapping $\phi(x)$, the optimal model w^* will be of the form $w^* = \sum_{i=1}^N \alpha_i \phi(x_i)$. In particular, this means that if define $\hat{\phi}(x) = [\phi(x), 1]$ by appending a 1 to the end of the $\phi(x)$ vector, we know the optimal model $\hat{w}^* = [w^*, b^*]$ trained on top of these $\hat{\phi}(x)$ vectors will have the form $w^* = \sum_{i=1}^N \alpha_i \phi(x_i)$ and $b^* = \sum_{i=1}^N \alpha_i$. Viewing this from the kernel perspective, appending a 1 to $\phi(x)$ corresponds to adding 1 to the value of all kernel evaluations: $\hat{k}(x, x') = \langle \hat{\phi}(x), \hat{\phi}(x') \rangle = k(x, x') + 1$. Thus, the Representer Theorem tells us that the optimal model in the RKHS corresponding to the kernel $\hat{k}$ will take the form

$$\begin{aligned} f^*(x) &= \sum_{i=1}^N \alpha_i \hat{k}(x, x_i) \\ &= \sum_{i=1}^N \alpha_i k(x, x_i) + \sum_{i=1}^N \alpha_i. \end{aligned}$$

As you can see, the optimal bias term b^* will always be equal to $\sum_{i=1}^N \alpha_i$, which implies that we can find the optimal model of the form $\sum_{i=1}^N \alpha_i k(x, x_i) + b$ by simply considering the modified kernel $\hat{k}$, and searching for the optimal model of the form $\sum_{i=1}^N \alpha_i \hat{k}(x, x_i)$. One important caveat to this is that we are assuming that we are including the bias term in the norm $\|f\|_{\mathcal{H}}$ of the model in the RKHS. Viewing this from the primal perspective, this corresponds to regularizing the bias term b in the model $[w, b]$, something which is often not done. This raises the question of whether there exists an extension of the representer theorem which applies to models for which the bias term is not regularized. This is addressed by the "Semiparametric Representer Theorem" [Schölkopf *et al.*, 2001].

Theorem 3. *(Semiparametric Representer Theorem, adapted from [Schölkopf* et al.*, 2001]): Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric positive definite kernel, $\mathcal{H}$ its RKHS, $G : \mathbb{R} \to \mathbb{R}$ a non-decreasing function, $L : (\mathcal{X} \times \mathcal{Y} \times \mathbb{R})^N \to \mathbb{R} \cup \{+\infty\}$ an arbitrary loss function, and $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ a set of N labeled points. Further, suppose we are given a set of M real-valued functions $\{\psi_p\}_{p=1}^M$ on $\mathcal{X}$, with the property that the $N \times M$ matrix Ψ with $\Psi_{ip} = \psi_p(x_i)$ has rank M. Then, letting $V = span(\{\psi_p\}_{p=1}^M)$, the optimization problem*

$$\underset{\substack{\tilde{f}=f+g,\\ f\in\mathcal{H},\, g\in V}}{\arg\min}\; G(\|f\|_{\mathcal{H}}) + L((x_1, y_1, \tilde{f}(x_1)), \ldots, (x_N, y_N, \tilde{f}(x_N)))$$

has a solution of the form $\tilde{f}^* = \sum_{i=1}^N \alpha_i k(x_i, \cdot) + \sum_{p=1}^M \beta_p \psi_p(\cdot)$. Furthermore, if G is a strictly increasing function, then any solution has this form.

Proof. The proof of this theorem is very similar to the one for the Representer Theorem. Let $\tilde{f} = f_A + f_{A^\perp} + g$ and $\tilde{f}_A = f_A + g$, where $A = span(k(x_1, \cdot), \ldots, k(x_N, \cdot)) \subset \mathcal{H}$, $f_A \in A$, $f_{A^\perp} \in A^\perp$, and $g \in V$. Letting $f = f_A + f_{A^\perp}$, by the orthogonality of $f_{A^\perp}$ with A we have that $f(x_i) = f_A(x_i)$ for any x_i in the labeled sample (same as Representer Theorem proof). Thus, $\tilde{f}(x_i) = \tilde{f}_A(x_i)$, and $G(\|f\|_{\mathcal{H}}) + L((x_1, y_1, \tilde{f}(x_1)), \ldots, (x_N, y_N, \tilde{f}(x_N))) \geq G(\|f_A\|_{\mathcal{H}}) + L((x_1, y_1, \tilde{f}_A(x_1)), \ldots, (x_N, y_N, \tilde{f}_A(x_N)))$. Thus, if $\tilde{f}$ is a minimizer for the above optimization problem, so is $\tilde{f}_A$, proving the first part of the theorem. The second part follows easily (as it does in the Representer Theorem proof). □

Now, if we consider the case where we have a single function $\psi(x) = 1\ \forall x \in \mathcal{X}$, then solving an optimization problem like the one above corresponds to searching through all functions of the form $\tilde{f} = f + b$, where f is in the RKHS $\mathcal{H}$, $b \in \mathbb{R}$, and the bias term b is not regularized. Leveraging the above theorem, it follows that the minimizer of this optimization problem will be of the form $\tilde{f}^* = \sum_{i=1}^N \alpha_i k(x_i, \cdot) + b$. Thus, we have shown how to extend the representer theorem to the more general case where an unregularized bias term is allowed.

2.5 Neural networks

Neural networks are a very flexible family of non-linear models, which are generally trained using gradient methods, and which have recently achieved remarkable performance on numerous empirical tasks. There are various different neural network architectures, designed for different types of tasks. However, all these architectures share some key properties:

1. They transform their input using a composition of linear and non-linear functions.
2. They are typically trained using gradient descent methods. The most common training algorithm is called *backpropagation*, which is an algorithm for efficiently computing the gradient of the loss function with respect to all the parameters in the network (see Section 2.5.1).
3. The optimization problem of finding the parameters which minimize the training objective is non-convex. As a result, lots of tricks are employed during training in order to improve the chances of the training algorithm finding a good model.

An important class of neural networks are called *fully-connected feedforward neural networks*. These networks make predictions on an input x as follows:

$$f(x) = \sigma_R(W_R \cdot \sigma_{R-1}(...W_2 \cdot \sigma_1(W_1 \cdot x + b_1) + b_2...) + b_R), \tag{2.7}$$

where $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $b_i \in \mathbb{R}^{d_i}$, $R \in \mathbb{N}$, and the $\sigma_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ functions are called activation functions. These activation functions typically transform their input in an element-wise fashion, which is generally non-linear. In Table 2.4, we show a list of common activation functions. Note that all these functions operate on scalars, except for softmax and maxout, which take vectors as input. The softmax function is commonly used at the final layer, to convert the output into a probability distribution. This is particularly important for classification problems, where it is often desirable for a model to produce a probability distribution over the set of classes. In the classification context, using the softmax function to convert the output of the network into a probability distribution also allows the network to be trained using the cross-entropy (CE) loss function. On a single example x, with true label y, the cross-entropy loss penalizes a model f which assigned probability $f_y(x)$ to the correct label as follows:

$$L_{CE}(f(x), y) = -\log(f_y(x)).$$

Activation Function Name	**Equation**
Identity	$\sigma(x) = x$
Sign function	$\sigma(x) = \text{sign}(x)$
Sigmoid	$\sigma(x) = (1 + \exp(-x))^{-1}$
Tanh	$\sigma(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
ReLU (Rectified Linear Unit)	$\sigma(x) = \max(0, x)$
Softplus	$\sigma(x) = \ln(1 + \exp(x))$
Maxout	$\sigma(x) = \max_i x_i$
Softmax	$\sigma_i(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$

Table 2.4: Activation functions for neural networks. For the maxout and softmax activation functions, the input x is a vector. For all others, it is a scalar.

Importantly, minimizing the cross-entropy objective on a training set corresponds to maximizing the probability of the dataset under the probability model f being trained ($\arg\min_f \sum_{i=1}^{N} - \log(f_{y_i}(x_i)) = \arg\max_f \prod_{i=1}^{n} f_{y_i}(x_i)$).

As we will see in Section 2.6, for speech recognition systems it is generally important for the model to output a probability distribution, given that the task of the speech recognition system is to output the sequence of words *most likely* to have produced a given sequence of acoustic features.[9]

2.5.1 Backpropagation

Neural networks are typically trained using the backpropagation algorithm (short for "backward propagation of errors"), which is simply an efficient way of calculating the gradient of a network's objective function with respect to its parameters. We now review how this algorithm works, partially following the explanation of Guenter *et al.*[2013] when discussing the gradients corresponding to matrix multiplication.

Suppose we have a feedforward DNN as in Equation 2.7, which outputs $h^R = \sigma_R(W^R \cdot \sigma_{R-1}(...W^2 \cdot \sigma_1(W^1 \cdot x + b^1) + b^2...) + b^R)$, given the input $x \in \mathbb{R}^{d_1}$.[10] Suppose further that

[9]Generally, a Hidden Markov Model (HMM) is used as the probability model; see Section 2.6 for more details.

[10]We use superscripts here for the W, h, and b matrices/vectors in order to allow for subscripts to index individual

we have some loss function $L(h^R)$ on the output of this network which we are trying to minimize using gradient descent. Now, assume we know the gradient of this objective function with respect to the output $h^r = \sigma_r(W^r h^{r-1} + b^r)$ of its r^{th} layer. We will now show that this is all that is needed in order to compute the gradients with respect to W^r, h^{r-1}, and b^r. Importantly, this allows for the algorithm to proceed recursively, because knowing the gradient with respect to h^{r-1} then allows the equivalent computations at the layers below. Note that to be precise, I will be considering the gradients of the following functions:

$$\begin{aligned}
L_1^r(h^r) &= L(\sigma_R(W^R \cdot \sigma_{R-1}(...\sigma_{r+1}(W^{r+1} \cdot h^r + b^{r+1})\ldots) + b_R) \\
L_2^r(c^r) &= L_1^r(\sigma_r(c^r)) \\
L_3^r(W^r, b^r) &= L_2^r(W^r h^{r-1} + b^r) \\
c^r &= W^r h^{r-1} + b^r \\
h^r &= \sigma_r(c^r) \\
h^0 &= x \\
L_1^R(h^R) &= L(h^R).
\end{aligned}$$

In order to derive the updates used for backpropagation, we will use the multivariate version of the chain rule, for functions $s\colon \mathbb{R}^m \to \mathbb{R}$, $g\colon \mathbb{R}^m \to \mathbb{R}^n$, and $f\colon \mathbb{R}^n \to \mathbb{R}$, where $s(t) = f(g(t)) = f(g_1(t_1, \ldots, t_m), \ldots, g_n(t_1, \ldots, t_m))$, we have the following form for the partial derivatives of s:

$$\frac{\partial s}{\partial t_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial g_j} \frac{\partial g_j}{\partial t_i}. \tag{2.8}$$

For the calculations that follow, let's assume that $h^r \in \mathbb{R}^{d_r}$, that $W^r \in \mathbb{R}^{d_r \times d_{r-1}}$ is the parameter matrix at this layer, that $b^r \in \mathbb{R}^{d_r}$ is the bias parameter at this layer, and that $h^{r-1} \in \mathbb{R}^{d_{r-1}}$ is the output of the previous layer. We assume we know $dh^r = \frac{\partial L_1^r}{\partial h^r}$, which is the vector of partial derivatives of L_1^r with respect to every element in h^r ($dh_i^r = \frac{\partial L_1^r}{\partial h_i^r}$). Now, how do we compute $dW^r = \frac{\partial L_3^r}{\partial W^r}$, $db^r = \frac{\partial L_3^r}{\partial b^r}$, and $dh^{r-1} = \frac{\partial L_1^{r-1}}{\partial h^{r-1}}$, given dh^r? First, let $c^r = W^r h^{r-1} + b^r$, so that $h^r = \sigma_r(c^r)$. We can see by a straightforward application of the one-dimensional chain rule that $dc_i^r = \frac{\partial L_2^r}{\partial c_i^r} = \frac{\partial L_1^r}{\partial h_i^r}\frac{\partial h_i^r}{\partial c_i^r} = dh_i^r \sigma_r'(c_i^r)$, where σ_r' is the derivative of the activation function σ_r.

elements of these matrices/vectors in the calculations below. We apologize for this notation change, relative to Eq. 2.7.

Thus, $dc^r = dh^r \circ \sigma_r'(c^r)$, where $\circ$ denotes the element-wise vector product. Now, we compute $dW^r = \frac{\partial L_3^r}{\partial W^r}$ and $db^r = \frac{\partial L_3^r}{\partial b^r}$ using the multivariate chain rule:

$$\frac{\partial L_3^r}{\partial W_{kl}^r} = \sum_{i=1}^{d_r} \frac{\partial L_2^r}{\partial c_i^r} \frac{\partial c_i^r}{\partial W_{kl}^r} \tag{2.9}$$

$$\frac{\partial L_3^r}{\partial b_k^r} = \sum_{i=1}^{d_r} \frac{\partial L_2^r}{\partial c_i^r} \frac{\partial c_i^r}{\partial b_k^r} \tag{2.10}$$

By the definition of matrix multiplication ($c^r = W^r h^{r-1} + b^r$), we can see that

$$c_i^r = \sum_{j=1}^{d_{r-1}} W_{ij}^r h_j^{r-1} + b_i^r,$$

and thus that

$$\frac{\partial c_i^r}{\partial W_{kl}^r} = \begin{cases} h_l^{r-1} & \text{if } k = i \\ 0 & \text{else.} \end{cases}$$

$$\frac{\partial c_i^r}{\partial b_k^r} = \begin{cases} 1 & \text{if } k = i \\ 0 & \text{else.} \end{cases}$$

Equations 2.9 and 2.10 can now be simplified:

$$\begin{aligned} \frac{\partial L_3^r}{\partial W_{kl}^r} &= \frac{\partial L_2^r}{\partial c_k^r} \cdot h_l^{r-1} \\ \Rightarrow dW^r &= dc^r \cdot (h^{r-1})^T \\ \frac{\partial L_3^r}{\partial b_k^r} &= \frac{\partial L_2^r}{\partial c_k^r} \\ \Rightarrow db^r &= dc^r \end{aligned}$$

With the same approach, we can show that $dh^{r-1} = (W^r)^T \cdot dc^r$. We have now shown that given dh^r, we can calculate dW^r, db^r, and dh^{r-1} as follows:

$$\begin{aligned} dW^r &= dc^r \cdot (h^{r-1})^T \\ dh^{r-1} &= (W^r)^T \cdot dc^r \\ db^r &= dc^r, \\ \text{where} \quad dc^r &= dh^r \circ \sigma'(W^r h^{r-1}). \end{aligned}$$

These recursive equations constitute the core of the backpropagation algorithm, which allows the gradient with respect to all the parameters in the network to be computed efficiently, starting from the output from the network, and moving toward the input of the network. The "base case" for this recursion is when r is equal to the depth R of the network. In this case, it is clear that one can easily compute the gradient dh^R of L_1^R with respect to the output $h^R = \sigma_R(W^R h^{R-1})$, because L_1^R is a direct function of h^R. For example, in the case where L corresponds to the least squares loss, and h^* is the vector of targets for the regression, then $L_1^R(h^R) = \frac{1}{2}\|h^R - h^*\|_2^2$, and $dh^R = h^R - h^*$.

Often, we call the "forward pass" of a network the process of computing the output from the input, and we call the "backward pass" the process of computing the gradients of the network. Notice that the forward pass for each layer requires one matrix multiplication $c^r = W^r h^{r-1}$ involving roughly $d_r d_{r-1}$ multiply-adds, while the backward pass requires two such matrix multiplications, and is thus twice as expensive computationally.

2.5.2 Other architectures

Two important types of neural network architectures, aside from fully-connected feedforward networks, are convolutional neural networks (CNNs) and recurrent neural networks (RNNs). We will now discuss these one at a time.

2.5.2.1 Convolutional Neural Networks (CNNs)

Although they can be used in a variety of domains (including speech recognition and natural language processing), CNNs are best known for their success on computer vision tasks [Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014; He *et al.*, 2016]. In 2012, Krizhevsky *et al.* won the ImageNet Large Scale Visual Recognition Challenge [Russakovsky *et al.*, 2015] by an impressive 10% margin in Top-5 error rate. CNNs are in a sense quite similar to fully-connected networks, but they perform a very particular kind of linear transformation known as convolution. Convolution corresponds to performing a sequence of dot products between a $d_p \times d_q$ "kernel" with the various $d_p \times d_q$ patches which make up an image.[11] Each kernel can be seen as a pattern detector, which

[11]This use of the word "kernel" is not to be confused with the kernels discussed elsewhere in this paper (e.g., Section 2.2). Additionally, when we say the "dot product" between two $d_p \times d_q$ matrices, we mean the regular dot product between the "flattened" versions of these matrices (i.e., turn the matrices into vectors by concatenating all their columns,

scans the full image to see if instances of a given pattern are found (a large dot product between a $d_p \times d_q$ kernel and an image patch indicates the presence of the pattern). With one-dimensional input, we can visualize convolution of a vector $x \in \mathbb{R}^8$ with a kernel $k = [w_1, w_2, w_3] \in \mathbb{R}^3$ as follows:

$$h = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix} \cdot x.$$

Thus, convolution is equivalent to matrix multiplication with a sparse matrix, which uses the same parameters (in this case, $\{w_1, w_2, w_3\}$) across the various rows of the matrix. This is called "parameter sharing", and is very important for both memory efficiency (can store the entire 6×8 matrix above with only 3 parameters), as well as generalization performance (by reducing the number of parameters, the complexity of the model is decreased, and more data is used to train each parameter). Furthermore, convolution can be performed very quickly on GPUs, using the Fast Fourier Transform [Vasilache *et al.*, 2015], thus also making training faster. This type of structured matrix multiplication used for convolution is particularly well suited for image recognition problems, as it creates models which exhibit some translation invariance *by design*.

There are several other important differences between CNNs and fully-connected networks. For example, CNNs typically have several convolutional layers, followed by some fully-connected layers. In addition, after applying the element-wise non-linearity on top of a convolution operation, an operation called "max-pooling" is often performed [Krizhevsky *et al.*, 2012]; this corresponds to applying a maxout non-linearity, which summarizes a small neighborhood of outputs from the convolutional layer by simply outputting the maximum value in the neighborhood. We refer the reader to [Goodfellow *et al.*, 2016] for a more complete discussion of CNNs.

---

and then take the dot product between these vectors.)

2.5.2.2 Recurrent Neural Networks (RNNs)

RNNs are designed for the processing of *sequences* of inputs and outputs, which in general can vary in length. Here, we will define a sequence to be an ordered list of elements from a set (e.g., $\mathcal{X}$), and we will denote it by $(x_1, \ldots, x_T) \equiv (x_t)_{t=1}^T$, where each $x_t \in \mathcal{X}$. A sequence can be though of as a signal with a temporal component. The most straightforward RNN processes the input in the following recursive manner:

$$\begin{aligned} h_t &= \sigma(Wx_t + Uh_{t-1}) \\ \hat{y}_t &= \hat{\sigma}(Vh_t), \end{aligned}$$

where we define $h_0 = \mathbf{0}$, and the above equations are for $t > 0$. The objective function of the network is a differentiable function, which penalizes the output sequence $(\hat{y}_t)_{t=1}^T$ with respect to the true output $(y_t)_{t=1}^T$. For example, if $\hat{\sigma}$ is the softmax function, then the loss L_t at time t could be the cross-entropy objective $\sum_{i=1}^C -y_{ti} \log(\hat{y}_{ti})$; here, we use $\hat{y}_{ti}$ to denote the probability under the model that the label at time t is $i \in [C]$, and we let y_{ti} be 1 if the true label at time t is i, and 0 otherwise. Then, the total loss L for the network on a given input could be $\sum_{t=1}^T L_t$. As usual, the parameters of the network (U, V, W) are trained using gradient methods. RNNs have the potential to leverage information from much earlier in a sequence in order to decide what to output at a given time step. This ability to capture long-range dependencies in the input is very important in certain application areas, like natural language processing (NLP). However, training RNNs can suffer from a variety of problems, including vanishing and exploding gradients. Long Short-Term Memory networks (LSTMs) are an effective modification to the RNN architecture described above, designed to deal with this problem [Hochreiter and Schmidhuber, 1997]. LSTMs give state of the art performance on a variety of tasks, including parsing in NLP [Kiperwasser and Goldberg, 2016], as well as acoustic modeling and language modeling in speech recognition systems [Saon *et al.*, 2017; Xiong *et al.*, 2017]. We refer the reader to [Goodfellow *et al.*, 2016] for a more complete discussion of RNNs.

2.6 Automatic speech recognition (ASR)

The goal of automatic speech recognition (ASR) systems is to accurately transcribe human speech. The input to an ASR system is an audio recording of a human speaking, which is typically called an "utterance." An utterance is typically represented as a sequence of "frames", where each frame is represented as a vector of acoustic features. A frame corresponds to a short segment of speech (e.g., 25 ms), and typically there is overlap between neighboring frames (for example, a 10 ms shift between frames). More formally, we will consider utterances represented as $X = (x_1, \ldots, x_T)$, where $x_i \in \mathbb{R}^d$ corresponds to the acoustic features of the i^{th} frame. The goal is to output the sequence of words $w^* = (w_1, \ldots, w_L)$ most likely to have generated the utterance X, under a probability model p:

$$\begin{aligned} w^* &= \arg\max_w p(w|X) \\ &= \arg\max_w \frac{p(X|w)p(w)}{p(X)} \\ &= \arg\max_w p(X|w)p(w) \\ &= \arg\max_w \log(p(X|w)) + \log(p(w)) \qquad (2.11) \end{aligned}$$

$p(X|W)$ is called the *acoustic model*, and $p(w)$ is the *language model*. The job of the acoustic model is to assign a score ($\log(p(X|w))$) describing how well the acoustics match a given word sequence w, while the job of the language model is to give a score ($\log(p(w))$) quantifying how likely a sequence of words w is in a given language, independent of the acoustics. Acoustic models are trained on large amounts of transcribed audio, while language models are trained on large amounts of text. The ASR system returns the sequence of words w^* maximizing the sum of scores from these two different models, as expressed in Equation 2.11. The process of searching for the optimal word sequence w^* is called *decoding*.

As an example, consider an utterance in which a speaker says "the dog barked." Perhaps, when the speaker pronounced this utterance, they said it in such a way that "barked" sounded more liked "parked". What should the ASR system output? Given that the speaker pronounced "barked" more like "parked", let's assume the acoustic model assigned a higher score to $P(X \mid \text{"the dog parked"})$ than to $P(X \mid \text{"the dog barked"})$. However, given that dogs bark much more often than they drive, the language model would likely assign a much higher score to "the dog barked" than "the dog parked”. Thus, the sum of the acoustic and language models scores would likely be higher for “the dog barked”, and the system would output this word sequence as its prediction.

In practice, given that these two models are trained separately and with different criteria, their relative scales might not be well suited for the ASR task. Thus, a scalar α is chosen to maximize the empirical performance of the ASR system which outputs $\arg\max_w \log(p(X|w)) + \alpha \log(p(w))$. Note that this corresponds to replacing the language model $p(w)$ with $p(w)^\alpha/Z$, where Z simply normalizes $p(w)^\alpha$ to sum to 1. Thus, it is still a proper probability model. Intuitively, this hyperparameter also allows an ASR system to place more relative weight on whichever of the two models is stronger.

ASR systems are generally evaluated using a metric known as “word error rate” (WER). This metric is calculated as follows:

1. The test utterances are processed by the ASR system, which makes a prediction for every utterance.

2. For each test utterance, the ASR output is aligned with the reference transcription, using a dynamic programming string alignment algorithm.

3. Given these alignments, the total number of substitutions (S), insertions (I), and deletions (D) are counted across *all* utterances, and the WER is calculated as:

$$WER = \frac{S + D + I}{N},$$

   where N is the total number of words in the reference transcriptions.

For languages whose primary units are not words (e.g., character-based languages like Cantonese), we can define the appropriate metric analogous to word error rate. For example, for Cantonese, the “character error rate” (CER) is used. In this thesis, we will use the more general term “token error rate” (TER) to refer to any of these metrics.

Given that an important part of this thesis is the training of acoustic models, we will now discuss how acoustic models are trained in more detail.

2.6.1 Acoustic model training

The most common type of acoustic model uses a hidden Markov model (HMM) in order to model words and sentences. At a high level, words and sentences are modeled as HMMs over a finite set of states $\{q_1, \ldots, q_S\}$. In the simplest case, the states represent phonemes, which are the smallest units of sound which distinguish one word from another in a given language. There are two important sets of parameters associated with HMMs: The first are the transition probabilities a_{ij}, representing the probability of transitioning from state q_i to q_j, given that the current state is q_i. The second are the emission probabilities $b_i : \mathbb{R}^d \to \mathbb{R}$, where $b_i(x) = p(x|q_i)$ is the probability of observing an acoustic vector x, given that the current state is q_i. For many years, the dominant model for the emission probabilities was a Gaussian mixture model (GMM); in this case, $p(x|q_i) = \sum_{i=1}^{M} c_i f(x; \mu_i, \Sigma_i)$, where $f(x; \mu_i, \Sigma_i)$ is the probability density function (pdf) of a multivariate normal distribution with mean μ_i and covariance matrix Σ_i, and the $c_i \in \mathbb{R}$ are the mixture weights ($c_i \geq 0$, $\sum_{i=1}^{M} c_i = 1$). Historically, all of these parameters (a_{ij}, c_i, μ_i, Σ_i) were trained using the Expectation-Maximization (EM) algorithm [Dempster *et al.*, 1977]. Nowadays, the more common approach is to use neural networks for acoustic modeling, which we discuss in Section 2.6.2.

Above, we mentioned that each state q_i could represent a phoneme. However, it is possible to attain much stronger performance by having more fine-grained states. For example, the way a person pronounces a phoneme is very affected by the phonemes which come before and after it. Furthermore, the pronunciation of a phoneme also varies *during* its pronunciation. These two observations suggest that instead of modeling phonemes we should model *context-dependent* phonemes and that each context-dependent phoneme should be split into three states: a beginning, a middle, and an end state. Two common choices for context-dependent modeling are to use *triphones* or *quinphones*: a triphone corresponds to a sequence of three phonemes, and a quinphone corresponds to a sequence of five phonemes, where in both cases the central phoneme is the one being pronounced during the state. Note, however, that this creates an explosion in the number of states. For example, using quinphones, the total number of states becomes $3S^5$; this is because we create a unique HMM state for the beginning, middle, and end for all S^5 quinphones, where S is the number of phonemes. For English, this would be approximately 500 million states (or around 250 thousand states, using triphones), using the standard 44 English phonemes. This poses two problems: data sparsity, and model size. The problem of data sparsity is that there would likely not be enough examples of each context-dependent state in the training data. In fact, some states might not appear at all. In this case, this would make it impossible to effectively learn the parameters of these rare or unobserved states. The other problem is model size; given that we have an emission model for each state, unless we somehow intelligently share parameters between the emission models of each state, the amount of memory required for the full acoustic model will scale linearly with the number of states. For example, in the case of GMMs, at the very minimum we must store a single d-dimensional mean for each state (if we assume identity covariance). This would have the additional effects of making training as well as decoding very slow.

In order to address all of these problems related to having a very large number of context-dependent states, the most common approach is to cluster states which are similar to one another using decision tress [Hwang *et al.*, 1993; Young *et al.*, 1994]. These clustered states are called *senones*, and there are typically on the order of 10^3 or 10^4 such states. For example, in the latest state of the art English ASR system, Saon *et al.* use an ensemble of acoustic models, each with 32000 states [2017].

For more details on the use of HMMs for speech recognition, see [Gales and Young, 2007]. For information on how weighted finite state transducers (WFSTs) are used in order to efficiently implement these HMMs in practice, combining the language model and the acoustic model into a single large but efficient network, see [Mohri *et al.*, 2002].

2.6.2 Using neural networks for acoustic modeling

As mentioned above, neural networks can also be used for acoustic modeling [Morgan and Bourlard, 1995]. However, neural network acoustic models differ in an important way from GMMs; instead of modeling the emission probability $p(x|q)$ directly, neural networks model $p(q|x)$—the posterior probability distribution over all the HMM states given the acoustic feature vector x. Typically, a neural network with a softmax output is used, where the dimension of the output is equal to the number of states. Thus, when a network is fed an example x, its output can be seen as the conditional probability distribution $p(q|x)$ over the states that may have generated that example. In order to use this “flipped” neural acoustic model in the existing HMM framework, we can simply use Bayes’

Rule:

$$\begin{aligned} p(x|q) &= \frac{p(q|x)p(x)}{p(q)} \\ &\propto \frac{p(q|x)}{p(q)} \end{aligned}$$

Above, $p(q)$ is a prior distribution over the HMM states, while $p(q|x)$ is the output of the neural acoustic model. One important detail, however, is that in order to train these probability models, we must have a dataset of labeled examples (x_i, q_i). However, training sets for speech recognition systems are typically in the form of transcribed utterances; this means that the only information which is provided to the speech recognition system is what sequence of words was pronounced for each utterance. Thus, we do not know in advance which state q was being pronounced for every frame in an utterance. In order to deal with this problem, the following approach is taken: first, a GMM system is trained using the EM algorithm on the training set. Then, this model is used to *force align* the provided transcriptions with the acoustic frames, mapping each frame x_i with its corresponding HMM state q_i; this can be done using the Viterbi algorithm [Viterbi, 1967]. These automatically produced frame-level labels are then used in order to train the $p(q)$ and $p(q|x)$ models. The $p(q)$ model is trained through counting (e.g., the number of times a given state appeared in the training data, over the total number of states observed), while $p(q|x)$ is the output of a neural network, often trained using the cross-entropy loss function, as discussed Section 2.5. In Section 4.1.3 we discuss alternative methods for training acoustic models, which more directly attempt to lower the token error rate (TER) of the ASR system. These techniques are known as "sequence training" methods, and they can lead to large improvements in the TER of the ASR system.

Chapter 3

Related work

Scaling up kernel methods has been a long-standing and actively studied problem [Bottou *et al.*, 2007; Smola, 2014; DeCoste and Schölkopf, 2002; Platt, 1998; Tsang *et al.*, 2005; Clarkson, 2010]. Approximating kernels by constructing explicit finite-dimensional feature representations, where the dot product between these representations approximates the kernel function, has emerged as a powerful technique (e.g., [Williams and Seeger, 2001; Rahimi and Recht, 2007]). The Nyström method constructs these feature maps, for arbitrary kernels, via a low-rank decomposition of the kernel matrix [Williams and Seeger, 2001]. For shift-invariant kernels, the random Fourier feature technique of Rahimi and Recht [2007] uses random projections in order to generate the features. Random projections can also be used to approximate a wider range of kernels [Kar and Karnick, 2012; Vedaldi and Zisserman, 2012; Hamid *et al.*, 2014; Pennington *et al.*, 2015]. Many recent works have been developed to speed-up the random Fourier feature approach to kernel approximation. One line of work attempts to reduce the time (and memory) needed to compute the random feature expansions by imposing structure on the random projection matrix [Le *et al.*, 2013; Yu *et al.*, 2015]. It is also possible to use doubly-stochastic methods to speed-up stochastic gradient training of models based on random features [Dai *et al.*, 2014]. For kernels with sparse feature expansions, [Sonnenburg and Franc, 2010] show how to efficiently scale kernel SVMs to datasets with up to 50 million training samples by using sparse vector operations for parameter updates.

Despite much progress in kernel approximation, there have been very few applications of these methods to challenging large-scale problems, or comparisons with deep learning on these tasks. Notable exceptions are the following: on image recognition problems, it has been shown that random Fourier features can be used to replace the fully-connected layers on top of the convolutional layers in the convolutional neural network (CNN) known as AlexNet [Krizhevsky *et al.*, 2012], and achieve comparable performance on the ImageNet 2012 dataset [Dai *et al.*, 2014; Yang *et al.*, 2015]. However, these kernel models remain intrinsically tied to the CNN used to train their input features, thus limiting the impact of this work. In ASR, the only existing works[1] applying kernel approximation methods have been quite limited in scope [Huang *et al.*, 2014; Chen *et al.*, 2016], using the relatively easy and small TIMIT dataset. While these papers pose the acoustic modeling classification task as a regression problem that they solve in specialized ways, we simply incorporate the random Fourier features into a multinomial logistic regression model, and are able to outperform the results on TIMIT from this previous work. In general, a detailed evaluation of kernel approximation methods on large-scale ASR tasks, together with a thorough comparison with DNNs, has not been performed. Our work fills this gap, tackling challenging large-scale acoustic modeling problems, where deep neural networks achieve strong performance. Additionally, we provide a number of important improvements to the kernel methods, which boost their performance significantly.

One contribution of our work is to introduce a feature selection method that works well in conjunction with random Fourier features in the context of large-scale multi-class classification problems. Recent work on feature selection methods with random Fourier features, for binary classification and regression problems, includes the Sparse Random Features algorithm of Yen *et al.* 2014. This algorithm is a coordinate descent method for smooth convex optimization problems in the (infinite) space of non-linear features: each step involves solving a batch ℓ_1-regularized convex optimization problem over randomly generated non-linear features (note that a natural extension of this method to multi-class problems is to use mixed norms such as ℓ_1/ℓ_2). Here, the ℓ_1-regularization may cause the learned solution to only depend on a subset of the generated features, allowing the others to be discarded. A drawback of this approach is the computational burden of fully solving many batch optimization problems, which is prohibitive for large data sets. In our attempts to implement an online variant of this method, using FOBOS [Duchi and Singer, 2009] and ℓ_1/ℓ_2-regularization for the multi-class setting, we observed that very strong regularization was required to obtain any intermediate sparsity, which in turn severely hurt prediction performance.

[1]Here, we are excluding the results presented in this thesis.

Effectively, the regularization was so strong that this method basically selected features uniformly at random from the pool of features. Our approach for selecting random features is more efficient, and more directly ensures sparsity, than regularization. The basic idea behind our approach is to iteratively train a model over a batch of random features, and to then replace the features whose corresponding rows in the parameter matrix have small ℓ_2 norm. This method bears some similarity to the methods of pruning neural networks which eliminate parameters whose magnitudes are below a certain threshold [Ström, 1997; Han *et al.*, 2015]; a difference is that in our method, we eliminate entire *rows* of the parameter matrix, instead of individual entries.

Recent years have seen huge improvements in the performance of state-of-the-art speech recognition systems. The most important factors leading to this success have been the following: sequence training [Povey *et al.*, 2008; Povey *et al.*, 2016], speaker adaptation through the use of i-vectors [Dehak *et al.*, 2011], training on large datasets [van den Berg *et al.*, 2017; Saon *et al.*, 2017], and improved deep architectures for both language modeling [Mikolov *et al.*, 2010; Sundermeyer *et al.*, 2012; Saon *et al.*, 2017], and acoustic modeling. For acoustic modeling, CNNs [Krizhevsky *et al.*, 2012; Sainath *et al.*, 2013b; Soltau *et al.*, 2014; Simonyan and Zisserman, 2014; Sercu and Goel, 2016; He *et al.*, 2016; Saon *et al.*, 2017] along with Long Short Term Memory (LSTM) networks [Sak *et al.*, 2014; Saon *et al.*, 2017], have been developed to leverage the time-frequency structure of the speech signal, and achieve better performance than fully-connected DNNs. The most recent state-of-the-art systems [Saon *et al.*, 2017; Xiong *et al.*, 2017] use an ensemble of LSTMs and CNNs for acoustic modeling. In [Saon *et al.*, 2016] they show an improvement of 1.3% in WER on the switchboard dataset when switching from a sigmoid DNN architecture to an LSTM, while in [Xiong *et al.*, 2016] they show that their ResNet CNN [He *et al.*, 2016] improves upon ReLU DNNs by 1.6%.

In the context of these recent advances, our results showing competitive performance with fully-connected feedforward sigmoid DNNs are significant, for a number of reasons. First of all, while no longer being state-of-the-art, DNNs still attain very strong performance on the acoustic modeling task. Second, fully-connected feedforward DNNs remain an important class of models, which are used widely (e.g., [Andor *et al.*, 2016]). Furthermore, fully-connected layers are an important building block within more complex and specialized deep learning architectures [Simonyan and Zisserman, 2014; He *et al.*, 2016]. Additionally, we believe that it should be a matter of deep importance to the research community to discover when and why deep architectures are necessary, while simultaneously working to explore which other families of models might be able to compete; we think kernel methods are an important family of models to consider, as they lend themselves to much simpler interpretation, as well as cleaner theoretical analysis based on convex optimization, relative to DNNs. For future work, we would like to develop specialized kernel methods to better leverage the structure in the speech signal, in a manner similar to CNNs and LSTMs.

This work also contributes to the debate on the relative strengths of deep and shallow models. Kernel models can generally be seen as shallow models, given that they involve learning a linear model on top of a *fixed* transformation of the data. Furthermore, as explained in Section 4.1.1, many types of kernels (including popular kernels like the Gaussian kernel and the Laplacian kernel) can actually be seen as a special case of a shallow neural network. Conversely, any neural network can be understood as a kernel model, in which the kernel function itself is learned. Classic results show that both deep and shallow neural networks, as well as kernel methods, are "universal approximators," meaning that they can approximate any real-valued continuous function with bounded support to an arbitrary degree of precision [Cybenko, 1989; Hornik *et al.*, 1989; Micchelli *et al.*, 2006]. However, a number of papers have argued that there exist functions which deep neural networks can express with exponentially fewer parameters than shallow neural networks [Montúfar *et al.*, 2014; Bianchini and Scarselli, 2014]. Other papers have argued that kernel methods may require a number of training samples which is exponential in the intrinsic dimension of the data manifold in order to generalize well, a problem known as the *curse of dimensionality* [Härdle *et al.*, 2004; Bengio and Lecun, 2007]. In [Ba and Caruana, 2014], the authors show that the performance of shallow neural networks can be increased considerably by training them to match the outputs of deep neural networks. In showing that kernel methods can compete with DNNs on large-scale speech recognition tasks, this work adds credence to the argument that shallow models can compete with deep networks.

In Chapter 6, we perform large-scale comparisons between the Nyström method [Williams and Seeger, 2001] and random Fourier features [Rahimi and Recht, 2007]. There is abundant literature about both of these methods; this work is often concerned with either (1) proposing an improvement to the method of interest [Le *et al.*, 2013; Yu *et al.*, 2015; Yen *et al.*, 2014; May *et al.*, 2016; Zhang *et al.*, 2008; Kumar *et al.*, 2009; Si *et al.*, 2014], (2) performing a theoretical analysis of the method [Rahimi and Recht, 2008; Gittens and Mahoney, 2013; Kumar *et al.*, 2012], or (3) performing an empirical evaluation of the method [Huang *et al.*, 2014; May *et al.*, 2017; Kumar *et al.*, 2012].

In spite of the abundant literature analyzing and building on each of these methods, there has been relatively little work attempting to understand the important differences between them. One notable exception is the work of Yang *et al.* [2012]; this work argues that from both theoretical and empirical perspectives, the Nyström method is preferable to RFFs, for a fixed number of features. They propose that the reason for this is that the Nyström method performs a data *dependent* transformation, while RFF performs a data *independent* transformation. In our work, we go beyond this existing work, in the following ways: (1) We perform experiments with *many* more random features than the previous work. We use up to $20{,}000$ Nyström features, and $1{,}600{,}000$ random Fourier features, whereas Yang et al. only use up to 1000. This exposes important differences between these two methods which are not evident in the smaller scale setting, while also allowing us to attain much stronger performance on all datasets. (2) In addition to running experiments on all six datasets used by Yang *et al.*, we run experiments on TIMIT, a significantly larger and more challenging dataset. We also include results on the relatively large YearPred regression task. (3) We take into consideration the relative computational expense of computing m Nyström features compared to m random Fourier features. (4) We make a novel observation, that random Fourier features perform consistently better on classification and regression problems than Nyström features with comparable kernel approximation error. (5) We analyze, from both theoretical and empirical perspectives, the differences in the ways Nyström features and random Fourier features make approximation errors, and argue that these differences have a large effect on training.

Another important difference between our work and existing theoretical analyses of the Nyström method [Gittens and Mahoney, 2013; Kumar *et al.*, 2012], is that we analyze the *element-wise* errors made by the Nyström method in approximation the kernel matrix, whereas existing work generally analyzes the Frobenius norm or spectral norm of the full error matrix.

Chapter 4

Random Fourier features for acoustic modeling

In this chapter, we discuss our experiments using random Fourier features for acoustic modeling on four datasets, along with comparisons to fully-connected feedforward DNNs. These experiments constitute the largest scale application to date of kernel approximation methods to a domain in which DNNs dominate. We begin in Section 4.1 by providing an overview of the methods we leverage in our experiments, including (1) our incorporation of random Fourier features in a multinomial logistic regression model, (2) our use of low rank decompositions of the output matrices of our models [Sainath *et al.*, 2013a], and (3) our use of a metric called "Entropy Regularized Perplexity" to determine learning rate decay and early stopping [Lu *et al.*, 2016]. We then move on to discussing our experiments: In Section 4.2 we describe the datasets we use, and our evaluation criteria. We then give an overview of our training procedure, and provide details regarding hyperparameter choices, in Section 4.3. We present our empirical results comparing the performance of kernel approximation methods to DNNs in Section 4.4, showing that the kernel methods match the DNNs on two out of four datasets. In Section 4.5 we discuss potential improvement to our DNN and kernel models which we *do not* include in this work, and explain our decisions. We conclude in Section 4.6.

4.1 Methods

In this section we describe three methods we leverage in order to get strong performance for our acoustic models.

4.1.1 Using kernel approximation methods for acoustic modeling

In order to train an acoustic model using kernel approximation methods, we can simply plug the feature vector $z(x)$ (for an acoustic frame x) into a multinomial logistic regression model:

$$p(y|x) = \frac{\exp\big(\langle \boldsymbol{\theta}_y,\ z(x)\rangle\big)}{\sum_{y'} \exp\big(\langle \boldsymbol{\theta}_{y'},\ z(x)\rangle\big)}. \tag{4.1}$$

The label y can take any value in $\{1, 2, \ldots, C\}$, each corresponding to a senone, and the parameter matrix $\Theta = [\boldsymbol{\theta}_1|\ldots|\boldsymbol{\theta}_C]$ is learned. Note that we also include a bias term, by appending a 1 to $z(x)$ in the equation above. We discuss how we train this model in Section 4.3.

In the case of random Fourier features, which is the method we use in this chapter, the model in Equation 4.1 can be seen as a shallow neural network (single hidden layer), with the following properties: (1) the parameters from the inputs x to the hidden units are set randomly, and are not learned; (2) the hidden layer uses $\cos(\cdot)$ as its activation function; (3) the parameters from the hidden units to the output units are learned (can be optimized with convex optimization); and (4) the softmax function is used to normalize the outputs of the network. See Figure 4.1 for a visual representation of this model architecture. Note that although using sinusoidal activation functions has been proposed previously [Goodfellow *et al.*, 2016], their use has remained quite rare in the deep learning context.

4.1.2 Linear bottlenecks

The number of senone state labels can be very large. In our experiments, this number varies from 147 to 5000. This significantly increases the number of parameters in $\Theta \in \mathbb{R}^{(D+1)\times C}$, where D is the number of random features, and C is the number of output classes. We can reduce this number with a *linear bottleneck* layer between the hidden layer and the output layer; the linear bottleneck corresponds to a low-rank factorization $\Theta = UV$ of the parameter matrix [Sainath *et al.*, 2013a]. We can think of UV as a lower-dimensional parametrization of Θ; instead of having a parameter

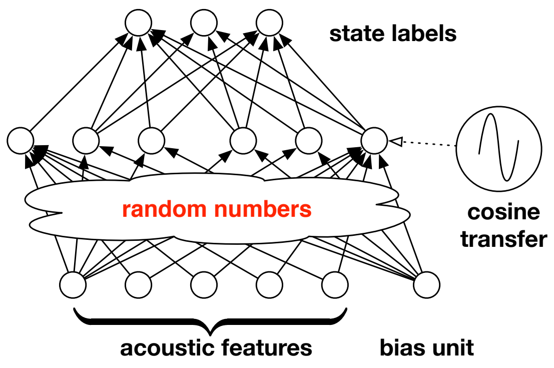

Figure 4.1: Kernel-acoustic model seen as a shallow neural network

for each element of Θ, our parameters correspond to the elements of U and V. If we let r denote the rank of this decomposition of Θ, then $U \in \mathbb{R}^{(D+1)\times r}$ and $V \in \mathbb{R}^{r\times C}$. This is particularly effective at reducing the number of parameters in our kernel models. Without this trick, the number of parameters is $dim(\Theta) = (D+1)C$, where $dim(\Theta)$ denotes the number of elements in Θ; with this trick, the number becomes $dim(U) + dim(V) = (D+1)r + rC$, which is significantly less than $(D+1)C$ when $r \ll \min(D, C)$. During training, we learn the parameters of U and V using gradient descent. Using a linear bottleneck strictly decreases the capacity of the resulting model, while unfortunately rendering the optimization problem non-convex. This method can be understood as a regularization technique, which typically improves the generalization performance of a trained model, as we will show in Section 4.4.

It is important to note that one can also replace a parameter matrix with a low-rank decomposition *after* training has completed, for example, using singular value decomposition [Xue *et al.*, 2013]. However, in the context of our work it was necessary to impose the low-rank decomposition *before* training, given our GPU memory constraints.

4.1.3 Entropy regularized perplexity (ERP)

Another method we leverage in order to improve the recognition performance of our models is to use a metric called “entropy regularized perplexity” (ERP) in order to decide when to decay our learning rate [Lu *et al.*, 2016]. ERP is defined as the sum of a model’s cross-entropy (CE) and its entropy (ENT), and is thus efficient to calculate. More formally, on a set of labeled points

$\{(x_1, y_1), \ldots, (x_N, y_N)\}$, the ERP is defined as follows:

$$\begin{aligned} ERP &= CE + ENT \\ &= -\frac{1}{N}\sum_{i=1}^{N} \log p(y_i|x_i) + -\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{C} p(y|x_i)\log p(y|x_i). \end{aligned}$$

This metric can be thought of as a more "lenient" version of the cross-entropy criterion, meaning that it doesn't penalize points on which the model assigns very low probability to the correct label as harshly as CE does. It accomplishes this by rewarding the model for giving confident answers (lower ENT), regardless of whether the answers are correct. Lu *et al.* demonstrate that on the large number of models they trained, there was a very high correlation between a model's heldout ERP and its development set token error rate (TER). In particular, it exhibited much higher correlation with TER than the heldout cross-entropy did. Thus, in our work we leverage this observation by calculating the ERP on our heldout sets at the end of every epoch of training, and using this to determine whether or not to decay our learning rate, as well as whether to terminate the training. In practice, this results in the training continuing past the point of lowest heldout cross-entropy, and producing models with lower heldout entropy (and lower ERP), but slightly higher cross-entropy. As we show in Section 4.4, using the ERP metric in this way generally leads to improved recognition performance relative to using the heldout cross-entropy.

This method shares the same goal as another set of methods known as *sequence training* techniques; in particular, the goal is to train the acoustic model in such a way that it performs better in terms of recognition performance (TER). There are a number of different sequence training criteria which have been proposed, including maximum mutual information (MMI) [Bahl *et al.*, 1986; Valtchev *et al.*, 1997], boosted MMI (BMMI) [Povey *et al.*, 2008], minimum phone error (MPE) [Povey and Woodland, 2002], or minimum Bayes risk (MBR) [Kaiser *et al.*, 2000; Gibson and Hain, 2006; Povey and Kingsbury, 2007]. These methods, though originally proposed for training Gaussian mixture model (GMM) acoustic models, can also be used for neural network acoustic models [Kingsbury, 2009; Veselý *et al.*, 2013]. Nonetheless, all of these methods are quite computationally expensive and are typically initialized with an acoustic model trained via the frame-level cross-entropy criterion. The ERP method, by contrast, is very simple, only making a small change to the frame-level training process. Furthermore, it can be used in conjunction with the above-mentioned sequence training techniques, by providing a better initial model. Recently, [Povey *et al.*, 2016] showed that it is possible to train an acoustic model using *only* sequence-training methods, with the lattice-free version of the MMI criterion. In a similar vein, the Connectionist Temporal Classification (CTC) method for acoustic model training directly models the conditional probabilities of *sequences* of labels, thus also eliminating the need for frame-level training altogether [Graves *et al.*, 2006]. For future work, we would like to see how much our kernel models can benefit from the various sequence training methods mentioned above, relative to DNNs.

4.2 Tasks, datasets, and evaluation metrics

We train both DNNs and kernel-based multinomial logistic regression models, as described in Section 4.1, to predict HMM state labels from acoustic feature vectors. We test these methods on four datasets. Each dataset is divided in four: a training set, a heldout set, a development set, and a test set. We use the heldout set to tune the hyperparameters of our training procedure (e.g., learning rate, kernel bandwidth). We then use the development set to select a small subset of models which perform best in terms of TER (e.g., the best kernel model, and the best DNN model, per dataset). Finally, we evaluate this select group of models on the test set, in order to get a fair comparison between the methods we are using. Having a separate development set helps us avoid the risk of over-fitting to the test set.

The first two datesets we use are the IARPA Babel Program Cantonese (IARPA-babel101-v0.4c) and Bengali (IARPA-babel103b-v0.4b) limited language packs. Each pack contains a 20-hour training set, a 20-hour development set, and a 30-hour test set. We designate about 10% of the training data as a heldout set. The training, heldout, development, and test sets all contain different speakers. Babel data is challenging because it is two-person conversations between people who know each other well (family and friends) recorded over telephone channels (in most cases with mobile telephones) from speakers in a wide variety of acoustic environments, including moving vehicles and public places. As a result, it contains many natural phenomena such as mispronunciations, disfluencies, laughter, rapid speech, background noise, and channel variability. An additional challenge in Babel is that the only data available for training language models is the acoustic transcripts, which are relatively small. The third dataset is a 50-hour subset of Broadcast News (BN-50) [Kingsbury, 2009; Sainath *et al.*, 2011; van den Berg *et al.*, 2017], which is a well-studied benchmark task in the ASR community. It has 45 hours of training data, and a 5 hour heldout set. For the development set, we use the "Dev04F" dataset provided by LDC, which consists of 2 hours of broadcast news from various new shows. We use the RT-03 Rich Transcription NIST benchmark test as our test set, consisting of 72 five minute conversations. The last dataset we use is TIMIT [Garofolo *et al.*, 1993], which contains recordings of 630 speakers, of various English dialects, each reciting ten sentences, for a total of 5.4 hours of speech. The training set (from which the heldout set is then taken) consists of data from 462 speakers each reciting 8 sentences (SI and SX sentences). The development set consists of speech from 50 speakers. For evaluation, we use the "core test set", which consists of 192 utterances total from 24 speakers (SA sentences are excluded). For reference, we use the exact same features, labels, and divisions of the dataset, as [Huang *et al.*, 2014] and [Chen *et al.*, 2016], which allows direct comparison of our results with theirs.

The acoustic features, representing 25 ms acoustic frames with context, are real-valued dense vectors. For the Cantonese, Bengali, and Broadcast News datasets we use a standard 360-dimensional speaker-adapted representation used by IBM [Kingsbury *et al.*, 2013]; these vectors are the concatenatation of nine 40-dimensional vectors, corresponding to features for the current frame, and the four frames before and after. The state labels are obtained via forced alignment using a GMM/HMM system. For the TIMIT experiments, we use 40 dimensional feature space maximum likelihood linear regression (fMLLR) features [Gales, 1998], and concatenate the 5 neighboring frames in either direction, for a total of 11 frames and 440 features.

The Cantonese and Bengali datasets each have 1000 labels, corresponding to quinphone context-dependent HMM states clustered using decision trees. For Broadcast News, there are 5000 such states. The TIMIT dataset has 147 context-independent labels, corresponding to the beginning, middle, and end of 49 phonemes.

For all datasets, the number of frames significantly exceeds typical machine learning tasks tackled by kernel methods. In particular our training sets all contain between 2 and 16 millions frames. Additionally, the large number of output classes for our datasets also presents a scalability challenge, given that the size of the kernel models scales linearly with the number of output classes (if no bottleneck is used). Table 4.1 provides details on the sizes of all the datasets (in terms of the number of acoustic frames), as well as on their number of features and classes.

We use five metrics to evaluate the acoustic models:

Dataset	Train	Heldout	Dev	# Features	# Classes
Beng.	7.7M	1.0M	7.1M	360	1000
BN-50	16M	1.8M	0.7M	360	5000
Cant.	7.5M	0.9M	7.2M	360	1000
TIMIT	2.3M	0.2M	0.1M	440	147

Table 4.1: Dataset details

1. **Cross-entropy (CE)**: Given examples, $\{(x_i, y_i), i = 1 \ldots N\}$, the cross-entropy is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}\log p(y_i|x_i). \tag{4.2}$$

2. **Average Entropy (ENT)**: The average entropy of a model is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{C} p(y|x_i)\log p(y|x_i)$$

   If a model has low average entropy, it is generally confident in its predictions.

3. **Entropy Regularized Perplexity (ERP)**: Defined in Section 4.1.3. Equal to $CE + ERP$.

4. **Classification Error (ERR)**: The classification error is defined as

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\left[y_i \neq \operatorname*{arg\,max}_{y\in 1,2,\ldots,C} p(y|x_i)\right].$$

5. **Token Error Rate (TER)**: Defined in Section 2.6. This metric measures the amount of errors made by the output of the ASR system, relative to the reference transcription of an utterance. For Bengali and BN-50, we measure the error in terms of the word error rate (WER), for Cantonese we use the character error rate (CER), and for TIMIT we use the phone error rate (PER). We use the term “token error rate” (TER) to refer, for each dataset, to its corresponding metric.

4.3 Details of acoustic model training

All our kernel models were trained with either the Laplacian or the Gaussian kernel. These kernel models typically have 3 hyperparameters: the kernel bandwidth (σ for the Gaussian kernels, λ for

	1000	2000	4000
3	17.5	16.8	16.7
4	17.1	**16.4**	16.5
5	16.9	16.5	16.7
6	17.0	16.5	16.6

Table 4.2: Effect of depth and width on DNN TER (development set): This table shows TER results for DNNs with 1000, 2000, or 4000 hidden units per layer, and 3-6 layers, on the Broadcast News development dataset. All of these models were trained using a linear bottleneck for the output parameter matrix, and using entropy regularized log loss for learning rate decay. The best result is in bold.

the Laplacian kernel; see Table 2.2), the number of random projections, and the initial learning rate of the optimization procedure. As a rule of thumb, we begin our search for a good setting of our kernel bandwidth parameter (specifically, $2\sigma^2$ for the Gaussian kernel, and $1/\lambda$ for the Laplacian kernel) at around the median of the pairwise distances in the data (squared ℓ_2 distance for Gaussian kernel, ℓ_1 distance for Laplacian kernel). We try various numbers of random features, ranging from $5k$ to $200k$. Using more random features leads to a better approximation of the kernel function, as well as to more powerful models, though there are diminishing returns as the number of features increases.

For all DNNs, we tune hyperparameters related to both the architecture and the optimization. This includes the number of layers, the number of hidden units in each layer, and the learning rate. We perform 1 epoch of layer-wise discriminative pre-training [Seide *et al.*, 2011b; Kingsbury *et al.*, 2013], and then train the entire network jointly using SGD. We find that 4 hidden layers is generally the best setting for our DNNs, so all the DNN results we present in this paper use this setting; in Table 4.2 we show how depth affects recognition performance on the Broadcast News dataset. As you can see, using more than 4 hidden layers does not improve performance. Additionally, all our DNNs use the $\tanh$ activation function. We vary the number of hidden units per layer (1000, 2000, or 4000). We use this same set of DNN architectures for all our datasets.

For both DNN and kernel models, we use stochastic gradient descent (SGD) as our optimization algorithm, with a mini-batch size of 250 or 256 samples, using the cross-entropy loss function

(Equation 4.2). We use the heldout set to tune the other hyperparameters (e.g., learning rate). We use the learning rate decay scheme described in [Morgan and Bourlard, 1990; Sainath *et al.*, 2013a; Sainath *et al.*, 2013c], which monitors cross-entropy performance on the heldout set in order to decide when to decay the learning rate. This method divides the learning rate in half at the end of an SGD epoch if the heldout cross-entropy doesn't improve by at least 1%; additionally, if the heldout cross-entropy gets *worse*, it reverts the model back to its state at the beginning of the epoch. In this work, we additionally experiment with using the heldout performance of the "Entropy Regularized Perplexity" (ERP) metric proposed by Lu et al. [Lu *et al.*, 2016], instead of the heldout cross-entropy, in order to determine learning rate decay; given the high correlation between heldout ERP and development set TER, this generally leads to better recognition performance than using cross-entropy.

We train models both with and without linear bottlenecks in the output matrix; the only exception is that we are unable to train BN50 kernel models *without* the bottleneck of size 1000 due to memory constrains on our GPUs. We use bottlenecks of size 1000, 250, 250, and 100 for BN50, Bengali, Cantonese, and TIMIT, respectively.

We initialize our DNN parameters uniformly at random in the range $[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, \frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}]$, as suggested by [Glorot and Bengio, 2010]; here, d_{in} and d_{out} refer to the dimensionality of the input and output of a DNN layer, respectively. For our kernel models, we initialize the random projection matrix as discussed in Section 2, and we initialize the output matrix as the zero matrix. When using a linear bottleneck to decompose the output matrix, we initialize the resulting two matrices randomly, as in [Glorot and Bengio, 2010].

All our training code is written in MATLAB, leveraging its GPU capabilities. We execute our code on Amazon EC2 machines, with instances of type g2.2xlarge. We use StarCluster[1] to more easily manage our clusters of EC2 machines.

4.4 Results

In this section, we report the results from our experiments comparing kernel methods to deep neural networks (DNNs) on ASR tasks. We report results on all 4 datasets. For both DNN and kernel

[1]`http://star.mit.edu/cluster`

methods, we train models with and without linear bottlenecks, and with and without using ERP to determine learning rate decay. For our kernel experiments, we use 100k random features on all datasets expect for TIMIT, where we are able to use 200k random features (because the output dimensionality is lower); we run experiments with both the Laplacian and the Gaussian kernels. For our DNN experiments, we train models with 4 hidden layers,[2] using the tanh activation function, and using either 1k, 2k, or 4k hidden units per layer. We focus on comparing the performance of these methods in terms of TER, but we also report results in terms of cross-entropy and classification error. Unless specified otherwise, all TER results are on the development set, and all cross-entropy, entropy, ERP, and classification error results are on the heldout set. In each table, the best result for each language is shown in bold.

In Tables 4.3 and 4.4, we show our TER results for our DNN and kernel models, respectively, across all datasets. There are many things to notice about these results. First of all, our best DNNs and kernels are tied on Cantonese and TIMIT, but the DNNs win by a relatively wide margin on both Bengali and Broadcast News (BN50). Within the kernel models, we see that incorporating a linear bottleneck brings large drops in TER across the board (recall that we are unable to train BN50 kernel models *without* using a bottleneck because the resulting models would not fit on our GPUs). Using the ERP metric to determine when to decay the learning rate also helps all our kernel models attain lower TER values. Typically, the Gaussian kernel, combined with these two methods, attains the lowest TER results (or close to it).

For our DNN models, linear bottlenecks almost always lower TER values, though in a few cases they have no effect on TER. Using ERP to determine when to decay the learning rate generally helps lower TER values for our DNNs, but in a few cases it actually hurts (Cantonese 4k, and TIMIT 2k and 4k). The DNNs with 4k hidden units typically attain the best results, though on a couple datasets they are matched or narrowly beaten by the 2k models.

In Table 4.5, for each dataset we compare the performance of the best DNN model with the best kernel model, across 5 metrics: cross-entropy (CE), entropy (ENT), entropy regularized perplexity (ERP), classification error (ERR), and Token Error Rate (TER). In terms of cross-entropy and ERP, the DNNs outperform the kernels on all datasets except TIMIT. On all datasets the DNNs were more confident in their predictions (lower entropy) than the kernels. In terms of classification error,

[2]As mentioned in Section 4.3, and shown in Table 4.2, we find that this is generally the best setting.

	1k				2k				4k			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	72.3	71.6	71.7	70.9	71.5	71.1	70.7	70.3	71.1	70.6	70.5	**70.2**
BN-50	18.0	17.3	17.8	17.1	17.4	16.7	17.1	**16.4**	16.8	16.7	16.7	16.5
Cant.	68.4	68.1	67.9	67.5	67.7	67.7	67.2	**67.1**	67.7	**67.1**	67.2	67.2
TIMIT	19.5	19.3	19.4	19.2	19.0	18.9	19.2	19.2	**18.6**	**18.6**	18.7	18.9

Table 4.3: DNN TER Results (development set): ‘B’ specifies that a linear bottleneck is used, ‘R’ specifies that ERP is used (‘BR’ means both are used), and ‘NT’ signifies that neither are used.

	Laplacian				Gaussian			
	NT	B	R	BR	NT	B	R	BR
Beng.	74.5	72.1	74.5	**71.4**	72.6	72.0	72.6	71.8
BN-50	N/A	17.9	N/A	17.7	N/A	17.3	N/A	**17.1**
Cant.	69.9	68.2	69.2	67.4	70.2	67.6	70.0	**67.1**
TIMIT	20.6	19.2	20.4	18.9	19.8	18.9	19.6	**18.6**

Table 4.4: Kernel TER Results (development set): ‘B’ specifies that a linear bottleneck is used, ‘R’ specifies that ERP is used (‘BR’ means both are used), and ‘NT’ signifies that neither are used. TIMIT models use $200k$ random features, and all others use $100k$ features.

	Beng (D/K)	BN50 (D/K)	Cant (D/K)	TIMIT (D/K)
CE	**1.243** / 1.315	**2.001** / 2.052	**1.916** / 1.931	1.056 / **0.9423**
ENT	**0.9079** / 1.082	**1.274** / 1.457	**1.375** / 1.516	**0.447** / 0.6076
ERP	**2.302** / 2.473	**3.548** / 3.691	**3.459** / 3.556	1.671 / **1.648**
ERR	**0.2887** / 0.3041	**0.4887** / 0.501	0.4353 / **0.4342**	0.324 / **0.3148**
TER	**70.2** / 71.4	**16.4** / 17.1	**67.1** / **67.1**	**18.6** / **18.6**

Table 4.5: Table of Best DNN vs. Kernel results, across 4 datasets and 5 metrics.

the DNNs did better on Bengali and BN50, while the kernels did better on Cantonese and TIMIT. Overall, the DNNs did slightly better than the kernels, though kernels were typically not far behind. We include more detailed results in Appendix C.

We will now illustrate the importance of the number of random features D on the final performance of the model. For this purpose, we trained a number of different models on the BN50 dataset, using $D \in \{5k, 10k, 25k, 50k, 100k\}$. We trained models using both kernels. We used a linear bottleneck of size 1000 for all these models, and used heldout cross-entropy to determine the learning rate decay. In Figure 4.2, we show how increasing the number of features dramatically improves the performance of the learned model, both in terms of cross-entropy and TER; there are diminishing returns, however, with small improvements in TER when increasing D from $50k$ to $100k$. This shows that in order to attain strong performance with these kernel approximation methods, it is very important to use a very large number of features.

4.5 Other Possible Improvements to DNNs and Kernels

It is important to mention a few things regarding other ways the performance of our DNN and kernel models could be improved, and why they are not investigated at length in this work. For the kernel methods, given that the optimization is convex when no bottleneck is used, it would be possible to get stronger convergence guarantees using the Stochastic Average Gradient (SAG) algorithm instead of SGD for training [Le Roux *et al.*, 2012]. In fact, in [Lu *et al.*, 2016] we did this on Cantonese and

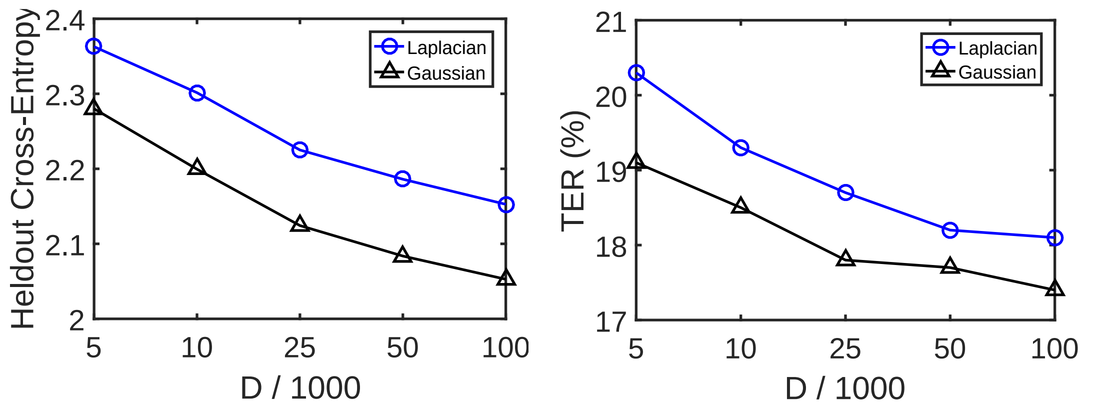

Figure 4.2: Performance of kernel acoustic models on BN50 dataset, as a function of the number of random features D used. Results are reported in terms of heldout cross-entropy as well as development set TER. The color and shape of the markers indicate the kernel used.

Bengali, and attained strong recognition performance.[3] Unfortunately, it is challenging to scale this algorithm to larger tasks, since it requires storing, for every training example, the previous gradient of the loss function at that example. Because $\frac{\partial L(x_i,y_i)}{\partial \boldsymbol{\theta}_y} = z(\boldsymbol{x}_i)[\mathbb{I}(y = y_i) - p(y|\boldsymbol{x}_i)]$, and because $z(x_i)$ is fixed, the gradient information can be stored by simply storing, for each training example, the vector $\boldsymbol{p}_i = [p(1|\boldsymbol{x}_i), \ldots, p(C|\boldsymbol{x}_i)]$. However, this still takes NC storage, which is quite expensive when there are millions of training examples N and thousands of output classes C (320 GB for the Broadcast News dataset, for example). Unfortunately, once a bottleneck is introduced, not only is the optimization problem non-convex, but we must also store the full gradients, thus making the memory requirement too large. As a result, for scalability reasons, as well as for consistency across all our experiments, we have used SGD for all our kernel experiments. Additionally, we did not investigate the use of sequence training techniques for our kernel methods, leaving this for future work.

For our DNN models, we have observed that restricted Boltzmann machine (RBM) pre-training [Hinton *et al.*, 2006] often improves recognition performance [Lu *et al.*, 2016]. Additionally, as

[3]A few more details regarding the experiments in [Lu *et al.*, 2016]: we did not use feature selection in that work, and we only used ERP as a model selection criterion (not for learning rate decay). Additionally, instead of training the large kernel models jointly, we trained them in blocks of 25,000 random features, and then combined the models via logit averaging (final models had 200,000 random features).

discussed in the introduction, there are various other deep architectures (e.g., CNNs, LSTMs), as well as numerous training techniques (e.g., momentum [Sutskever *et al.*, 2013], dropout [Srivastava *et al.*, 2014], batch normalization [Ioffe and Szegedy, 2015]), which can further improve the performance of neural networks. However, as we have mentioned, our goal for this paper was to provide a comparison between kernel methods and a strong DNN baseline (DNN with $\tanh$ activation, and discriminative pre-training), not to build a state-of-the-art speech recognition system, or to provide an exhaustive comparison against all possible deep learning architectures and optimization methods.

4.6 Conclusion

In this chapter, we have shown that kernel approximation methods can be scaled to large scale acoustic modeling tasks, using a simple multinomial logistic regression model, trained with SGD. We demonstrated across four datasets that the performance of these kernel models is near that of DNNs, matching their performance on two datasets, and performing slightly worse on the other two. We showed that using a linear bottleneck can speed up training, reduce the number of parameters needed, and significantly improve the performance of a model. We have also shown that using held-out ERP for deciding learning rate decay is an effective and cheap way of improving the recognition performance of a model.

These kernel models require a very large number of features in order to attain strong performance. We see that even at $100k$ features, there are marginal gains from increasing the number of features. In the next chapter, we explore how to reduce the number of features required to attain a given level of performance, using a feature selection method.

Chapter 5

Compact kernel models via random feature selection

In this chapter, we present a feature selection algorithm which provides significant performance gains to kernel models under a fixed memory budget. Our algorithm is iterative, at each step searching through large numbers of random features, selecting a subset, and discarding the rest. We begin by describing and motivating our proposed feature selection algorithm in Section 5.1. We then describe a new "sparse Gaussian kernel" in Section 5.2, which behaves nicely in conjunction with the feature selection algorithm. In Section 5.3 we present experimental results demonstrating the effectiveness of this method on ASR, and comparing performance with fully-connected feedforward DNNs. We show that using our proposed feature selection algorithm, along with the methods presented in Chapter 4, we are able to perform on par with DNNs. We then discuss the dynamics of the feature selection process in Section 5.4, showing that features that are selected in early iterations typically survive all remaining rounds. We conclude in Section 5.5.

5.1 Random feature selection

Our proposed feature selection method, shown in Algorithm 1, is iterative; at each step, a subset of random features are selected from a pool, while the rest are discarded and replaced with new

Algorithm 1 Random feature selection

```
input  Target number of random features $D$, data subset size $R$,
       selection schedule $0 = s_0 < s_1 < \cdots < s_T = D$.
1: initialize feature pool $P := \emptyset$.
2: for $t = 1, 2, \ldots, T$ do
3:     Generate $D - s_{t-1}$ new random features, and add them to $P$.
4:     Learn weights $W \in \mathbb{R}^{P \times C}$ over the $D$ features in $P$ using a single pass of SGD over $R$
       randomly selected training examples.
5:     Select $s_t$ features $j \in P$ for which $\sum_{c=1}^{C}(W_{j,c})^2$ are largest; discard the remaining $D - s_t$.
6: end for
7: return  Final collection of $D$ random features $P$.
```

random features. The selection criterion is based on a feature's weights,[1] which are learned using stochastic gradient descent (SGD).

This method has the following advantages: The overall computational cost is mild, as it requires just T passes through subsets of the data of size R (equivalent to $\approx TR/n$ full SGD epochs). In fact, in our experiments, we find it sufficient to use $R = O(D)$. Note that this is less computationally demanding than fully solving an ℓ_1-regularized optimization problem, as in the Sparse Random Features method of [Yen *et al.*, 2014]. Moreover, the method is able to explore a large number of non-linear features, while maintaining a compact model. If $s_t = Dt/T$, then the learning algorithm is exposed to roughly $DT/2$ random features throughout the feature selection process. For example, if $T = 50$ and $D = 100k$, this algorithm considers around 2.5 million features; this is a typical setting for our experiments. We show in Section 5.3 that this empirically increases the predictive quality of the selected features.

It is important to note the similarities between this method, and the FOBOS method with ℓ_1/ℓ_2-regularization [Duchi and Singer, 2009]. In the latter method, one solves the ℓ_1/ℓ_2-regularized problem in a stochastic fashion by alternating between taking unregularized stochastic gradient descent (SGD) steps, and then "shrinking" the rows of the parameter matrix; each time the parameters are shrunk, the rows whose ℓ_2-norm is below a threshold are set to 0. After training completes, the

[1]The weights corresponding to the feature $z_i(\boldsymbol{x})$ in the model $f(\boldsymbol{x}) = W^{\mathrm{T}} z(\boldsymbol{x})$ are those in the i^{th} row of W.

solution will likely have some rows which are all zero, at which point the features corresponding to those rows can be discarded. In our method, on the other hand, we take many consecutive unregularized SGD steps, and only thereafter do we choose to discard the rows whose ℓ_2-norm is below a threshold.[2] As mentioned in the Related Work section, our attempts at using FOBOS for feature selection failed, because the amount of regularization needed in order to produce a sparse model was so strong that it dominated the learning process; as a result, the models learned performed terribly, and the selected features were essentially random.

One disadvantage of our method is that the index used for selection may misrepresent the features' actual predictive utilities. For instance, the presence of some random feature $i \in P$ may increase or decrease the weights for other random features relative to what it would be if $i \notin P$. An alternative would be to consider features in isolation, and add features one at a time (as in stagewise regression methods and boosting), but this would require many passes through the data, which is expensive due to the I/O overhead of each pass. We find empirically that the influence of the additional random features in the selection criterion is tolerable, and it is still possible to select useful features with this method.

5.2 A sparse Gaussian kernel

In Section 5.3 we will show that when we perform feature selection on models using the Laplacian kernel, we see much larger improvements in performance than for models using the Gaussian kernel. This leads us to study what might be the cause of this large difference. In particular, we study the differences in the sampling distributions used to approximate each of these kernels. Recall from Table 2.2 that for the Laplacian kernel, the sampling distribution used for the random Fourier features is the multivariate Cauchy density $p(\omega) \propto \prod_{i=1}^{d}(1+\omega_i^2)^{-1}$ (we let $\lambda = 1$ here for simplicity). If we draw $\omega = (\omega_1, \dots, \omega_d)$ from p, then each ω_i has a two-sided fat tail distribution, and hence ω will typically contain some entries much larger than the rest.

This property of the sampling distribution implies that many of the random features generated in this way will each effectively concentrate on a few of the input features. We can thus regard

[2]Note that if we are using a linear bottleneck to decompose W into $U \cdot V$, we first compute $W = UV$, and then select features based on the ℓ_2-norm of the rows of W

each such random feature as being a non-linear combination of a small number of the original input features. Thus, the feature selection method is effectively picking out useful non-linear interactions between small sets of input features.

We can also directly construct sparse non-linear combinations of the input features. Instead of relying on the properties of the Cauchy distribution, we can choose a small number k of coordinates $F \subseteq \{1, 2, \ldots, d\}$, say, uniformly at random, and then choose the random vector ω so that it is always zero in positions outside of F; the same non-linearity (e.g., $x \mapsto \cos(\omega^{\mathrm{T}}x+b)$) can be applied once the sparse random vector is chosen. Compared to the random Fourier feature approximation to the Laplacian kernel, the vectors ω chosen this way are truly sparse, which can make the random feature expansion more computationally efficient to apply (if efficient sparse matrix operations are used).

Note that random Fourier features with such sparse sampling distributions in fact correspond to shift-invariant kernels that are rather different from the Laplacian kernel. For instance, if the non-zero entries of ω are chosen as i.i.d. $\mathcal{N}(0, \sigma^{-2})$, then the corresponding shift-invariant kernel is

$$k(x, x') = \sum_{F \subseteq \{1,2,\ldots,d\}:|F|=k} \binom{d}{k}^{-1} \prod_{i \in F} \exp\left(-\frac{(x_i - x'_i)^2}{2\sigma^2}\right). \tag{5.1}$$

The kernel in Eq. (5.1) puts equal emphasis on all input feature subsets F of size k. However, the feature selection may effectively bias the distribution of the feature subsets to concentrate on some smaller family of input feature subsets. We call this kernel the *sparse Gaussian kernel*.

5.3 Results

The results in this section build on those from Section 4.4. In particular, we perform feature selection as a first step prior to training our kernel models, and demonstrate that this improves performance. We also show results for the sparse Gaussian kernel, with and without using feature selection.

Regarding implementation details and hyperparameter choices: For each iteration of random feature selection, we draw a random subsample of the training data of size $R = 10^6$ (except when $D \geq 10^5$, in which case we use $R = 2\times10^6$, to ensure a safe R to D ratio). Thus, each iteration of feature selection has equivalent computational cost to a R/N fraction of an SGD epoch, where N is the total number of training points. For example, on the Broadcast News dataset, this corresponds to roughly $1/16$ or $1/8$ of an epoch, for $D < 10^5$ and $D \geq 10^5$, respectively. We use $T = 50$ iterations of feature selection, and in iteration t, we select $s_t = t \cdot (D/T) = 0.02Dt$ random features. Thus, the total computational cost we incur for feature selection is equivalent to approximately three or six epochs of training on Broadcast News.[3] After feature selection completes, we train the model using the features which have been selected. For all experiments with the sparse Gaussian kernel, we use $k = 5$.

In Table 5.1 we show the performance of our kernel models, using all three kernels (Laplacian, Gaussian, sparse Gaussian), with and without feature selection. All of these models use $100k$ random features, except for the TIMIT models, which use $200k$ random features. There are several things to notice about these results. First of all, we see that performing feature selection improves TER considerably for the Laplacian kernel, and modestly for the sparse Gaussian kernel. For the Gaussian kernel, it typically helps, though there are several instances in which feature selection hurts TER (see Section 5.4 for discussion). Secondly, we see that without using feature selection, the sparse Gaussian kernel has the best performance across the board. After we include feature selection, it performs very comparably to the Laplacian kernel with feature selection. In general, the kernel models which perform best are the Laplacian and sparse Gaussian kernels with feature selection (as well as with bottleneck and ERP). It is interesting to note that without using feature selection, the Gaussian kernel is generally better than the Laplacian kernel; however, with feature selection, the Laplacian kernel surpasses the Gaussian kernel (see Section 5.4). In general, the kernel function which performed best, across the majority of settings, was the sparse Gaussian kernel.

In order to see whether we could attain *even stronger* performance with our kernel models by using more random features, we trained a small number of models with up to $400k$ features on Broadcast News, our most challenging dataset. Due to the large computational expense of training these models, we only trained a few, and only used the Laplacian and sparse Gaussian kernels, as these attained the best performance after feature selection, in terms of TER. We report results in Table 5.2. All of these models were trained with a linear bottleneck of size 1000, and using ERP for learning rate decay. We include results using $100k$ random features in this table for comparison. As you can see, our best kernel model on BN-50 now attains a TER of 16.4%, which if you recall

[3]It is possible that feature selection could remain effective using much smaller random subsamples of the training set. This would make the feature selection process a lot faster, though this is not something we carefully tested.

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	74.5	72.1	74.5	71.4	72.6	72.0	72.6	71.8	73.0	71.5	73.0	**70.9**
+FS	72.9	71.1	72.8	70.4	74.1	71.4	74.2	**70.3**	72.9	71.2	72.8	70.7
BN-50	N/A	17.9	N/A	17.7	N/A	17.3	N/A	17.1	N/A	17.3	N/A	**17.0**
+FS	N/A	17.1	N/A	**16.7**	N/A	17.5	N/A	17.0	N/A	17.1	N/A	**16.7**
Cant.	69.9	68.2	69.2	67.4	70.2	67.6	70.0	**67.1**	68.6	67.5	68.1	**67.1**
+FS	68.4	67.5	68.5	**66.7**	69.9	67.7	69.8	66.9	68.6	67.4	68.5	66.8
TIMIT	20.6	19.2	20.4	18.9	19.8	18.9	19.6	18.6	19.9	18.8	19.6	**18.4**
+FS	19.5	18.6	19.3	18.4	19.5	18.6	19.4	18.4	19.3	18.4	19.1	**18.2**

Table 5.1: Kernel TER Results (development set):‘B’ specifies that a linear bottleneck is used, ‘R’ specifies that ERP is used (‘BR’ means both are used), and ‘NT’ signifies that neither are used. ‘+FS’ specifies that feature selection was used for the experiments in that row. TIMIT models use $200k$ random features, and all others use $100k$ features.

from Section 4.4 is equal to the performance of our best DNN model. Furthermore, we continue to see improvements in the performance of our kernel models, even as we increase our number of features beyond $100k$; for the Laplacian kernel we get a gain of 0.3% in TER when increasing from $100k$ to $400k$ (both with and without feature selection), while for the sparse Gaussian kernel we get gains of 0.2% and 0.4%, with and without feature selection (respectively). To date, these are the largest models we have trained, though it seems likely we could continue getting performance improvements with even larger models.

In Table 5.3, we compare for each dataset the performance of the best DNN model with the best kernel model, across 6 metrics. Importantly, for each metric (except for test set TER), we find the kernel and DNN model which performs best for that specific metric; this is in contrast to picking the kernel and DNN models which are best in terms of TER, and reporting all metrics on these models. For Broadcast News, we consider the kernel models from Table 5.2 with $200k$ and $400k$ random features, along with those in Table 5.1, in the process of finding the best performing models. In terms of heldout cross-entropy, the kernels outperformed the DNNs on Cantonese and TIMIT,

	$100k$	$200k$	$400k$
Laplacian	17.7	17.7	17.4
+FS	16.7	**16.4**	**16.4**
Sparse Gaussian	17.0	16.8	16.6
+FS	16.7	16.6	**16.5**

Table 5.2: Kernel TER Results on Broadcast News development set for models with a very large number of random feature (up to $400k$). All models use a bottleneck of size 1000, and use ERP for learning rate decay.

while the DNNs beat the kernels on Bengali and BN-50. With regard to classification error, the kernels beat the DNNs on all datasets except for Bengali. In terms of the average heldout entropy of the models, the DNNs were consistently more confident in their predictions (lower entropy) than the kernels. Significantly, we observe that the best development set TER results for our DNN and kernel models are quite comparable; on Cantonese and TIMIT, the kernel models outperform the DNNs by 0.4% absolute, on Broadcast News the kernels exactly match the DNNs, while on Bengali the DNNs do better by 0.1%.

We will now discuss the results on the test sets. First of all, in order to avoid overfitting to the test sets, for each dataset we only performed test set evaluations for the DNN and kernel models which performed best in terms of the development set TER. The final row of Table 5.3 thus contains all the test results we collected.[4] As one can see, the relative performance of the DNN and kernel models is quite similar to the development set results; the DNNs perform slightly better (0.1%) on Bengali, and the kernels perform better on the rest, winning by 0.1% on Broadcast News, 0.5% on Cantonese, and 0.1% on TIMIT. For direct comparison on the TIMIT dataset, we include in Table 5.4 the test results for the best DNN and kernel models from [Huang *et al.*, 2014] and [Chen *et al.*, 2016]. As mentioned in Section 4.2, we use the same features, labels, data set partitions (train/heldout/dev/test), and decoding script as these papers, and thus our results are directly com-

[4]The only exception for this is on Broadcast News. We had already evaluated the best model using $100k$ random features before we decided to train models with more random features (Table 5.2). Thus, in Table 5.3, we report the result for the $400k$ Laplacian model, which attained a TER of 11.6%, whereas the $100k$ Laplacian model attained a TER of 11.9% on the test set.

	Beng. (D/K)	BN-50 (D/K)	Cant. (D/K)	TIMIT (D/K)
CE	**1.243** / 1.256	**2.001** / 2.004	1.916 / **1.883**	1.056 / **0.9182**
ENT	**0.9079** / 1.082	**1.274** / 1.434	**1.375** / 1.516	**0.447** / 0.5756
ERP	**2.302** / 2.406	**3.548** / 3.552	**3.459** / 3.493	1.671 / **1.607**
ERR	**0.2887** / 0.2936	0.4887 / **0.4881**	0.4353 / **0.4287**	0.324 / **0.3085**
TER (dev)	**70.2** / 70.3	**16.4** / **16.4**	67.1 / **66.7**	18.6 / **18.2**
TER (test)	**69.1** / 69.2	11.7 / **11.6**	63.7 / **63.2**	20.5 / **20.4**

Table 5.3: Table comparing the Best DNN ('D') and kernel ('K') results, across 4 datasets and 6 metrics. The first 4 metrics are on the heldout set, the fifth is on the development set, and the last metric is reported on the test set. For BN50, the large models from Table 5.2 are included in the set of models from which the best performing model is picked (for each metric). See Section 4.2 for metric definitions.

	Test TER (DNN)	Test TER (Kernel)
[Huang *et al.*, 2014]	20.5	21.3
[Chen *et al.*, 2016]	N/A	20.9
This work	20.5	**20.4**

Table 5.4: Table comparing the Best DNN and kernel results from this work to those from [Huang *et al.*, 2014] and [Chen *et al.*, 2016], on the TIMIT test set.

parable. We achieve a 0.9% absolute improvement in TER with our kernel model relative to [Huang *et al.*, 2014], and 0.5% improvement relative to [Chen *et al.*, 2016]; furthermore, our best DNN matches the performance of the best performing DNN from [Huang *et al.*, 2014]. We believe the most significant of these results is that the kernels (narrowly) beat the DNNs on Broadcast News, our largest and most challenging dataset, and one which has been used extensively in large scale speech recognition research.

In Appendix C, we include more detailed tables comparing the various models we trained across these metrics. Some important things to take note of in those tables are as follows:

- The linear bottleneck typically causes a large drop in the average entropy of kernel models, while not having as strong or consistent an effect on cross-entropy. For DNNs, the bottleneck typically causes increases in cross-entropy, and relatively modest decreases in entropy.

- Using entropy regularized perplexity (ERP) to determine learning rate decay typically causes increases in cross-entropy, and decreases in entropy, with the decrease in entropy typically being larger than the increase in cross-entropy. As a result, the ERP is typically lower for models that use this trick (with the exception of TIMIT DNN models).

- Feature selection typically results in large drops in cross-entropy, especially for Laplacian and sparse Gaussian kernels, while its effect on entropy is quite small. It thus helps lower ERP across the board, as well as TER in the vast majority of cases.

In Figure 5.1, we explore the performance of our three kernels, with and without feature selection, as we vary the number of random Fourier features $D \in \{5k, 10k, 25k, 50k, 100k\}$ used for training. As in Figure 4.2, we consider BN50 models which used heldout cross-entropy to determine the learning rate decay. Once again, we see that increasing the number of features leads to stronger performance, both in terms of cross-entropy and TER. Furthermore, the size of the gap between the dashed and solid lines (representing experiments with and without feature selection, respectively), indicates the importance of feature selection in attaining strong performance. This gap is very large for the Laplacian kernel, modest for the Sparse Gaussian kernel, and relatively insignificant for the Gaussian kernel. Across different values for D, without feature selection, the Sparse Gaussian kernel typically does best; once feature selection is used, the Laplacian kernel and the sparse Gaussian kernel perform similarly, beating the performance of the Gaussian kernel.

5.4 Analysis: Effects of random feature selection

We now explore the dynamics of the feature selection process. In our method, there is no guarantee that a feature selected in one iteration will be selected in the next. In Figure 5.2, we plot the fraction of the s_t features selected in iteration t that actually remain in the model after all T iterations. We only show the results for Cantonese (models without linear bottleneck, and without using entropy regularized log loss for LR decay), as the plots for other datasets are qualitatively similar. In nearly all iterations and for all kernels, over half of the selected features survive to the final model. For

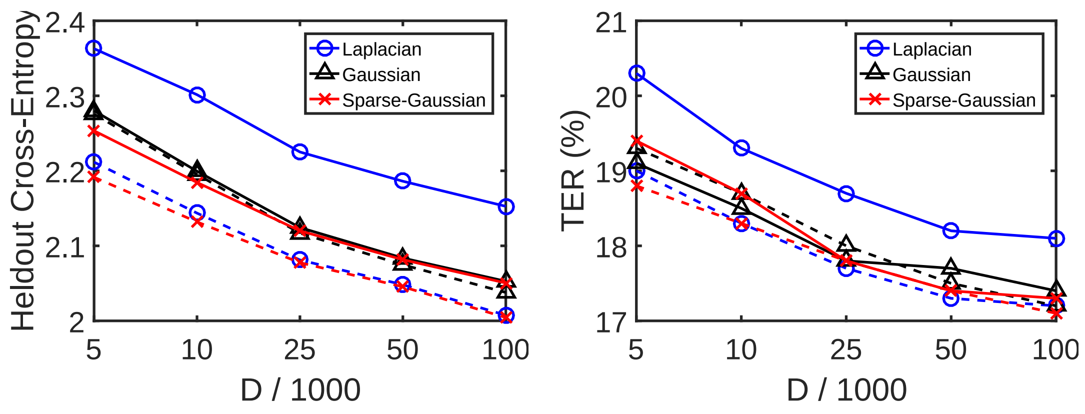

Figure 5.1: Performance of kernel acoustic models on BN50 dataset, as a function of the number of random features D used. Results are reported in terms of heldout cross-entropy as well as development set TER. Dashed lines signify that feature selection was performed, while solid lines mean it was not. The color and shape of the markers indicate the kernel used.

instance, over 90% of the Laplacian kernel features selected at iteration 10 survive the remaining 40 rounds of selection. For comparison, we also plot the expected fraction of the s_t features selected in iteration t that would survive until the end if the selected features in each iteration were chosen uniformly at random from the pool. Since we use $s_t = Dt/T$, the expected fraction at iteration t is $T!/(t! \cdot T^{T-t})$, which is exponentially small in T when $t \leq \beta T$ for any fixed $\beta < 1$.[5] For example, at $t = 10$ the expected survival rate is approximately 9×10^{-11} with $T = 50$.

Finally, we consider how the random feature selection process can be regarded as selecting non-linear combinations of input features. Consider the final matrix of random vectors $\Theta = [\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(D)}] \in \mathbb{R}^{d \times D}$ after random feature selection. A coarse measure of how much influence an input feature $i \in \{1, 2, \ldots, d\}$ has in the final feature map is the relative "weight" of the i-th row of Θ. In Figure 5.3, we plot $\frac{1}{Z} \sum_{j=1}^{D} |\Theta_{i,j}|$ for each input feature $i \in \{1, 2, \ldots, d\}$. Here, $Z = \frac{1}{d} \sum_{i,j} |\Theta_{i,j}|$ is a normalization term.[6] There is a strong periodic effect as a function of the input feature number. The reason for this stems from the way the acoustic features are generated. Recall that the features are the concatenation of nine 40-dimensional acoustic feature vectors cor-

[5]This can be shown using Stirling's formula. See [Jameson, 2015] for a useful review.

[6]For the Laplacian kernel, we discard the largest element in each of the d rows of Θ, because there are sometimes outliers which dominate the entire sum for their row.

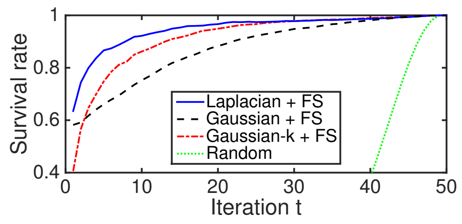

Figure 5.2: Fraction of the s_t features selected in iteration t that are in the final model (survival rate) for Cantonese dataset.

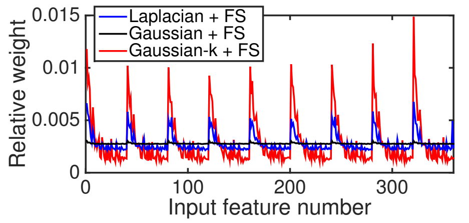

Figure 5.3: The relative weight of each input feature in the random matrix Θ, for Cantonese dataset, $D = 50{,}000$.

responding to nine audio frames. An examination of the feature pipeline from [Kingsbury *et al.*, 2013] reveals that these 40 features are ordered by a measure of discriminative quality (via linear discriminant analysis). Thus, it is expected that the features with low $(i-1) \bmod 40$ value may be more useful than the others; indeed, this is evident in the plot. Note that this effect exists, but is extremely weak, with the Gaussian kernel. We believe this is because Gaussian random vectors in $\mathbb{R}^d$ are likely to have *all* their entries be bounded in magnitude by $O(\sqrt{\log(d)})$.

5.5 Conclusion

In this chapter, we have presented a novel feature selection algorithm, which scales effectively to problems with millions of points and thousands of labels. We have additionally introduced a sparse Gaussian kernel, which works well in conjunction with the feature selection algorithm, and generally attains our strongest results. We showed that using these methods, we are able to perform on par with fully-connected feedforward DNNs across four acoustic modeling tasks. This is the first time that kernel approximation methods have been shown to be competitive with deep learning methods in a large scale domain in which DNNs are known to dominate.

Chapter 6

Nyström method vs. random Fourier features

In this chapter, we study the relative merits of the random Fourier features (RFF) method [Rahimi and Recht, 2007] and the Nyström method [Williams and Seeger, 2001], in terms of kernel approximation error, efficiency, and classification accuracy. We describe experiments on eight datasets that show the following phenomena: (1) The Nyström method gives *much* better kernel approximation error than random Fourier features, given the same number of features. At first glance this would suggest that Nyström features are preferable. (2) The computational overhead for Nyström, both in terms of computation time and in terms of memory, relative to RFF, results in the kernel approximation performance gap between the two methods shrinking considerably when a fixed computational budget is used for the two methods. (3) Perhaps most strikingly, the superior kernel approximation performance of the Nyström method does not directly translate to improved classification performance. Specifically, we observe that an RFF model will perform significantly better than a Nyström model with comparable average kernel approximation error, or comparable memory requirements.

This final point leads us to study how Nyström and RFF methods differ in the nature of their kernel approximation errors. Our finding, supported by both empirical and theoretical results, is that the Nyström method is much noisier at approximating the kernel value between points with large kernel value, and in fact systematically *underestimates* kernel values between these points. Given that training points of high kernel value with a data point have disproportionate influence on the output of a model, our work suggests that these errors are particularly costly, leading to inferior classification performance for the Nyström method.

This chapter is organized as follows: In Section 6.1 we review some important mathematical properties of the Nyström method. In Section 6.2 we discuss our extensive experiments, across eight datasets, comparing these two methods. We then discuss a number of theoretical results about the Nyström representations in Section 6.3, which further elucidate the important differences between them and random Fourier features. We conclude in Section 6.4.

6.1 Review of Nyström method properties

In this section, we review the fact that the Nyström method can be understood as a projection onto a subspace of the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$, and discuss some important implications of this fact. For a more complete mathematical presentation of this material, including proofs, please see Appendix D. Some related results are presented in Section 10.2 of Schölkopf and Smola [2002].

Claim 1. *Let $\varphi(x) \in \mathcal{H}$ denote the element in the RKHS $\mathcal{H}$ corresponding to a point $x \in \mathcal{X}$. The Nyström representation $z(x)$ for x, given the set of landmark points $\{\hat{x}_1, \ldots, \hat{x}_m\}$, corresponds to the projection of $\varphi(x)$ onto the subspace $A = span(\varphi(\hat{x}_1), \ldots, \varphi(\hat{x}_m)) \subset \mathcal{H}$.*

This result allows us to understand at a more fundamental level how the Nyström method is approximating the kernel; it is projecting the (possibly) infinite dimensional feature representation $\varphi(x) \in \mathcal{H}$ onto a subspace of the RKHS. Specifically, the subspace onto which all points are projected is the subspace in $\mathcal{H}$ spanned by the landmark points' representations. This suggests that for some points this projection will be a faithful representation, while for others it will not be. Specifically, points x for which $\varphi(x)$ is close to the subspace A should be well represented by their Nyström features, while points for which $\varphi(x)$ is far from A will not be. We make this more formal in the corollaries below:

Corollary 1.1. *Let $A^{\perp}$ denote the orthogonal complement to A in $\mathcal{H}$, and let $\varphi_B(x)$ denote the projection of $\varphi(x)$ onto B, for $B \in \{A, A^{\perp}\}$. Then the error made by the Nyström method in predicting $k(x, y)$ can be written as follows:*

$$k(x, y) - \langle z(x), z(y) \rangle = \langle \varphi_{A^{\perp}}(x), \varphi_{A^{\perp}}(y) \rangle_{\mathcal{H}}$$

Corollary 1.2. $\|z(x)\|^2 \leq k(x,x)$.

Corollary 1.3. *If $\varphi(x) \in A$, then the Nyström method makes 0 error approximating $k(x,y)\,\forall y \in \mathcal{X}$.*

In Corollary 1.1, we show that the error made by Nyström in approximating the kernel between two points is *precisely* equal to the dot-product between the components of those points orthogonal to the subspace A. Because Nyström is performing a projection, this implies that for all points x, the norm of $z(x)$ will be no larger than the norm of $\varphi(x)$; this means that the Nyström method will always *underestimate* the value of $k(x,x) = \|\varphi(x)\|_{\mathcal{H}}^2$, which is what we show in Corollary 1.2. Lastly, Corollary 1.3 says that for any point x such that $\varphi(x) \in A$, the Nyström representation for x will *perfectly* approximate the kernel between x and all other points. In particular, this guarantees that for a landmark point $\hat{x}_i$, the Nyström method will perfectly approximate $k(\hat{x}_i, x)$ for any $x \in \mathcal{X}$.

We include proofs of Claim 1 and each of these corollaries in Appendix D.

6.2 Experiments

In this section, we compare the performance of random Fourier features and Nyström features, along a number of metrics, including kernel approximation error, efficiency, and performance on classification and regression tasks, on eight datasets. We show that while the Nyström method does perform much better in terms of kernel approximation error for a fixed number of features, this does not translate directly into an equally large gap in classification and regression performance. We go on to argue that this discrepancy can be explained by the important differences in the ways these two methods make approximation errors.

6.2.1 Task and dataset details

For this work, we present experiments on various regression and classification tasks, summarized in Table 6.1. For all datasets besides TIMIT, we pre-processed the features and labels as follows: We normalized all continuous features to have zero mean and unit variance. We did not normalize the binary features in any way. For regression datasets, we normalized the labels to have zero mean across the training set. For more details on the datasets, see Appendix D.1. Notably, one of our datasets is the TIMIT dataset, which is quite a challenging and large-scale dataset relative to

Dataset	**Task**	**Train**	**Heldout**	**Test**	**#Attr.**
ADULT	Class. (2)	29k	3k	16k	123
COD-RNA	Class. (2)	54k	6k	272k	8
COVTYPE	Class. (2)	418k	46k	116k	54
FOREST	Class. (2)	470k	52k	58k	54
TIMIT	Class. (147)	2.3M	245k	116k	440
CENSUS	Reg.	16k	2k	2k	119
CPU	Reg.	6k	0.7k	0.8k	21
YEARPRED	Reg.	417k	46k	52k	90

Table 6.1: Dataset details. For classification tasks, we write the number of classes in parentheses in the "task" column.

the datasets on which these methods have been compared before. For all experiments, we use the Gaussian kernel, which corresponds to $k(x, y) = \exp(\frac{-\|x-y\|_2^2}{2\sigma^2})$.

6.2.2 Train details

For the binary classification tasks we train logistic regression models on top of either Nyström features or random Fourier features, using stochastic gradient descent (SGD). For the TIMIT dataset, which is a multi-class problem, we train multinomial logistic regression models. For the regression tasks, we train our models using SGD to minimize the least squares loss. We train all our models with mini-batches of size 250, and use GPUs on AWS EC2 machines for fast training (instance type p2.xlarge).

For all datasets, we tune the initial learning rate, as well as the kernel bandwidth, on the held-out set. We use the exact same kernel bandwidth and initial learning rate across all our models; in particular, Nyström and RFF models use the same hyperparameters. Using the same hyperparameters makes the trained models directly comparable. We use the same learning rate decay scheme described in Section 4.3.

For the Nyström models, we choose the landmark points either randomly from the training set, or using the k-means algorithm, as proposed by Zhang *et al.* [2008]. The k-means algorithm has been shown to be a particularly effective method for picking the landmark points, and was shown to be the best landmark selection algorithm among the ones compared in [Kumar *et al.*, 2012]. We train Nyström models with the number of features $D \in \{1250, 2500, 5000, 10000, 20000\}$, and we train RFF models with $D \in \{1250, 2500, 5000, 10000, 20000, 50000, 100000, 200000, 400000, 800000, 1600000\}$. We repeat these experiments 10 times for Nyström experiments with $D \leq 2500$, and for RFF experiments with $D \leq 20000$, initializing the landmark points and RFF projection matrix randomly for each experiment. Note that for the TIMIT dataset we do not train models with $D = 1{,}600{,}000$ due to the larger computational expense and memory requirements. Additionally, for Nyström experiments, if D is ever greater than the number of training points, we do not run the experiment. In the plots for these experiments, we show the minimum, median, and maximum values for a given experiment across its 10 random initializations.

Note that the reason we are able to train RFF models with *many* more features than for the Nyström method is because of the much larger memory requirements for the Nyström method. Specifically, to compute D features over d dimensional data, the Nyström method requires $O(Dd + D^2)$, while RFF only require $O(dD)$, as discussed in Section 2.3.2. This assumes that for Nyström, the number of landmark points m is equal to D, which is the setting we use for all our experiments.

For further details on training and hyperparameter choices, please see see Appendix D.2.

6.2.3 Results

In this section, we show how the Nyström method compares with random Fourier features, both in terms of kernel approximation error, and in terms of regression and classification performance. For clarify of presentation, we typically only show results on TIMIT, Forest, and YearPred, which correspond to our two largest classification tasks, and our largest regression task. We include results for the other datasets in Appendix D.3.

6.2.3.1 Kernel Approximation Error

In Figure 6.1, we show the mean squared kernel approximation error (MSE) of our various Nyström and RFF models, measured on a large number of random pairs of heldout points. Specifically, we show the average value of $(k(x, y) - z(x)^T z(y))^2$ for random x, y in the heldout set, where $z(x)$ can be either Nyström or RFF features.

We visualize these kernel approximation results in two ways. In the top of Figure 6.1, the x-axis denotes the number of features used in each model, while the y-axis shows the corresponding MSE. In the bottom plots, we instead show the kernel approximation error as a function of the memory required for each feature representation. Note that this is equal to $dD + D^2$ floating point numbers for Nyström, while it is equal to dD floats for RFF (to compute D features of d-dimensional data). The amount of memory used to compute a Nyström or RFF feature representation is very significant for at least three reasons: (1) it is roughly proportional to the amount of time it takes to compute the features, and to train a model with these features, (2) it determines how large a model one can train given the memory limitations of whatever hardware is being used, and (3) it lower bounds the memory footprint of any application using this model. Importantly, Nyström features are much more memory intensive than RFF, because they involve the multiplication with a very large matrix after computing the pairwise kernel values with all the landmark points. Furthermore, the memory (and computation time) requirements for computing the RFF features can be reduced quite significantly using structured matrix multiplications [Le *et al.*, 2013; Yu *et al.*, 2015], while the memory reductions are less extreme for methods that reduce the memory requirements of the Nyström method [Si *et al.*, 2014; Kumar *et al.*, 2009].

As you can see in Figure 6.1, for a fixed number of features, the Nyström method generally attains a *much* lower approximation error. Even when accounting for the increased memory required by the Nyström method (for a fixed number of features), it typically does better than RFF for a fixed amount of memory. However, for most of our datasets, the largest RFF models match and sometimes even surpass the kernel approximation performance of the Nyström models of equal size.

Interestingly, in many cases, the rate at which the RFF method's kernel approximation error decreases as its memory usage increases is greater than the Nyström method. This allows the RFF models to "catch up" to the performance of the Nyström models as the sizes of the models get larger.

In order to help interpret the above kernel approximation results, in Figure 6.2, we include plots of the normalized eigenvalues of the exact kernel matrices constructed from $N = 20k$ random training points. As one can see, for all the datasets the largest eigenvalues are significantly larger than the others, though there is a long tail of non-negligible eigenvalues.

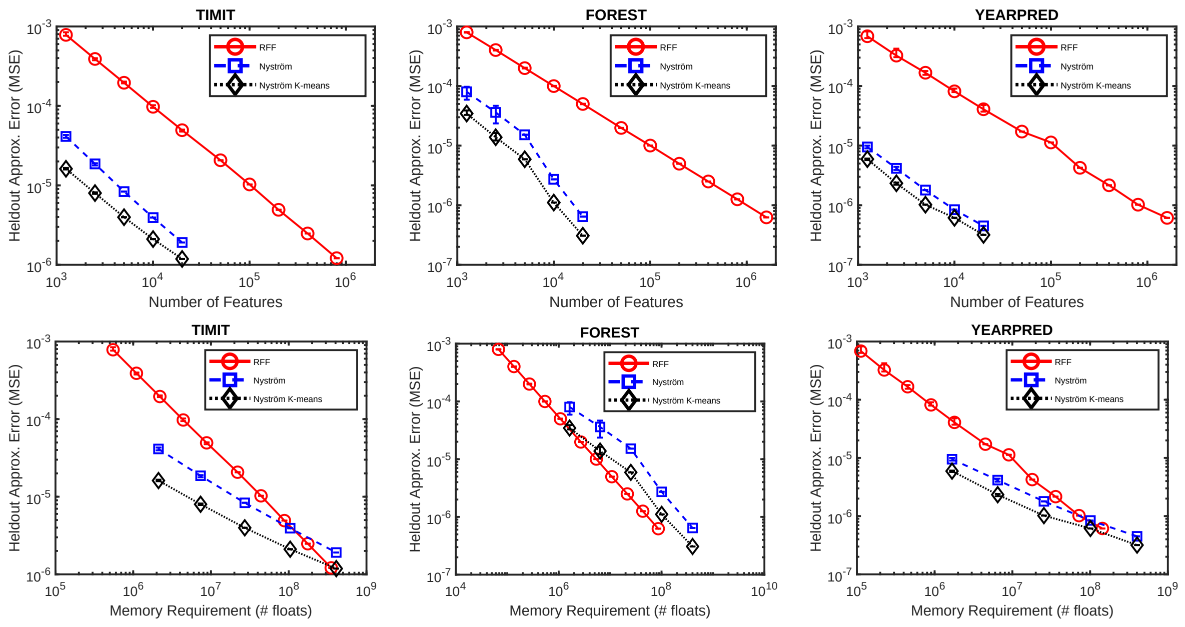

Figure 6.1: Kernel approximation errors for the Nyström method vs. random Fourier features, in terms of the total numbers of features (top) and the total memory requirement (bottom) in the respective models. Error is measured as mean squared error on the heldout set. For Nyström experiments with $D \leq 2500$, and RFF experiments with $D \leq 20000$, we run the experiments 10 times, and report the median, with error bars indicating the minimum and maximum. Note that due to small variance, error bars are often not clearly visible.

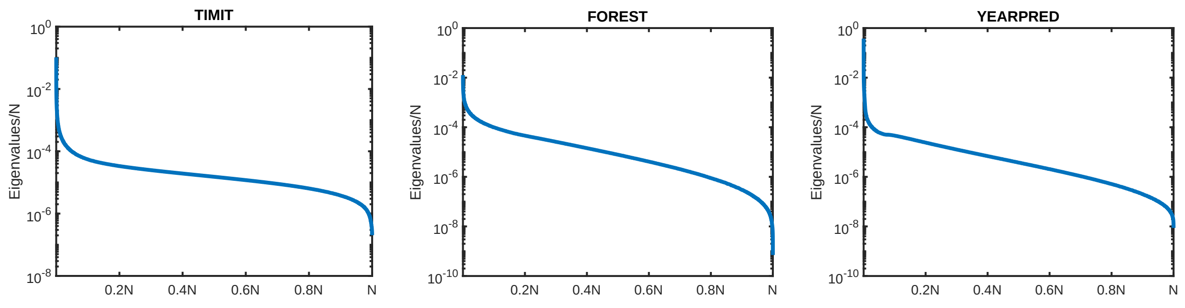

Figure 6.2: Spectrum of kernel matrices generated from $N = 20k$ random training points.

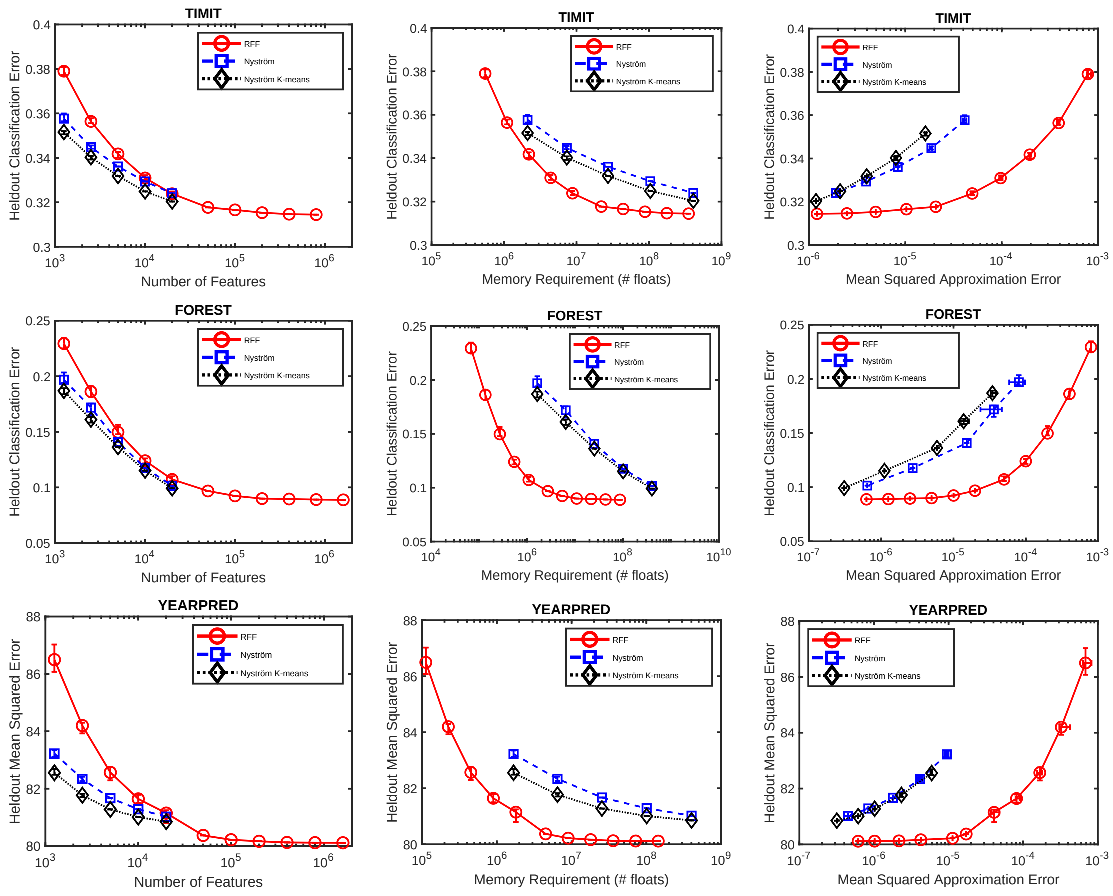

Figure 6.3: Heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the total numbers of features (left), total memory requirement (middle), and kernel approximation error (right) of the corresponding models. For Nyström experiments with $D \leq 2500$, and RFF experiments with $D \leq 20000$, we run the experiments 10 times, and report the median performance, with error bars indicating the minimum and maximum. Note that due to small variance, error bars are often not clearly visible.

6.2.3.2 Classification Performance

We now examine the classification performance of models trained using the Nyström and RFF features discussed above. In Figure 6.3, we show RFF and Nyström heldout classification or regression performance for all the models we trained. We see that for a fixed number of features, the Nyström method generally does better. However, the performance gap between these methods narrows for large numbers of features, and at $20{,}000$ features they generally attain very similar performance. Furthermore, because random Fourier features require significantly less memory and are cheaper to compute than Nyström features, we can train models with very large numbers of features (up to $1{,}600{,}000$), and these large RFF models outperform the best Nyström models we are able to train.

In the middle column of plots in Figure 6.3, we see that for a given amount of memory, the RFF models perform better than the Nyström models. This suggests that for a fixed memory or time budget, it is advisable to use RFF features instead of Nyström features. In the rightmost plots, we plot the mean squared kernel approximation error of each model (x-axis), and the corresponding heldout performance of the model (y-axis). Here we see that for a fixed mean squared kernel approximation error, random Fourier features perform much better than Nyström features. This result was quite surprising for us; we had imagined that two sets of features with similar approximation errors would perform very similarly in terms of classification performance when they were used to train models. However, this is definitely not the case with respect to Nyström features and random Fourier features. This surprising fact led us to perform a thorough analysis of the ways in which Nyström features and random Fourier features differ in the way they make approximation errors. This is what we investigate in the following section.

6.2.3.3 Kernel Approximation Error: A Second Look

In this section, we explore how the Nyström method and random Fourier features differ in the way they make approximation errors. In Figure 6.4, we show the histogram of errors for Nyström features and random Fourier features which have similar mean squared approximation errors on the TIMIT heldout set (1250 Nyström features have a MSE of 4.07×10^{-05}, while 20,000 RFF features have a MSE of 4.79×10^{-05}). To be precise, we show the histogram of $k(x, y) - z(x)^T z(y)$ values for a large number of random pairs of points in the heldout set. We show two different histograms for each method, corresponding to different ranges for the true value of $k(x, y)$. In the left plot, we

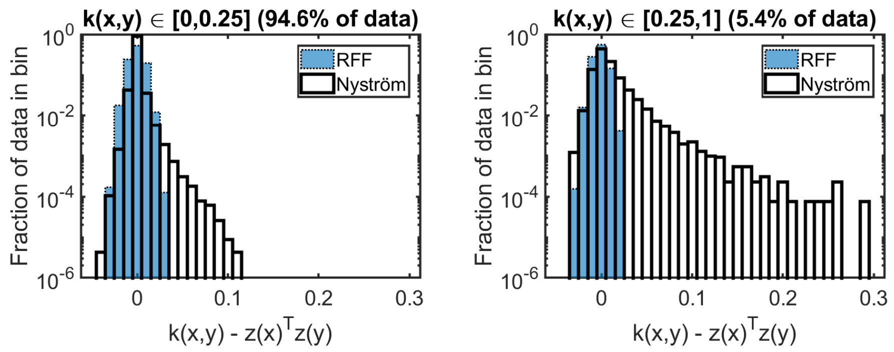

Figure 6.4: Histograms of kernel approximation errors for Nyström features random Fourier features. The different histograms correspond to a partition of the $k(x, y) - z(x)^T z(y)$ values based on the values of $k(x, y)$. Note that the Nyström method has many outliers for $k(x, y) \geq 0.25$, some of which are truncated from the histogram.

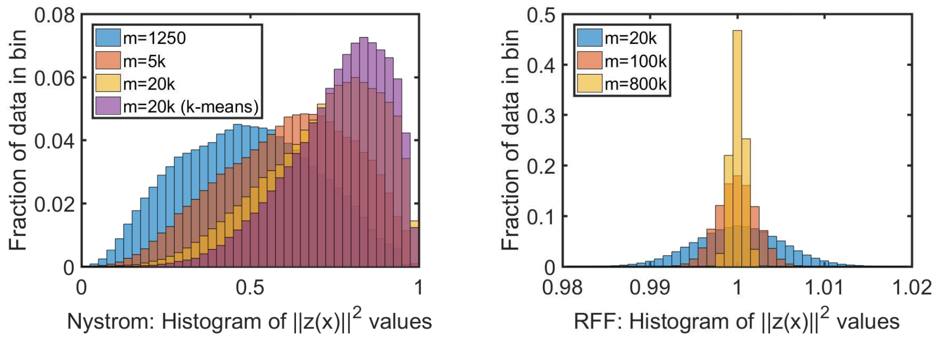

Figure 6.5: Histograms of the feature vector norms for Nyström (left) and RFF (right), for various numbers of features. Note that for the RBF kernel, $k(x, x) = 1 \; \forall x \in \mathcal{X}$, so a feature vector $z(x)$ of norm close to 1 approximates this self-similarity measure well.

Method	Metric	$k(x,y) \leq 0.25$	$k(x,y) \geq 0.25$	Global
Nyström	MSE	1.58×10^{-5}	$\mathbf{4.78 \times 10^{-4}}$	4.07×10^{-5}
RFF	MSE	4.82×10^{-5}	4.22×10^{-5}	4.79×10^{-5}
Nyström	Bias	-1.15×10^{-5}	$\mathbf{7.59 \times 10^{-3}}$	3.97×10^{-4}
RFF	Bias	-5.94×10^{-4}	-1.54×10^{-3}	-6.45×10^{-4}
Nyström	Variance	1.58×10^{-5}	$\mathbf{4.20 \times 10^{-4}}$	9.23×10^{-5}
RFF	Variance	4.79×10^{-5}	3.98×10^{-5}	4.83×10^{-5}

Table 6.2: MSE, Bias, and Variance of kernel approximation errors $k(x,y)-z(x)^T z(y)$ for Nyström ($m = 1250$), and RFF ($m = 20000$) features, estimated using many random pairs of points in the TIMIT heldout set. We partition these pairs of points (x,y) based on whether the true kernel value $k(x,y)$ is greater than or less than 0.25.

see that when the true kernel value is small ($k(x,y) \leq 0.25$), the Nyström method generally attains smaller errors, as can be seen by the larger number of points in the bin corresponding to errors in the range $[-.005, .005]$; note that while this effect appears quite small in the logarithmic scale plot, it is significant, corresponding to 91% vs. 53% of the data in this center bin for Nyström and RFF respectively. However, the Nyström method has more outliers than RFF does, with approximately 0.14% of the data having error greater than 0.035, while RFF has no outliers of this magnitude. The RFF distribution appears to be symmetric around 0, while the Nyström distribution is assymetric, with more outliers on the positive side than the negative side. In the right plot, we see that when the true kernel value is larger ($k(x,y) \geq 0.25$), the histogram of Nyström errors has much larger variance, and also shifts to the right (note also that this histogram is truncated on the right, as Nyström has some outliers beyond this range). In particular, approximately 6.6% of the data has error greater than 0.035. This shift indicates that the Nyström method has a positive bias in this range of $k(x,y)$ values, meaning that the Nyström features are typically *underestimating* $k(x,y)$ when $k(x,y) \geq 0.25$. The errors of the random Fourier features, in contrast, are always centered around zero. In Table 6.2, we show the MSE for both methods for the true kernel value ranges discussed above, and we also decompose the MSE into bias and variance ($MSE = Bias^2 + Var$). As you can see, the Nyström method has large variance, and positive bias, for $k(x,y) \geq 0.25$.

As we have seen, the Nyström method does poorly on points that are close in the input space (i.e., have high kernel value). The point closest to any point is itself, so in Figure 6.5 we examine the histograms of $\|z(x)\|^2$ values, which relate to the kernel approximation error at $k(x, x)$ as follows: $\|z(x)\|^2 = k(x, x) - (k(x, x) - z(x)^T z(x))$, where in our setting $k(x, x) = 1\ \forall x \in \mathcal{X}$. As can be seen in the left plot, the norms of the Nyström features are spread widely across the $[0, 1]$ range (shifting toward 1 for larger numbers of features m), whereas the norms of the RFF features are centered tightly around 1, indicating small error at approximating $k(x, x)$.

We now consider whether there is a way to penalize kernel approximation errors in such a way that the average penalty would correlate better with the heldout performance of the model. We have seen that the mean squared kernel approximation error is not a good metric for predicting heldout performance, given that RFF models generally perform better than Nyström models with similar kernel approximation MSE; viewed differently, this is equivalent to stating that for Nyström and RFF models which perform similarly in terms of heldout performance, the Nyström models have lower kernel approximation MSE. We have also seen that the Nyström method makes many more *large* kernel approximation errors than RFF does. This leads us to consider approximation penalties of the form $|k(x, y) - z(x)^T z(y)|^r$. This metric penalizes an error which is twice as big (in absolute value) 2^r times more. Thus, intuitively, for large values of r, the Nyström method will have many points on which it is penalized highly, thus bringing its average penalty closer to that of an RFF model with similar heldout performance. In Figure 6.6, we plot the heldout performance of our different models in terms of the kernel approximation error, for several values of r. As you can see, for each dataset there is a value of r for which the lines corresponding to the Nyström and RFF models overlap (for TIMIT and Forest, $r = 3.5$; for YearPred, $r = 5.5$). This suggests that large kernel approximation errors have a much larger negative effect on the performance of the trained model than smaller ones do (an error which is twice as big seems to "cost" $2^{3.5}$ or $2^{5.5}$ times as much as the smaller error), and that perhaps these large errors cause the Nyström models to not perform as well as the RFF models.

It is important to note that the bound on the approximation error of random Fourier features which we presented in Equation 2.6, can be easily modified to bound the probability that $|k(x, y) - z(x)^T z(y)|^r \geq \epsilon$:

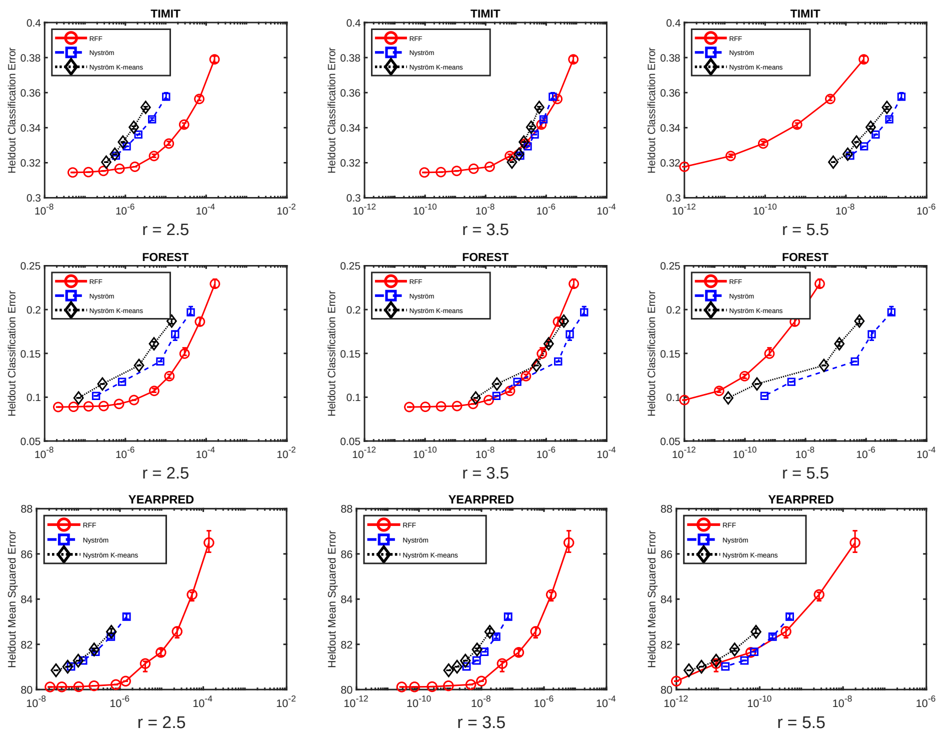

Figure 6.6: Heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the average kernel approximation errors, measured as $|k(x,y) - z(x)^T z(y)|^r$ for $r \in \{2.5, 3.5, 5.5\}$. Note that due to numeric underflow, some of the models with lowest approximation error sometimes do not appear in the $r = 5.5$ plots.

$$\mathbb{P}_{\omega,b}\big[|z(x)^T z(y) - k(x,y)|^r \geq \epsilon\big] \leq 2\exp\left(\frac{-D\epsilon^{2/r}}{8}\right). \tag{6.1}$$

6.3 Nyström method error analysis

In this section, we explore theoretically the question of why Nyström features have larger variance and positive bias when $k(x,y)$ is bigger, as well as the question of why the Nyström vectors have norms distributed in the fashion described above. For the full proofs of all the theorems presented in this section, as well as for short "proof sketches", please see Appendix D.

We begin with a theorem that states that if a Nyström representation $z(x)$ has small norm, then there exists an open ball around $x \in \mathcal{X}$ such that Nyström underestimates $k(x,y)$ for any y in this ball.

Theorem 4. *Given a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is $\hat{C}$-Lipschitz continuous in each of its arguments, let $C = \sup_{x,y} \frac{|k(x,x)-k(x,y)|}{\|x-y\|_2} \le \hat{C}$, and let $z_r(x) = \Sigma_r^{-1/2} U_r^T k_x$ be the rank-r Nyström approximation, using m landmark points $\{\hat{x}_1, \dots, \hat{x}_m\}$. Let $\hat{K}$ be the m by m kernel matrix of the landmark points ($\hat{K}_{ij} = k(\hat{x}_i, \hat{x}_j)$), and let λ_r denote the r^{th} biggest eigenvalue of $\hat{K}$. Then for any $\epsilon > 0$, and any $x, y \in \mathcal{X}$ satisfying*

$$\|x-y\|_2 < \frac{k(x,x) - \|z_r(x)\|^2 - \epsilon}{C(\lambda_r^{-1} m + 1)},$$

it follows that $k(x,y) - z_r(x)^T z_r(y) > \epsilon$.

In what follows, we will use the notation introduced in Appendix D.4, letting $\mathcal{H}$ denote the Reproducing Kernel Hilbert Space (RKHS) for a kernel function k, and letting $\varphi(x) \in \mathcal{H}$ denote the element in the RKHS corresponding to a point $x \in \mathcal{X}$. We will leverage Mercer's Theorem, which allows us to write any kernel function as $k(x,y) = \sum_{j=1}^{N} \lambda_j e_j(x) e_j(y)$. For further background, see Appendix D.4.

We will now show that assuming a kernel function $k(x,y) = \sum_{j=1}^{N} \lambda_j e_j(x) e_j(y)$ has an infinite spectrum ($\lambda_j > 0\ \forall j \in \mathbb{N}$) there must exist points x for which $k(x,x) - \|z(x)\|^2 > 0$, and thus, by Theorem 4, open sets around those points on which Nyström underestimates $k(x,y)$. In order to prove this, we will use the following definition for a *centered* kernel. Letting $\mu = \mathbb{E}_X[\varphi(X)] \in \mathcal{H}$, we define the centered kernel $k^c(x,y) = \langle \varphi(x) - \mu, \varphi(y) - \mu \rangle \equiv \langle \varphi^c(x), \varphi^c(y) \rangle$. It follows that $\mathbb{E}_X[\varphi^c(x)] = 0$, and that this k^c is a proper positive definite kernel, as it is defined as the dot-product in a Hilbert Space. Thus, by Mercer's theorem, it can be written as

$$k^c(x,y) = \sum_{j=1}^{N^c} \lambda_j^c e_j^c(x) e_j^c(y),$$

where $N_c \le \infty$ and all λ_j^c are strictly positive. Furthermore, because the (centered) covariance operator for $\varphi^c(X)$ is equal to the (uncentered) covariance operator of $\varphi(X)$ minus $\mu\mu^T$, the number of non-zero eigenvalues N^c decreases by at most 1 relative to N (and thus N^c is infinite if N is infinite). We now proceed with Theorem 2.

Theorem 5. *For $k(x,y) = \sum_{j=1}^{N} \lambda_j \phi_j(x) \phi_j(y)$, and its corresponding centered kernel $k^c(x,y) = \sum_{j=1}^{N^c} \lambda_j^c \phi_j^c(x) \phi_j^c(y)$, the expected error made by* any *m-dimensional Nyström approximation $z(x)$*

(for any $m \le N$) in approximating $k(x,x)$ satisfies

$$\mathbb{E}_X\left[k(X,X) - \|z(X)\|^2\right] \ge \sum_{j=m+1}^{N^c} \lambda_j^c,$$

regardless of the choice of landmark points.

One immediate consequence of the above theorem is that $\mathbb{E}_X\left[\|z(X)\|^2\right] \le \mathbb{E}_X\left[k(X,X)\right] - \sum_{j=m+1}^{N^c} \lambda_j^c$. Thus, $\mathbb{E}_X\left[\|z(X)\|^2\right]$ is strictly less than $\mathbb{E}_X\left[k(X,X)\right]$, which means that the Nyström method produces biased estimates of $k(x,x)$. We note that this result is in sharp contrast to random Fourier features, which produced unbiased estimates of $k(x,y)$ for all $x, y \in \mathcal{X}$.

Corollary 5.1. *For a Nyström mapping $z_m : \mathcal{X} \to \mathbb{R}^m$ using m landmark points, approximating a kernel k with centered spectrum $(\lambda_i^c)_{i=1}^N$, there must exist $x \in \mathcal{X}$ such that*

$$k(x,x) - \|z_m(x)\|^2 \ge \sum_{j=m+1}^{N^c} \lambda_j^c$$

Proof. This follows directly from the previous theorem, because in order for the expectation of $k(X,X) - \|z_m(X)\|^2$ to be greater than or equal to $\sum_{j=m+1}^{N^c} \lambda_j^c$, there must exist a point $x \in \mathcal{X}$ such that $k(x,x) - \|z_m(x)\|^2 \ge \sum_{j=m+1}^{N^c} \lambda_j^c$. □

In the next theorem we lower bound the probability that the Nyström method with m landmark points underestimates $k(x,x)$ by more than ϵ. We prove this in the context of bounded kernels $k(x,y) \in [a,b]$, and we will assume without loss of generality that $k(x,y) \in [0,1]$.

Theorem 6. *For a bounded kernel $k(x,y) = \sum_{j=1}^N \lambda_j \phi_j(x)\phi_j(y) \in [0,1]$, and* any *Nyström approximation $z(x)$ with m landmark points (for $m \le N$), the following holds:*

$$\mathbb{P}_X\Big[k(X,X) - \|z(X)\|^2 \ge \epsilon\Big] \ge 1 - \exp\Big(-2\big(R_m - \epsilon\big)^2\Big),$$

for $R_m = \sum_{j=m+1}^N \lambda_j^c$, and $0 \le \epsilon \le R_m$. If we further restrict $\epsilon \le R_m/2$, it immediately follows that

$$\mathbb{P}_X\Big[k(X,X) - \|z(X)\|^2 \ge \epsilon\Big] \ge 1 - \exp\big(-R_m^2/2\big).$$

This proof (see Appendix D) involves a straightforward application of Hoeffding's inequality to the result of Theorem 2, letting the random variable under question be $f(X) = k(X, X) - \|z(X)\|^2 \in [0, 1]$.

We now show that not only is the Nyström method biased in estimating $k(x, x)$, it is in general biased in its estimation of $k(x, y)$ for random pairs of $x, y \in \mathcal{X}$.

Theorem 7. *Let $\mu = \mathbb{E}_X[\varphi(X)]$ be the average element in the RKHS $\mathcal{H}$ corresponding to the kernel k, where X is drawn from a probability distribution p over the input space $\mathcal{X}$. Let $A = span(\varphi(\hat{x}_1), \ldots, \varphi(\hat{x}_m))$ be the m-dimensional subspace of $\mathcal{H}$ corresponding to landmark points $\{\hat{x}_1, \ldots, \hat{x}_m\} \subset \mathcal{X}$. Additionally, let $z(x)$ be the Nyström representation corresponding to these landmark points, let $A^\perp$ denote the orthogonal complement of A in $\mathcal{H}$, and let $\mu = \mu_A + \mu_{A^\perp}$, where $\mu_A \in A$ and $\mu_{A^\perp} \in A^\perp$. Then*

$$\mathbb{E}_{X,Y}\left[k(X, Y) - z(X)^T z(Y)\right] = \|\mu_{A^\perp}\|^2,$$

where this expectation is over X and Y drawn independently from p.

Thus, $\mu \in A$ implies $\mathbb{E}_{X,Y}\left[k(X, Y) - z(X)^T z(Y)\right] = 0$, and $\mu \notin A$ implies $\mathbb{E}_{X,Y}\left[k(X, Y) - z(X)^T z(Y)\right] > 0$. So the Nyström method is biased when $\mu \notin A$, underestimating the true kernel value in expectation.

6.4 Conclusion

In this chapter, we explored from both empirical and theoretical perspectives, the differences between random Fourier features and Nyström features. We found that they differ significantly in their distributions of approximation errors, and argued that these differences lead to the Nyström method performing poorly on classification and regression tasks, in spite of its low average kernel approximation errors. We further showed that random Fourier features outperform Nyström features in classification and regression performance under a fixed computational budget, and quite surprisingly, that random Fourier features outperform Nyström features with similar mean approximation error. We believe that together, these contributions shed light on important questions at the heart of kernel approximation methods.

Chapter 7

Conclusion

Kernel approximation methods have several attractive qualities: (1) Training is a convex optimization problem, and many of the convergence and generalization bounds from the kernel literature can be adapted to the approximation setting. (2) They are relatively straightforward to interpret: In particular, they are like a smoothed version of a k-nearest neighbor classifier.

In this thesis, we have demonstrated the viability of kernel approximation methods for large-scale multi-class classification problems. Specifically, we have shown that the random Fourier features method of Rahimi and Recht can compete with fully-connected feedforward neural networks on the acoustic modeling task for speech recognition. We have proposed a new feature selection method, which is able to quickly search through large numbers of features to find those best suited for the task. We have introduced a new sparse kernel, the sparse Gaussian kernel, which can be more efficient to approximate, and generally performs better than (or on par with) the Gaussian and Laplacian kernels, regardless of whether or not feature selection is performed. These methods, together with the linear bottleneck method of Sainath *et al.* [2013a], and using entropy regularized perplexity for learning rate decay [Lu *et al.*, 2016], brought the performance of the kernel methods on par with that of the DNNs. This is the first time that it has been demonstrated, in a very large-scale setting where deep learning techniques have been known to dominate, that kernel approximation methods can effectively compete with fully-connected feedforward neural networks.

In addition to these contributions, we have uncovered an important fact about kernel approximation methods—namely, that the specific *types* of approximation errors made by these methods have an important effect on the performance of the models trained with these features. In particular, two sets of features with the same mean squared kernel approximation error can yield models with very different performance. Along these lines, we found that the Nyström method sometimes *largely underestimates* the true value of the kernel, while the random Fourier features method never makes large mistakes. Thus, even though the Nyström method attains impressively low kernel approximation errors for a fixed number of features, the random Fourier features method performs better than it under a fixed computational budget.

We believe that these contributions demonstrate the potential of kernel approximation methods on challenging large scale tasks. We now discuss some areas for future research which we believe to be promising.

7.1 Future work

For future work, we propose to:

1. Develop "recurrent" and "convolutional" kernels, which take advantage of the temporal and/or spatial structure of an input in order to attain better performance, and put these to the test on challenging large-scale tasks (e.g., computer vision, speech recognition, NLP). We believe specialized kernels, which leverage the structure of the input, will be important for competing with CNNs and LSTMs, which are currently the state of the art methods in a number of domains.

2. Test kernel methods in other domains in order to see to what extent the results presented in this thesis generalize.

3. Leverage the shallow structure of kernel models in order to develop highly efficient parallel implementations for training kernel models, as well as for evaluating kernel models. Thus, even if a model is quite large, it could be very fast to evaluate in parallel. This could help mitigate the fact that sometimes a *very* large number of features is required to attain strong performance.

4. Analyze, in the case of the linear bottleneck with the cross-entropy objective, whether all local minima are also global minimizers. Similar results have been shown in the context of matrix sensing, matrix completion, and robust PCA [Ge *et al.*, 2016; Ge *et al.*, 2017].

5. Determine if there are any formal guarantees which can be proven about our feature selection method.

6. Formalize the way in which the larger kernel approximation errors made by the Nyström method affect the performance of the models trained using Nyström features.

7. Use approximation methods to efficiently learn the optimal dual parameters[1] for large-scale kernel models. This could lead to improved performance, and also allow whoever is deploying the model to tune the level of approximation they are willing to tolerate, in order to evaluate the model more quickly.[2]

8. Develop more interpretable kernel models. While in theory kernel models are easy to interpret, this interpretability can be clouded by things like: (1) The high-dimensionality of the problem (hard to visualize points in high-dimensions, or understand what it really means for two points to be near each other in this space, especially when the features themselves are somewhat opaque). (2) The fact that these models are *approximate* kernel models. (3) The fact that to save space, we generally do not store the α values ($f(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x)$), which in a sense explain exactly what the model is doing. Furthermore, even if we did store the α values, it would be difficult to make sense of them, given the very large number of training examples and classes. A couple avenues which could help improve interpretability are: (1) storing the α parameters, (2) learning models with sparse α coefficients, and (3) developing kernel functions which are easier to visualize in high-dimensions.

9. Recent work has shown that deep neural networks do more than simply memorize the training set; rather, they learn simple patterns first, and then learn the more difficult training examples [Arpit *et al.*, 2017]. Do kernel methods do something similar?

10. It is an open question whether there are inherent limitations to kernel methods, as suggested, for example, by Bengio and Lecun [2007]. Are "template matching methods" doomed to fail

[1]The dual parameters are the α parameters in the kernel model $f(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x)$

[2]For example, they can take $f(x) = \langle w_D, z_D(x) \rangle$, where $z_D(x)$ is the D-dimensional random Fourier feature representation for $x \in \mathcal{X}$, and where $w_D = \sum_{i=1}^{N} \alpha_i z_D(x_i) \in \mathbb{R}^D$.

in the very high-dimensional setting, due to the very large[3] number of training points which would be necessary to "cover" this space? Or are there simple kernels which are well-suited for these very high dimensional settings (e.g., images)? Along a similar vein, are DNNs really doing something very different than "template matching", or are they actually template matching algorithms in disguise?

We believe that there are many open questions regarding the limits of kernel methods, and their performance relative to DNNs. We are excited to continue pushing the boundaries of what is known in this area.

[3]Exponential in the dimension of the data, for example.

Bibliography

[Agarwal *et al.*, 2017] Naman Agarwal, Zeyuan Allen-Zhu, Brian Bullins, Elad Hazan, and Tengyu Ma. Finding approximate local minima faster than gradient descent. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1195–1199, New York, NY, USA, 2017. ACM.

[Anandkumar and Ge, 2016] Animashree Anandkumar and Rong Ge. Efficient approaches for escaping higher order saddle points in non-convex optimization. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*, pages 81–102, 2016.

[Andor *et al.*, 2016] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[Aronszajn, 1950] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[Arpit *et al.*, 2017] Devansh Arpit, Stanislaw K. Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron C. Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 233–242, 2017.

[Ba and Caruana, 2014] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2654–2662, 2014.

[Bahl *et al.*, 1986] L. Bahl, P. Brown, P. de Souza, and R. Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86.*, volume 11, pages 49–52, Apr 1986.

[Bartlett *et al.*, 2002] Peter L. Bartlett, Olivier Bousquet, and Shahar Mendelson. Localized Rademacher complexities. In *Proceedings of the 15th Annual Conference on Computational Learning Theory*, COLT '02, pages 44–58, London, UK, UK, 2002. Springer-Verlag.

[Bartlett, 1996] Peter L. Bartlett. For valid generalization the size of the weights is more important than the size of the network. In *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pages 134–140, 1996.

[Bengio and Lecun, 2007] Yoshua Bengio and Yann Lecun. Scaling learning algorithms towards ai, 2007.

[Berlinet and Thomas-Agnan, 2003] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer US, 2003.

[Bianchini and Scarselli, 2014] Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Trans. Neural Netw. Learning Syst.*, 25(8):1553–1565, 2014.

[Bottou *et al.*, 2007] Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston, editors. *Large Scale Kernel Machines*. MIT Press, Cambridge, MA., 2007.

[Chen *et al.*, 2016] Jie Chen, Lingfei Wu, Kartik Audhkhasi, Brian Kingsbury, and Bhuvana Ramabhadrari. Efficient one-vs-one kernel ridge regression for speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 2454–2458, 2016.

[Choromanska *et al.*, 2015] Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015.

[Clarkson, 2010] Kenneth L. Clarkson. Coresets, Sparse Greedy Approximation, and the Frank-Wolfe Algorithm. *ACM Trans. Algorithms*, 6(4):63:1–63:30, 2010.

[Cybenko, 1989] George Cybenko. Approximation by superpositions of a sigmoidal function. *MCSS*, 2(4):303–314, 1989.

[Dai *et al.*, 2014] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3041–3049, 2014.

[Dasgupta and McAllester, 2013] Sanjoy Dasgupta and David McAllester, editors. *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*. JMLR.org, 2013.

[Dauphin *et al.*, 2014] Yann N. Dauphin, Razvan Pascanu, Çaglar Gülçehre, KyungHyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2933–2941, 2014.

[DeCoste and Schölkopf, 2002] Dennis DeCoste and Bernhard Schölkopf. Training Invariant Support Vector Machines. *Mach. Learn.*, 46:161–190, 2002.

[Dehak *et al.*, 2011] Najim Dehak, Patrick Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification. *IEEE Trans. Audio, Speech & Language Processing*, 19(4):788–798, 2011.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[Duchi and Singer, 2009] John C. Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

[Gales and Young, 2007] Mark Gales and Steve Young. The application of hidden Markov models in speech recognition. *Found. Trends Signal Process.*, 1(3):195–304, January 2007.

[Gales, 1998] M.J.F. Gales. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech & Language*, 12(2):75 – 98, 1998.

[Garofolo *et al.*, 1993] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren. DARPA TIMIT acoustic phonetic continuous speech corpus CDROM, 1993.

[Ge *et al.*, 2016] Rong Ge, Jason D. Lee, and Tengyu Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2973–2981, 2016.

[Ge *et al.*, 2017] Rong Ge, Chi Jin, and Yi Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1233–1242, 2017.

[Gibson and Hain, 2006] Matthew Gibson and Thomas Hain. Hypothesis spaces for minimum Bayes risk training in large vocabulary speech recognition. In *INTERSPEECH 2006 - ICSLP, Ninth International Conference on Spoken Language Processing, Pittsburgh, PA, USA, September 17-21, 2006*. ISCA, 2006.

[Gittens and Mahoney, 2013] Alex Gittens and Michael W. Mahoney. Revisiting the Nyström method for improved large-scale machine learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 567–575, 2013.

[Glorot and Bengio, 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010.

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

[Graves *et al.*, 2006] Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 369–376, 2006.

[Guenter *et al.*, 2013] Brian Guenter, Dong Yu, Adam Eversole, Oleksii Kuchaiev, and Mike Seltzer. Stochastic gradient descent algorithm in the computational network toolkit. December 2013.

[Halko *et al.*, 2011] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[Hamid *et al.*, 2014] Raffay Hamid, Ying Xiao, Alex Gittens, and Dennis DeCoste. Compact random feature maps. In Xing and Jebara [2014], pages 19–27.

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[Härdle *et al.*, 2004] Wolfgang Karl Härdle, Marlene Müller, Stefan Sperlich, and Axel Werwatz. *Nonparametric and semiparametric models*. Springer Science & Business Media, 2004.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[Hinton *et al.*, 2006] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[Hinton *et al.*, 2012] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[Hornik *et al.*, 1989] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[Huang *et al.*, 2014] Po-Sen Huang, Haim Avron, Tara N. Sainath, Vikas Sindhwani, and Bhuvana Ramabhadran. Kernel methods match deep neural networks on TIMIT. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*, pages 205–209. IEEE, 2014.

[Hwang *et al.*, 1993] M. Y. Hwang, X. Huang, and F. Alleva. Predicting unseen triphones with senones. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 311–314 vol.2, April 1993.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[Jameson, 2015] G. J. O. Jameson. A simple proof of Stirling's formula for the gamma function. 99(544):68–74, March 2015.

[Kaiser *et al.*, 2000] Janez Kaiser, Bogomir Horvat, and Zdravko Kacic. A novel loss function for the overall risk criterion based discriminative training of HMM models. In *Sixth Interna-*

tional Conference on Spoken Language Processing, ICSLP 2000 / INTERSPEECH 2000, Beijing, China, October 16-20, 2000, pages 887–890. ISCA, 2000.

[Kar and Karnick, 2012] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In Neil D. Lawrence and Mark A. Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, April 21-23, 2012*, volume 22 of *JMLR Proceedings*, pages 583–591. JMLR.org, 2012.

[Kingsbury *et al.*, 2013] B. Kingsbury, J. Cui, X. Cui, M. J. F. Gales, K. Knill, J. Mamou, L. Mangu, D. Nolden, M. Picheny, B. Ramabhadran, R. Schlüter, A. Sethy, and P. C. Woodland. A High-performance Cantonese Keyword Search System. In *Proc. ICASSP*, pages 8277–8281, 2013.

[Kingsbury, 2009] Brian Kingsbury. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2009, 19-24 April 2009, Taipei, Taiwan*, pages 3761–3764. IEEE, 2009.

[Kiperwasser and Goldberg, 2016] Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327, 2016.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Pereira et al. [2012], pages 1097–1105.

[Kumar *et al.*, 2009] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Ensemble Nyström method. In Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1060–1068, 2009.

[Kumar *et al.*, 2012] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the Nyström method. *Journal of Machine Learning Research*, 13:981–1006, 2012.

[Le *et al.*, 2013] Quoc V. Le, Tamás Sarlós, and Alexander J. Smola. Fastfood – approximating kernel expansions in loglinear time. In Dasgupta and McAllester [2013], pages 244–252.

[Le Roux *et al.*, 2012] Nicolas Le Roux, Mark W. Schmidt, and Francis R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In Pereira et al. [2012], pages 2672–2680.

[Lu *et al.*, 2016] Zhiyun Lu, Dong Quo, Alireza Bagheri Garakani, Kuan Liu, Avner May, Aurélien Bellet, Linxi Fan, Michael Collins, Brian Kingsbury, Michael Picheny, and Fei Sha. A comparison between deep neural nets and kernel acoustic models for speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 5070–5074. IEEE, 2016.

[May *et al.*, 2016] Avner May, Michael Collins, Daniel J. Hsu, and Brian Kingsbury. Compact kernel models for acoustic modeling via random feature selection. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 2424–2428. IEEE, 2016.

[May *et al.*, 2017] Avner May, Alireza Bagheri Garakani, Zhiyun Lu, Dong Guo, Kuan Liu, Aurélien Bellet, Linxi Fan, Michael Collins, Daniel J. Hsu, Brian Kingsbury, Michael Picheny, and Fei Sha. Kernel approximation methods for speech recognition. *CoRR*, abs/1701.03577, 2017.

[mer, 1909] Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446, 1909.

[Micchelli *et al.*, 2006] Charles A. Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *Journal of Machine Learning Research*, 6:2651–2667, 2006.

[Mikolov *et al.*, 2010] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.

[Mikolov *et al.*, 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR Workshop*, 2013.

[Mohamed *et al.*, 2012] Abdel-rahman Mohamed, George Dahl, and Geoffrey Hinton. Acoustic Modeling Using Deep Belief Networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[Mohri *et al.*, 2002] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[Montúfar *et al.*, 2014] Guido F. Montúfar, Razvan Pascanu, KyungHyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Corinna Cortes, Neil Lawrence, and Kilian Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2924–2932, 2014.

[Morgan and Bourlard, 1990] N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in Neural Information Processing Systems 2*, 1990.

[Morgan and Bourlard, 1995] N. Morgan and H. A. Bourlard. Neural networks for statistical recognition of continuous speech. *Proceedings of the IEEE*, 83(5):742–772, May 1995.

[Neyshabur *et al.*, 2015] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *International Conference on Learning Representations*, 2015.

[Pennington and Bahri, 2017] Jeffrey Pennington and Yasaman Bahri. Geometry of neural network loss surfaces via random matrix theory. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 2798–2806, 2017.

[Pennington *et al.*, 2015] Jeffrey Pennington, Felix X. Yu, and Sanjiv Kumar. Spherical random features for polynomial kernels. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1846–1854, 2015.

[Pereira *et al.*, 2012] F. Pereira, C.J.C. Burges, L. Bottou, and K. Q. Weinberger, editors. *Advances in Neural Information Processing Systems 25*, 2012.

[Platt, 1998] John C. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[Povey and Kingsbury, 2007] Daniel Povey and Brian Kingsbury. Evaluation of proposed modifications to MPE for large scale discriminative training. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2007, Honolulu, Hawaii, USA, April 15-20, 2007*, pages 321–324. IEEE, 2007.

[Povey and Woodland, 2002] D. Povey and P. C. Woodland. Minimum phone error and i-smoothing for improved discriminative training. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 1, pages I–105–I–108, May 2002.

[Povey *et al.*, 2008] Daniel Povey, Dimitri Kanevsky, Brian Kingsbury, Bhuvana Ramabhadran, George Saon, and Karthik Visweswariah. Boosted MMI for model and feature-space discriminative training. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2008, March 30 - April 4, 2008, Caesars Palace, Las Vegas, Nevada, USA*, pages 4057–4060, 2008.

[Povey *et al.*, 2016] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahrmani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for asr based on lattice-free mmi. In *Interspeech*, 2016.

[Rahimi and Recht, 2007] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1177–1184, 2007.

[Rahimi and Recht, 2008] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1313–1320, 2008.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[Sainath *et al.*, 2011] T.N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A. Mohamed. Making deep belief networks effective for large vocabulary continuous speech recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 30–35. IEEE, 2011.

[Sainath *et al.*, 2013a] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arısoy, and Bhuvana Ramabhadran. Low-rank Matrix Factorization for Deep Neural Network Training with High-dimensional Output Targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE, 2013.

[Sainath *et al.*, 2013b] Tara N. Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 8614–8618, 2013.

[Sainath *et al.*, 2013c] T.N. Sainath, B. Kingsbury, H. Soltau, and B. Ramabhadran. Optimization techniques to improve training speed of deep neural networks for large speech tasks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(11):2267–2276, Nov 2013.

[Sak *et al.*, 2014] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 338–342, 2014.

[Saon *et al.*, 2016] George Saon, Tom Sercu, Steven J. Rennie, and Hong-Kwang Jeff Kuo. The IBM 2016 english conversational telephone speech recognition system. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 7–11, 2016.

[Saon *et al.*, 2017] George Saon, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, Bergul Roomi, and Phil Hall. English conversational telephone speech recognition by humans and machines. In *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, 2017.

[Schölkopf and Smola, 2002] Bernhard Schölkopf and Alexander Johannes Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond.* Adaptive computation and machine learning series. MIT Press, 2002.

[Schölkopf *et al.*, 2001] Bernhard Schölkopf, Ralf Herbrich, and Alexander J. Smola. A generalized representer theorem. In *Computational Learning Theory, 14th Annual Conference on Computational Learning Theory, COLT 2001 and 5th European Conference on Computational Learning Theory, EuroCOLT 2001, Amsterdam, The Netherlands, July 16-19, 2001, Proceedings*, pages 416–426, 2001.

[Seide *et al.*, 2011a] Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In David Nahamoo and Michael Picheny, editors, *2011 IEEE Workshop on Automatic Speech Recognition & Understanding, ASRU 2011, Waikoloa, HI, USA, December 11-15, 2011*, pages 24–29. IEEE, 2011.

[Seide *et al.*, 2011b] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011*, pages 437–440. ISCA, 2011.

[Sejdinovic and Gretton, 2012] Dino Sejdinovic and Arthur Gretton. What is an RKHS? `http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/RKHS_Notes1.pdf`, 2012. [Online; accessed 27-October-2017].

[Sercu and Goel, 2016] Tom Sercu and Vaibhava Goel. Advances in very deep convolutional neural networks for LVCSR. In *Interspeech 2016, 17th Annual Conference of the International Speech*

Communication Association, San Francisco, CA, USA, September 8-12, 2016, pages 3429–3433, 2016.

[Si *et al.*, 2014] Si Si, Cho-Jui Hsieh, and Inderjit S. Dhillon. Memory efficient kernel approximation. In Xing and Jebara [2014], pages 701–709.

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[Smola, 2014] Alex Smola. Personal communication, 2014.

[Soltau *et al.*, 2014] Hagen Soltau, George Saon, and Tara N. Sainath. Joint training of convolutional and non-convolutional neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*, pages 5572–5576, 2014.

[Sonnenburg and Franc, 2010] Sören Sonnenburg and Vojtech Franc. COFFIN: A computational framework for linear svms. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 999–1006. Omnipress, 2010.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[Steinwart, 2004] I. Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In *Advances in Neural Information Processing Systems 16*, 2004.

[Ström, 1997] Nikko Ström. Sparse connection and pruning in large dynamic artificial neural networks. In *Fifth European Conference on Speech Communication and Technology, EUROSPEECH 1997, Rhodes, Greece, September 22-25, 1997*, 1997.

[Sundermeyer *et al.*, 2012] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pages 194–197, 2012.

[Sutskever *et al.*, 2013] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In Dasgupta and McAllester [2013], pages 1139–1147.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

[Tsang *et al.*, 2005] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core Vector Machines: Fast SVM Training on Very Large Data Sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[Valtchev *et al.*, 1997] V Valtchev, J.J Odell, P.C Woodland, and S.J Young. MMIE training of large vocabulary recognition systems. *Speech Communication*, 22(4):303 – 314, 1997.

[van den Berg *et al.*, 2017] Ewout van den Berg, Bhuvana Ramabhadran, and Michael Picheny. Training variance and performance evaluation of neural networks in speech. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pages 2287–2291, 2017.

[Vasilache *et al.*, 2015] Nicolas Vasilache, Jeff Johnson, Michaël Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A GPU performance evaluation. In *Proceedings of the 3rd International Conference on Learning Representations, San Diage, CA, USA, May 7-9, 2015*, 2015.

[Vedaldi and Zisserman, 2012] A. Vedaldi and A. Zisserman. Efficient Additive Kernels via Explicit Feature Maps. *IEEE Trans. on Pattern Anal. & Mach. Intell.*, 34(3):480–492, 2012.

[Veselý *et al.*, 2013] Karel Veselý, Arnab Ghoshal, Lukás Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In Frédéric Bimbot, Christophe Cerisara, Cécile Fougeron, Guillaume Gravier, Lori Lamel, François Pellegrino, and Pascal Perrier, editors, *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 2345–2349. ISCA, 2013.

[Viterbi, 1967] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[Williams and Seeger, 2001] C.K.I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T.K. Leen, T.G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[Xie *et al.*, 2017] Bo Xie, Yingyu Liang, and Le Song. Diverse neural network learns true target functions. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, pages 1216–1224, 2017.

[Xing and Jebara, 2014] Eric P. Xing and Tony Jebara, editors. *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*. JMLR.org, 2014.

[Xiong *et al.*, 2016] W. Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *CoRR*, abs/1610.05256, 2016.

[Xiong *et al.*, 2017] W. Xiong, L. Wu, F. Alleva, Jasha Droppo, X. Huang, and Andreas Stolcke. The microsoft 2017 conversational speech recognition system. *CoRR*, abs/1708.06073, 2017.

[Xue *et al.*, 2013] Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 2365–2369, 2013.

[Yang *et al.*, 2012] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. In Pereira et al. [2012], pages 485–493.

[Yang *et al.*, 2015] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alexander J. Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1476–1483, 2015.

[Yen *et al.*, 2014] E.-H. Yen, T.-W. Lin, S.-D. Lin, P.K. Ravikumar, and I.S. Dhillon. Sparse random feature algorithm as coordinate descent in Hilbert space. In *Advances in Neural Information Processing Systems 27*, 2014.

[Young *et al.*, 1994] S. J. Young, J. J. Odell, and P. C. Woodland. Tree-based state tying for high accuracy acoustic modelling. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 307–312, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[Yu *et al.*, 2015] Felix X. Yu, Sanjiv Kumar, Henry A. Rowley, and Shih-Fu Chang. Compact nonlinear maps and circulant extensions. *CoRR*, abs/1503.03893, 2015.

[Zhang *et al.*, 2008] Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved Nyström low-rank approximation and error analysis. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 1232–1239, 2008.

[Zhang *et al.*, 2017] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proceedings of the 5th International Conference on Learning Representations, Toulon, France, April 24-26, 2017*, 2017.

Appendices

Appendix A

Definitions

- **Metric Space** A metric space $(\mathcal{X}, d)$ is a set $\mathcal{X}$ together with a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying:

    1. $d(x, x') \geq 0 \; \forall x, x' \in \mathcal{X}$, with equality if and only if $x = x'$ (non-negative).
    2. $d(x, x') = d(x', x)$ (symmetric).
    3. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

- **Cauchy Sequence**: A Cauchy sequence $(x_1, x_2, \ldots)$ is an infinite sequence of points x_i in a metric space $(\mathcal{X}, d)$, satisfying the following: For every $\epsilon > 0$, there exists $N \in \mathbb{N}$ such that for all $m, n > N$, $d(x_m, x_n) < \epsilon$.

- **Limit of a sequence**: A point x in a metric space $(\mathcal{X}, d)$ is the limit of an infinite sequence $(x_1, x_2, \ldots)$ if for every $\epsilon > 0$, there exists $N \in \mathbb{N}$ such that for all $n > N$, $d(x, x_n) < \epsilon$.

- **Complete Space**: A metric space $(\mathcal{X}, d)$ is complete if every Cauchy sequence in $\mathcal{X}$ has a limit, and the limit is in $\mathcal{X}$.

- **Normed Space**: A normed vector space $(\mathcal{X}, \|\cdot\|)$ is a vector space $\mathcal{X}$ over the field $\mathbb{F}$, together with a norm function $\|\cdot\| : \mathcal{X} \to \mathbb{R}$. The norm function must satisfy (for all $x, y \in \mathcal{X}$):

    1. $\|x\| \geq 0$, with $\|x\| = 0$ if and only if $x = \mathbf{0}$.
    2. $\|\alpha x\| = |\alpha| \|x\|$, for any scalar $\alpha \in \mathbb{F}$
    3. $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality).

- **Inner Product Space**: An inner product space $(\mathcal{X}, \langle \cdot, \cdot \rangle)$ is a is a vector space $\mathcal{X}$ over the field $\mathbb{F}$, together with an inner product function $\langle \cdot, \cdot \rangle : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The inner product function must satisfy (for all $x, y, z \in \mathcal{X}$, and all $\alpha \in \mathbb{F}$):
  1. $\langle x, y \rangle = \overline{\langle y, x \rangle}$ (conjugate symmetry).
  2. $\langle ax + y, z \rangle = a\langle x, z \rangle + \langle y, z \rangle$ (linearity in the first argument).
  3. $\langle x, x \rangle \geq 0$, with equality if and only if $x = \mathbf{0}$ (positive-definiteness).
- **Banach Space**: A Banach space is a complete normed space.
- **Hilbert Space**: A Hilbert space is a complete inner product space; i.e., it is a Banach space with an inner product.
- **Isometric Isomorphism**: An isometric isomorphism $L : \mathcal{U} \to \mathcal{V}$ between two inner product spaces is a bijective linear map which preserves inner products—namely, for any $u, u' \in \mathcal{U}$, $\langle u, u' \rangle_{\mathcal{U}} = \langle L(u), L(u') \rangle_{\mathcal{V}}$.
- **Positive semi-definite matrix**: A symmetric matrix $K \in \mathbb{R}^{N \times N}$ is positive semi-definite if $c^T K c \geq 0 \; \forall c \in \mathbb{R}^N$. This is equivalent to all of its eigenvalues being non-negative.
- **Positive definite function**: A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called positive definite if for any $c_1, \dots, c_N \in \mathbb{R}$, any $x_1, \dots, x_N \in \mathcal{X}$, and any $N \in \mathbb{N}$, $\sum_{i,j=1}^{N} c_i c_j k(x_i, x_j) \geq 0$.

Appendix B

Derivation for random Fourier features

In this appendix, we will prove that for a properly-scaled (i.e., $Z = 1$) positive-definite shift-invariant kernel k,

$$k(x,y) = \mathbb{E}_{\omega,b}\left[\sqrt{2}\cos(\omega^T x + b)\cdot\sqrt{2}\cos(\omega^T y + b)\right], \tag{B.1}$$

where ω is drawn from $p(\omega)$, the inverse Fourier transform of k, and b is drawn uniformly from $[0, 2\pi]$. We begin this proof using Equation 2.5 from Section 2.3.1:

$$\begin{aligned}
k(x,y) &= \int_{R^d} p(\omega)e^{j\omega^T(x-y)}\,d\omega \\
&= \mathbb{E}_\omega\left[e^{j\omega^T x}e^{-j\omega^T y}\right] \\
&= \mathbb{E}_\omega\left[\big(\cos(\omega^T x) + j\sin(\omega^T x)\big)\big(\cos(\omega^T y) - j\sin(\omega^T y)\big)\right] \\
&= \mathbb{E}_\omega\left[\cos(\omega^T x)\cos(\omega^T y) + \sin(\omega^T x)\sin(\omega^T y)\right] \\
&\quad + j\cdot\mathbb{E}_\omega\left[\sin(\omega^T x)\cos(\omega^T y) - \sin(\omega^T y)\cos(\omega^T x)\right] \\
&= \mathbb{E}_\omega\left[\cos(\omega^T x)\cos(\omega^T y) + \sin(\omega^T x)\sin(\omega^T y)\right]
\end{aligned} \tag{B.2}$$

Note the Equation B.2 is true because we know that $k(x,y)$ is a real-valued function, and thus the imaginary part of the expectation must disappear. We now show that the right-hand side of Equation

B.1 is equal to this same expression:

$$
\begin{aligned}
&\mathbb{E}_{\omega,b}\left[\sqrt{2}\cos(\omega^T x + b)\cdot\sqrt{2}\cos(\omega^T y + b)\right]\\
&\qquad = 2\cdot\mathbb{E}_{\omega,b}\Bigg[\bigg(\cos(\omega^T x)\cos(b) - \sin(\omega^T x)\sin(b)\bigg)\cdot\\
&\qquad\qquad\bigg(\cos(\omega^T y)\cos(b) - \sin(\omega^T y)\sin(b)\bigg)\Bigg] \qquad \text{(B.3)}\\
&\qquad = 2\cdot\mathbb{E}_{\omega,b}\Bigg[\cos(\omega^T x)\cos(\omega^T y)\cos^2(b)\\
&\qquad\qquad - \cos(\omega^T x)\sin(\omega^T y)\cos(b)\sin(b)\\
&\qquad\qquad - \sin(\omega^T x)\cos(\omega^T y)\cos(b)\sin(b)\\
&\qquad\qquad + \sin(\omega^T x)\sin(\omega^T y)\sin^2(b)\Bigg]\\
&\qquad = 2\cdot\mathbb{E}_{\omega}\left[\frac{1}{2}\cos(\omega^T x)\cos(\omega^T y) + \frac{1}{2}\sin(\omega^T x)\sin(\omega^T y)\right] \qquad \text{(B.4)}\\
&\qquad = \mathbb{E}_{\omega}\left[\cos(\omega^T x)\cos(\omega^T y) + \sin(\omega^T x)\sin(\omega^T y)\right]\\
&\qquad = k(x,y)
\end{aligned}
$$

Equation B.3 is true by the cosine sum of angles formula, and Equation B.4 is true because $\mathbb{E}_b\left[\cos^2(b)\right] = \mathbb{E}_b\left[\sin^2(b)\right] = \int_0^{2\pi}\frac{1}{2\pi}\sin^2(b) = \frac{1}{2}$, and because $\mathbb{E}_b\left[\sin(b)\cos(b)\right] = 0$. This concludes the proof.

Appendix C

Detailed results

C.1 Results from Section 4

In this section, we include tables comparing the models we trained in terms of 4 different metrics (CE, ENT, ERR, and ERP). The notation is the same as in Tables 4.3 and 4.4. For both DNN and kernel models, 'NT' specifies that no tricks were used during training (no bottleneck, no special learning rate decay). A 'B' specifies that a linear bottleneck was used for the output matrix, while an 'R' specifies that entropy regularized perplexity was used for learning rate decay.

	1000				2000				4000			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	1.25	1.26	1.24	1.27	**1.24**	1.26	1.26	1.32	1.24	1.25	1.30	1.39
BN-50	2.05	2.05	2.04	2.08	2.01	2.04	2.05	2.22	**2.00**	2.03	2.09	2.27
Cant.	1.92	1.96	1.92	1.98	1.93	1.94	1.97	2.06	**1.92**	1.97	2.03	2.10
TIMIT	**1.06**	1.08	1.20	1.28	1.08	1.09	1.25	1.31	1.10	1.11	1.25	1.33

Table C.1: DNN: Metric CE

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	1.34	1.32	1.35	1.39	1.35	1.33	1.36	1.34	1.31	**1.29**	1.34	1.33
BN-50	N/A	2.15	N/A	2.43	N/A	2.05	N/A	2.16	N/A	**2.05**	N/A	2.19
Cant.	**1.93**	1.95	1.95	2.04	1.99	1.98	2.00	2.04	1.93	1.94	1.95	2.00
TIMIT	0.97	0.99	0.97	1.07	0.94	0.96	0.94	1.02	0.94	0.95	**0.94**	1.03

Table C.2: Kernel: Metric CE

	1000				2000				4000			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	1.23	1.17	1.18	1.09	1.18	1.13	1.09	0.99	1.14	1.11	1.02	**0.91**
BN-50	1.95	1.77	1.90	1.68	1.76	1.68	1.65	1.40	1.65	1.60	1.48	**1.27**
Cant.	1.71	1.67	1.67	1.57	1.66	1.64	1.55	1.42	1.63	1.55	1.43	**1.38**
TIMIT	0.72	0.70	0.58	0.53	0.63	0.63	0.50	0.48	0.57	0.57	0.48	**0.45**

Table C.3: DNN: Metric ENT

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	1.43	1.23	1.41	**1.08**	1.36	1.31	1.35	1.28	1.35	1.23	1.30	1.10
BN-50	N/A	1.89	N/A	**1.46**	N/A	1.83	N/A	1.53	N/A	1.81	N/A	1.48
Cant.	1.84	1.67	1.76	**1.52**	1.94	1.73	1.91	1.58	1.77	1.69	1.70	1.55
TIMIT	0.95	0.72	0.91	0.61	0.88	0.73	0.86	0.62	0.89	0.76	0.85	**0.61**

Table C.4: Kernel: Metric ENT

	1000				2000				4000			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	2.48	2.43	2.43	2.37	2.42	2.39	2.35	2.31	2.39	2.37	2.32	**2.30**
BN-50	3.99	3.82	3.94	3.76	3.77	3.72	3.70	3.63	3.65	3.63	3.58	**3.55**
Cant.	3.63	3.63	3.58	3.56	3.59	3.58	3.51	3.48	3.55	3.52	**3.46**	3.47
TIMIT	1.77	1.77	1.77	1.81	1.71	1.72	1.76	1.79	**1.67**	1.68	1.73	1.78

Table C.5: DNN: Metric ERP

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	2.77	2.55	2.76	2.47	2.71	2.65	2.71	2.62	2.67	2.52	2.64	**2.44**
BN-50	N/A	4.04	N/A	3.88	N/A	3.88	N/A	3.69	N/A	3.86	N/A	**3.67**
Cant.	3.77	3.62	3.71	3.56	3.94	3.71	3.91	3.62	3.71	3.63	3.65	**3.54**
TIMIT	1.92	1.71	1.87	1.68	1.82	1.70	1.80	1.65	1.83	1.71	1.79	**1.64**

Table C.6: Kernel: Metric ERP

	1000				2000				4000			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	0.29	0.29	0.29	0.29	**0.29**	0.29	0.29	0.30	0.29	0.29	0.29	0.30
BN-50	0.50	0.50	0.50	0.50	0.49	0.50	0.50	0.51	**0.49**	0.49	0.50	0.51
Cant.	0.44	0.44	**0.44**	0.44	0.44	0.44	0.44	0.44	0.44	0.44	0.44	0.44
TIMIT	0.33	0.33	0.34	0.34	0.33	0.33	0.34	0.34	0.33	**0.32**	0.33	0.33

Table C.7: DNN: Metric ERR

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	0.30	0.30	0.31	0.31	0.31	0.31	0.31	0.31	0.30	**0.30**	0.30	0.30
BN-50	N/A	0.52	N/A	0.54	N/A	**0.50**	N/A	0.51	N/A	0.50	N/A	0.51
Cant.	**0.43**	0.44	0.44	0.44	0.45	0.45	0.45	0.45	0.43	0.44	0.44	0.44
TIMIT	0.32	0.32	0.32	0.33	0.31	0.32	0.31	0.32	0.31	0.31	**0.31**	0.32

Table C.8: Kernel: Metric ERR

C.2 Results from Section 5

In this section, we include tables comparing the kernel models we trained in terms of 4 different metrics (CE, ENT, ERR, and ERP). The notation is the same as Table 5.1. ‘NT’ specifies that no tricks were used during training (no bottleneck, no feature selection, no special learning rate decay). A ‘B’ specifies that a linear bottleneck was used for the output matrix, while an ‘R’ specifies that ERP was used for learning rate decay (‘BR’ means both were used). ‘+FS’ specifies that feature selection was used for the experiments in that row.

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	1.34	1.32	1.35	1.39	1.35	1.33	1.36	1.34	1.31	**1.29**	1.34	1.33
+FS	1.28	**1.26**	1.29	1.27	1.35	1.31	1.36	1.35	1.28	1.26	1.31	1.27
BN-50	N/A	2.15	N/A	2.43	N/A	2.05	N/A	2.16	N/A	**2.05**	N/A	2.19
+FS	N/A	2.01	N/A	2.07	N/A	2.04	N/A	2.13	N/A	**2.00**	N/A	2.06
Cant.	**1.93**	1.95	1.95	2.04	1.99	1.98	2.00	2.04	1.93	1.94	1.95	2.00
+FS	**1.88**	1.90	1.89	1.95	1.97	1.97	1.98	2.03	1.90	1.91	1.91	1.96
TIMIT	0.97	0.99	0.97	1.07	0.94	0.96	0.94	1.02	0.94	0.95	**0.94**	1.03
+FS	0.92	0.95	0.92	1.03	0.93	0.96	0.93	1.02	**0.92**	0.96	0.92	1.03

Table C.9: Kernel: Metric CE

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	1.43	1.23	1.41	**1.08**	1.36	1.31	1.35	1.28	1.35	1.23	1.30	1.10
+FS	1.32	1.21	1.28	1.14	1.44	1.27	1.45	**1.13**	1.32	1.22	1.26	1.14
BN-50	N/A	1.89	N/A	**1.46**	N/A	1.83	N/A	1.53	N/A	1.81	N/A	1.48
+FS	N/A	1.81	N/A	1.56	N/A	1.84	N/A	**1.55**	N/A	1.80	N/A	1.57
Cant.	1.84	1.67	1.76	**1.52**	1.94	1.73	1.91	1.58	1.77	1.69	1.70	1.55
+FS	1.75	1.66	1.73	1.54	1.91	1.72	1.87	1.57	1.75	1.68	1.72	**1.54**
TIMIT	0.95	0.72	0.91	0.61	0.88	0.73	0.86	0.62	0.89	0.76	0.85	**0.61**
+FS	0.86	0.70	0.82	0.58	0.86	0.70	0.83	0.61	0.84	0.69	0.82	**0.58**

Table C.10: Kernel: Metric ENT

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	2.77	2.55	2.76	2.47	2.71	2.65	2.71	2.62	2.67	2.52	2.64	**2.44**
+FS	2.60	2.47	2.57	**2.41**	2.79	2.58	2.80	2.48	2.60	2.48	2.57	2.41
BN-50	N/A	4.04	N/A	3.88	N/A	3.88	N/A	3.69	N/A	3.86	N/A	**3.67**
+FS	N/A	3.82	N/A	3.63	N/A	3.88	N/A	3.67	N/A	3.80	N/A	**3.62**
Cant.	3.77	3.62	3.71	3.56	3.94	3.71	3.91	3.62	3.71	3.63	3.65	**3.54**
+FS	3.63	3.56	3.63	**3.49**	3.88	3.69	3.86	3.60	3.64	3.58	3.63	3.50
TIMIT	1.92	1.71	1.87	1.68	1.82	1.70	1.80	1.65	1.83	1.71	1.79	**1.64**
+FS	1.78	1.65	1.74	1.61	1.79	1.67	1.76	1.64	1.76	1.64	1.74	**1.61**

Table C.11: Kernel: Metric ERP

	Laplacian				Gaussian				Sparse Gaussian			
	NT	B	R	BR	NT	B	R	BR	NT	B	R	BR
Beng.	0.30	0.30	0.31	0.31	0.31	0.31	0.31	0.31	0.30	**0.30**	0.30	0.30
+FS	0.29	**0.29**	0.30	0.29	0.31	0.31	0.31	0.31	0.30	0.29	0.30	0.30
BN-50	N/A	0.52	N/A	0.54	N/A	**0.50**	N/A	0.51	N/A	0.50	N/A	0.51
+FS	N/A	0.49	N/A	0.50	N/A	0.50	N/A	0.50	N/A	**0.49**	N/A	0.49
Cant.	**0.43**	0.44	0.44	0.44	0.45	0.45	0.45	0.45	0.43	0.44	0.44	0.44
+FS	**0.43**	0.43	0.43	0.44	0.44	0.44	0.44	0.45	0.43	0.44	0.43	0.44
TIMIT	0.32	0.32	0.32	0.33	0.31	0.32	0.31	0.32	0.31	0.31	**0.31**	0.32
+FS	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.32	0.31	0.31	**0.31**	0.31

Table C.12: Kernel: Metric ERR

Appendix D

Nyström Appendix

D.1 Datasets

For the TIMIT dataset, we use the same exact training/heldout/dev/test sets as described in Section 4.2. We acquired the Cod-RNA, CovType (binary), and YearPred from the LIBSVM webpage,[1] and the Adult, Census, CPU, and Forest datasets from Ali Rahimi's webpage.[2] For these seven datasets, we randomly set aside 10% of the training data as a heldout set for tuning the learning rate and kernel bandwidth.

The specific files we used were as follows:

- Cod-RNA: We used "cod-rna" as training/heldout set, "cod-rna.t" file as test set.
- CovType: We randomly chose 20% of "covtype.libsvm.binary" as test, and used the rest for training/heldout.
- YearPred: We used "YearPredictionMSD" as training/heldout set, and "YearPredictionMSD.t" as test.
- For the Adult, Census, CPU, and Forest datasets which we downloaded from Ali Rahimi's webpage, we used the included matlab dataset files (adult.mat, census.mat, cpu.mat, forest.mat). These Matlab files each had already split the data into train and test. We used a

[1]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

[2]https://keysduplicated.com/ ali/random-features/data/

random 10% of the training data as heldout.

D.2 Hyperparameter Choices

We tune the initial learning rates, as well as the kernel bandwidths, on the heldout sets. In Table D.1, we report for each dataset, the initial learning rate we use, as well as the kernel bandwidth value (we report the value of $1/2\sigma^2$ which we use).

Dataset	$1/2\sigma^2$	Initial LR
ADULT	0.1	100
COD-RNA	0.4	200
COVTYPE	0.6	10
FOREST	0.5	1000
TIMIT	0.0015	51.2
CENSUS	0.0006	1
CPU	0.03	10
YEARPRED	0.01	0.6

Table D.1: Hyperparameters used for all datasets

D.3 Results

In the main body of the paper, we only included results on TIMIT, Forest, and YearPred, for the sake of clarity. We now include results for all eight of our datasets. In Figure D.1, we show the kernel approximation performance for Nyström and RFF features as a function of the number of features. In Figure D.2 we show kernel approximation performance as a function of total memory requirement. In Figures D.3 and D.4, we show heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the total numbers of features (left), total memory requirement (middle), and kernel approximation error (right) of the corresponding models. For Nyström experiments with $D \leq 2500$, and RFF experiments with $D \leq 20000$, we

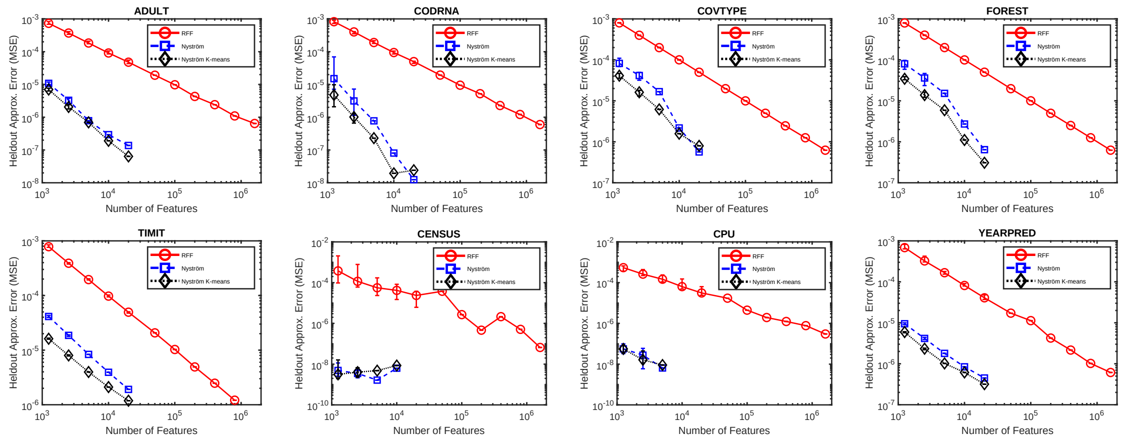

Figure D.1: Kernel approximation error, in terms of the number of features.

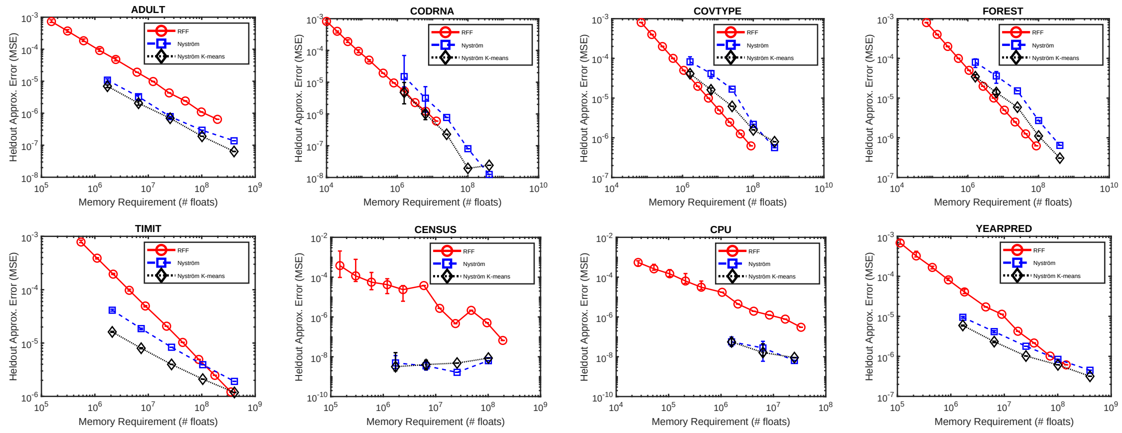

Figure D.2: Kernel approximation error, in terms of the total memory requirement.

run the experiments 10 times, and report the median performance, with error bars indicating the minimum and maximum. Note that due to small variance, error bars are often not clearly visible. In Figure 6.2 we show the spectra for each dataset of a kernel matrix generated from 20k random training points. In Figures D.6 and D.7, we show heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the average kernel approximation error, as measured by $|k(x,x') - z(x)^T z(x')|^r$, for various values of r ($r \in \{2.5, 3.5, 5.5\}$).

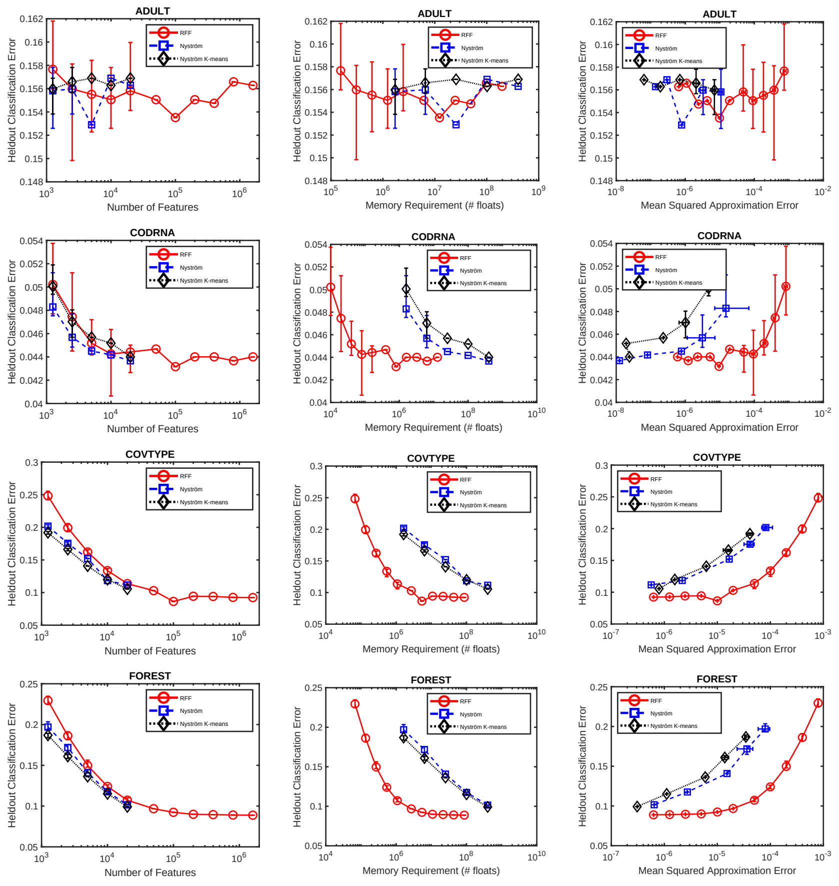

Figure D.3: Heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the total numbers of features (left), total memory requirement (middle), and kernel approximation error (right) of the corresponding models. Results reported on Adult, Cod-RNA, CovType, and Forest.

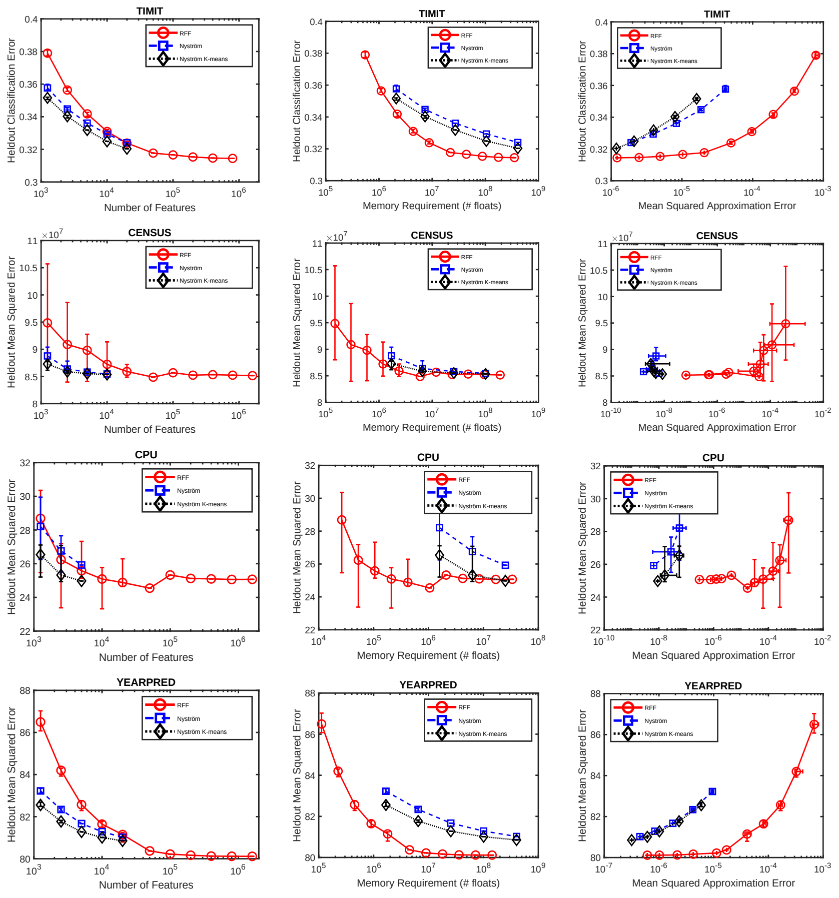

Figure D.4: Heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the total numbers of features (left), total memory requirement (middle), and kernel approximation error (right) of the corresponding models. Results reported on TIMIT, Census, CPU, and YearPred.

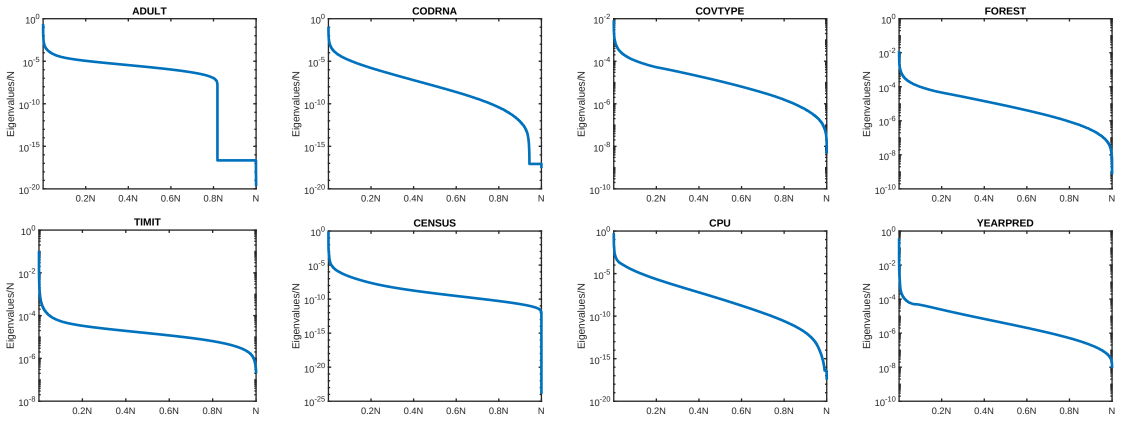

Figure D.5: Spectrum of kernel matrices generated from $N = 20k$ random training points.

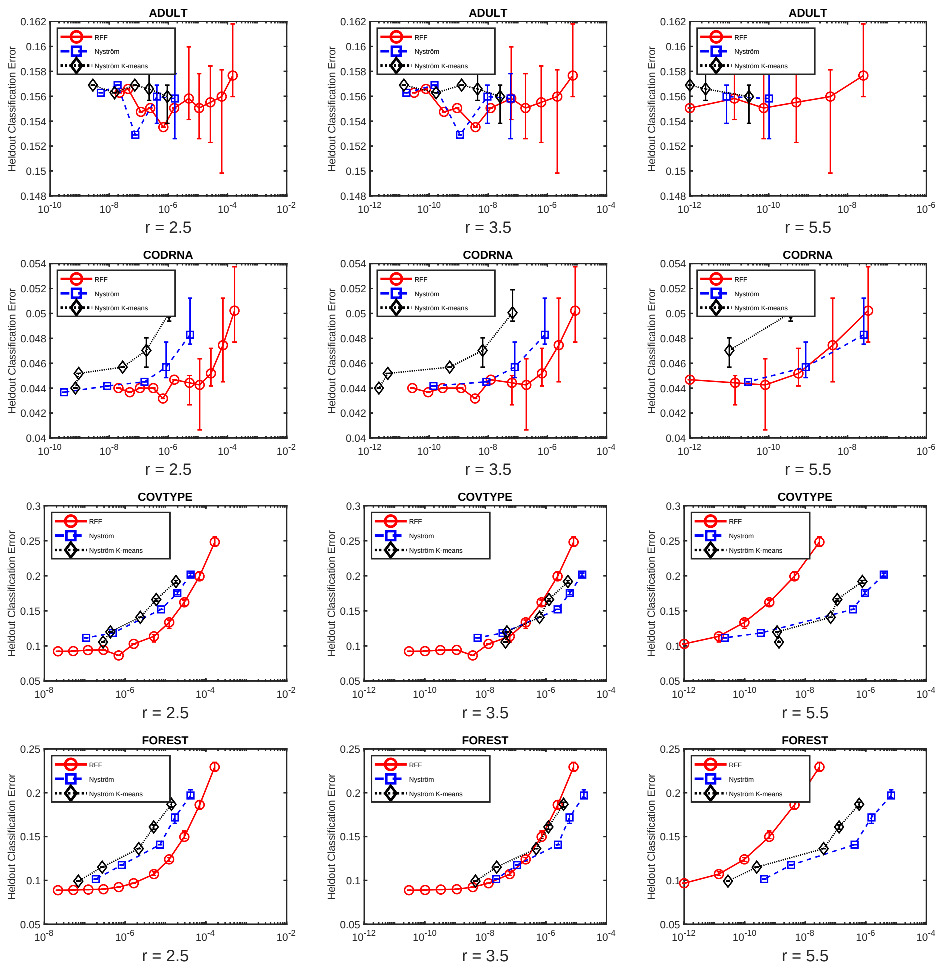

Figure D.6: Heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the average kernel approximation errors, measured as $|k(x, y) - z(x)^T z(y)|^r$ for $r \in \{2.5, 3.5, 5.5\}$. Note that due to numeric underflow, some of the models with lowest approximation error sometimes do not appear in the plots. Results reported on Adult, Cod-RNA, CovType, and Forest.

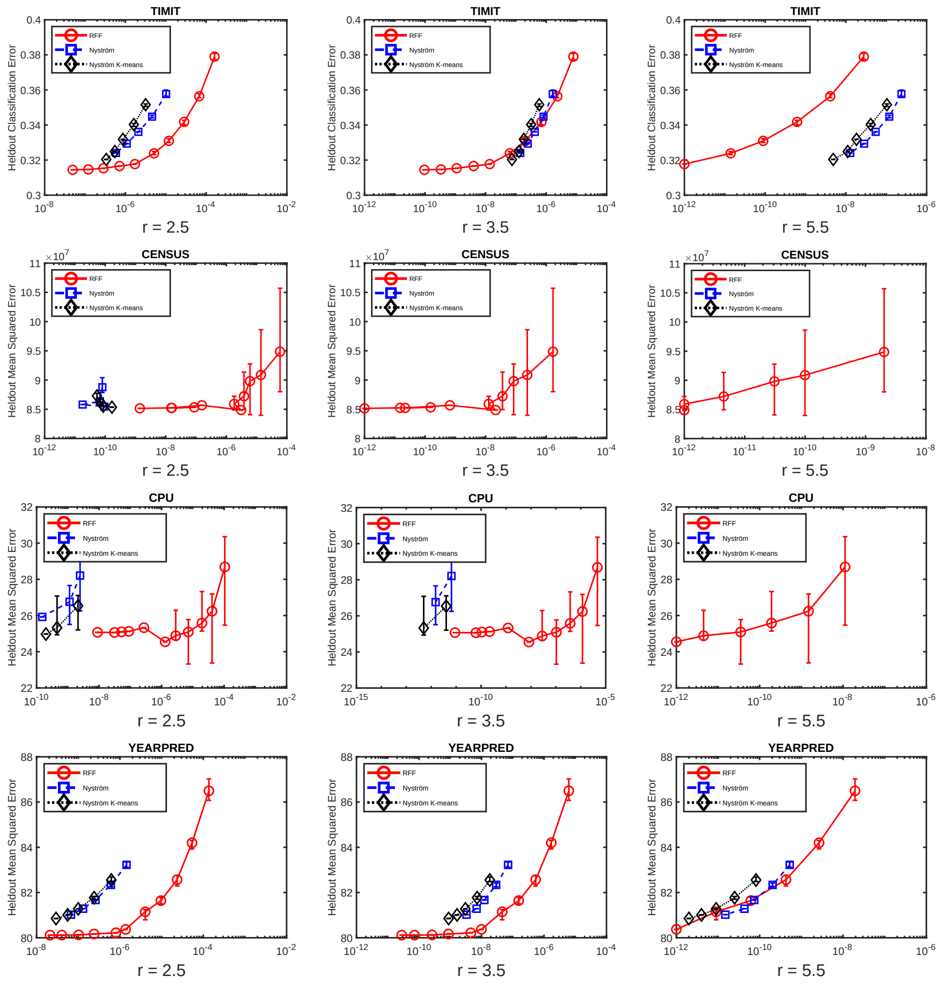

Figure D.7: Heldout classification or regression performance for the Nyström method vs. random Fourier features, in terms of the average kernel approximation errors, measured as $|k(x, y) - z(x)^T z(y)|^r$ for $r \in \{2.5, 3.5, 5.5\}$. Note that due to numeric underflow, some of the models with lowest approximation error sometimes do not appear in the plots. Results reported on TIMIT, Census, CPU, and YearPred.

D.4 Background for Proofs

D.4.1 Definitions of a couple infinite dimensional Hilbert Spaces

Here we review the definitions of the Hilbert spaces $\ell_2(J)$ and $L_2(\mathcal{X};\mu)$, with their respective inner products. Here, we assume that $\mathcal{X}$ is a compact metric space, that J is a countable index set, and that μ is a positive measure on $\mathcal{X}$. We now define these Hilbert Spaces:

$$\begin{aligned}
\ell_2(J) &= \left\{ (a_j)_{j\in J} \,\middle|\, \sum_{j\in J} a_j^2 < \infty, a_j \in \mathbb{C} \right\} \\
\langle (a_j),(b_j)\rangle_{\ell_2(J)} &= \sum_{j\in J} a_j \overline{b_j} \\
L_2(\mathcal{X},\mu) &= \left\{ f : \mathcal{X} \to \mathbb{C} \text{ measurable} \,\middle|\, \int_{\mathcal{X}} |f(x)|^2 d\mu < \infty \right\} \\
\langle f,g\rangle_{L_2} &= \int_{\mathcal{X}} f(x)\overline{g(x)} d\mu
\end{aligned}$$

Note that in the definitions above, we used the complex numbers as the field of interest, using $\overline{a}$ to denote the complex conjugate of a complex number a, and using $|a|^2 = a\overline{a} = \langle a,a\rangle_{\mathbb{C}}$. The equivalent Hilbert spaces can also be defined over the real numbers, by simply restricting $a_j \in \mathbb{R}$ and $f(x) \in \mathbb{R}$; in the remainder of this appendix, we will assume we are working in the *real* versions of these Hilbert spaces. Also, to be precise, the elements of $L_2(\mathcal{X};\mu)$ are equivalence classes of functions which differ on at most a set of measure zero. This ensures that $\|f_1 - f_2\|_{L_2} = 0 \Leftrightarrow f_1 = f_2$. Generally we will take the measure μ to be a probability density function p over the input space $\mathcal{X}$. When μ is the standard Lebesgue measure, we simply write $L_2(\mathcal{X})$ instead of $L_2(\mathcal{X};\mu)$.

D.4.2 Review of Reproducing Kernel Hilbert Space Definitions

In this appendix, we will be working with two separate (but equivalent) representations for the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ corresponding to a kernel function. The first is the Moore-Aronsajn RKHS construction, as discussed in Section 2.4. We will define $\varphi(x) \equiv k(\cdot,x)$, so that $\langle \varphi(x),\varphi(y)\rangle = k(x,y)$.

The second representation is given by Mercer's Theorem. Mercer's Theorem states that $k(x,y) = \sum_{j=1}^{N} \lambda_j e_j(x)e_j(y) = \left\langle \sqrt{\lambda_j}e_j(x), \sqrt{\lambda_j}e_j(y)\right\rangle_{\ell_2([N])}$, where the λ_j and e_j are the (strictly positive) eigenvalues and eigenfunctions, respectively, of the linear operator $L : L_2(\mathcal{X}; p) \to L_2(\mathcal{X}; p)$ defined as $L[f](y) = \int_{x\in\mathcal{X}} k(x,y)f(x)p(x)dx$. We will define $\varphi'(x) \equiv (\sqrt{\lambda_j}e_j(x))_{j=1}^N$, so that $\langle\varphi'(x), \varphi'(y)\rangle = k(x,y)$. We will assume $\lambda_1 \geq \lambda_2 \geq \ldots > 0$.

D.5 Proofs: Nyström Background Section

Proof of Claim 1: Let $\hat{K} = U\Lambda U^T$ be the SVD of the landmark kernel matrix. Let $\lambda_i^{(m)}$ denote the i^{th} element on the diagonal of Λ, and let $U_i^{(m)}$ denote the i^{th} column of U, and let $U_{k,i}^{(m)}$ be the k^{th} element in this column. Consider the set $S = \{\hat{\phi}_1(\cdot), \ldots, \hat{\phi}_m(\cdot)\} \in \mathcal{H}$ of elements in the RKHS $\mathcal{H}$ corresponding to the kernel k, where $\hat{\phi}_i(\cdot) = \frac{1}{\sqrt{\lambda_i^{(m)}}}\sum_{k=1}^m U_{k,i}^{(m)} k(\hat{x}_k, \cdot)$.

We will first show that this set S is an orthonormal basis of the subspace $A = span(\{k(\hat{x}_1,\cdot),\ldots,k(\hat{x}_m,\cdot)\}) \subset \mathcal{H}$, and we will then show that the Nyström method can be understood as performing a projection onto this subspace A.

In order to prove that the elements of S are orthonormal, we consider $\left\langle \hat{\phi}_i(\cdot), \hat{\phi}_j(\cdot)\right\rangle_{\mathcal{H}}$:

$$\begin{aligned}
\left\langle \hat{\phi}_i(\cdot), \hat{\phi}_j(\cdot)\right\rangle_{\mathcal{H}} &= \left\langle \frac{1}{\sqrt{\lambda_i^{(m)}}}\sum_{k=1}^m U_{k,i}^{(m)} k(\hat{x}_k,\cdot), \frac{1}{\sqrt{\lambda_j^{(m)}}}\sum_{l=1}^m U_{l,j}^{(m)} k(\hat{x}_l,\cdot)\right\rangle \\
&= \frac{1}{\sqrt{\lambda_i^{(m)}\lambda_j^{(m)}}}\sum_{k,l=1}^m U_{k,i}^{(m)}U_{l,j}^{(m)} k(\hat{x}_k,\hat{x}_l) \\
&= \frac{1}{\sqrt{\lambda_i^{(m)}\lambda_j^{(m)}}}[U^T\hat{K}U]_{i,j} \\
&= \frac{1}{\sqrt{\lambda_i^{(m)}\lambda_j^{(m)}}}\Lambda_{i,j} \\
&= 1 \text{ if } i=j, 0 \text{ otherwise.}
\end{aligned}$$

This proves that the elements of S are an orthonormal set. Because all the elements of S are in the $A = span(\{k(\hat{x}_1,\cdot),\ldots,k(\hat{x}_m,\cdot)\})$, and because the size of S matches the dimension of A, this proves that the set S is an orthonormal basis of A.

We will now show that the Nyström representation $z(x)$ corresponds to performing a projection of $k(x,\cdot)$ onto the subspace A. Let $P_A : \mathcal{H} \to A$ denote the projection operator onto the subspace

A of $\mathcal{H}$. Note that $P_A(k(x,\cdot)) = \sum_{i=1}^m \left\langle \hat{\phi}_i(\cdot), k(x,\cdot)\right\rangle_{\mathcal{H}} \hat{\phi}_i(\cdot)$. Treating S as the "standard basis" of A, we can consider the isometric isomorphism $\psi : A \to \mathbb{R}^m$, defined as $\psi(\sum_{i=1}^m a_i\hat{\phi}_i(\cdot)) = [a_1,\ldots,a_m]^T$. Thus, $\psi(P_A(k(x,\cdot))) = \left[\left\langle \hat{\phi}_1(\cdot), k(x,\cdot)\right\rangle_{\mathcal{H}},\ldots,\left\langle \hat{\phi}_m(\cdot), k(x,\cdot)\right\rangle_{\mathcal{H}}\right]$. We can now consider $\left\langle \hat{\phi}_i(\cdot), k(x,\cdot)\right\rangle_{\mathcal{H}}$:

$$\begin{aligned}
\left\langle \hat{\phi}_i(\cdot), k(x,\cdot)\right\rangle_{\mathcal{H}} &= \left\langle \frac{1}{\sqrt{\lambda_i^{(m)}}}\sum_{k=1}^m U_{k,i}^{(m)} k(\hat{x}_k,\cdot), k(x,\cdot)\right\rangle \\
&= \frac{1}{\sqrt{\lambda_i^{(m)}}}\sum_{k=1}^m U_{k,i}^{(m)} k(\hat{x}_k,x) \\
&= \frac{1}{\sqrt{\lambda_i^{(m)}}} U_i^{(m)T} k_x \\
&= \left[\Lambda^{-1/2} U^T k_x\right]_i \\
&= z_i(x)
\end{aligned}$$

Thus, $\psi(P_A(k(x,\cdot))) = z(x)$, which proves that the Nyström representation corresponds to projecting the (potentially) infinite dimensional element $k(x,\cdot) \in \mathcal{H}$ onto the finite dimensional subspace A.

Proof of Corollary 1: Clearly $\varphi(x) = \varphi_A(x) + \varphi_{A^\perp}(x)$, and by the Pythagorean theorem $\|\varphi(x)\|_{\mathcal{H}}^2 = \|\varphi_A(x)\|_{\mathcal{H}}^2 + \|\varphi_{A^\perp}(x)\|_{\mathcal{H}}^2 = 1$. Thus,

$$\begin{aligned}
k(x,y) &= \langle \varphi(x), \varphi(y)\rangle_{\mathcal{H}} \\
&= \langle \varphi_A(x) + \varphi_{A^\perp}(x), \varphi_A(y) + \varphi_{A^\perp}(y)\rangle_{\mathcal{H}} \\
&= \langle \varphi_A(x), \varphi_A(y)\rangle_{\mathcal{H}} + \langle \varphi_{A^\perp}(x), \varphi_{A^\perp}(y)\rangle_{\mathcal{H}} \\
&= \langle z(x), z(y)\rangle_{\ell_2} + \langle \varphi_{A^\perp}(x), \varphi_{A^\perp}(y)\rangle_{\mathcal{H}} \\
\Rightarrow \quad k(x,y) - \langle z(x), z(y)\rangle_{\ell_2} &= \langle \varphi_{A^\perp}(x), \varphi_{A^\perp}(y)\rangle_{\mathcal{H}}
\end{aligned}$$

Thus, the error Nyström makes in predicting $k(x,y)$ is precisely the dot-product of the components of $\varphi(x)$ and $\varphi(y)$ which are orthogonal to the subspace A.

Proof of Corollary 2: This follows directly from the above corollary.

$$\begin{aligned} k(x,x) - \|z(x)\|^2_{\ell_2} &= \|\varphi_{A^\perp}(x)\|^2_{\mathcal{H}} \\ \Rightarrow \quad \|z(x)\|^2_{\ell_2} &= k(x,x) - \|\varphi_{A^\perp}(x)\|^2_{\mathcal{H}} \\ &\leq k(x,x) \end{aligned}$$

Thus, Nyström systematically *underestimates* "self-similarity" $k(x,x)$.

Proof of Corollary 3: This follows directly from Corollary 1. If $k(x,\cdot) \in A$, then $\varphi_{A^\perp}(x) = \mathbf{0}$, and thus

$$\begin{aligned} k(x,y) - \langle z(x), z(y)\rangle_{\ell_2} &= \langle \varphi_{A^\perp}(x), \varphi_{A^\perp}(y)\rangle_{\mathcal{H}} \\ &= \langle \mathbf{0}, \varphi_{A^\perp}(y)\rangle_{\mathcal{H}} \\ &= 0 \\ \Rightarrow \quad k(x,y) &= \langle z(x), z(y)\rangle_{\ell_2} \end{aligned}$$

D.6 Proofs: Nyström Error Analysis

D.6.1 Theorem 4

First, we give the following proof sketch: Let $\tilde{k}(x,y) = \langle z(x), z(y)\rangle$, $R(x,\epsilon) = \frac{k(x,x)-\tilde{k}(x,x)-\epsilon}{C(\lambda_r^{-1}m+1)}$, and let $B(x,R)$ be the *open* ball of radius R around x. First we show that if $\|x-y\|_2 < R(x,\epsilon)$, then $|k(x,y) - \tilde{k}(x,y)| > \epsilon$ (part 1). We then show that if this same condition holds it must also be true that $k(x,y) - \tilde{k}(x,y) > 0$ (part 2), which proves that it must be true that $k(x,y) - \tilde{k}(x,y) > \epsilon$. In order to prove the first part, we use the triangle inequality to show that $|\tilde{k}(x,y) - k(x,y)| \geq |\tilde{k}(x,x) - k(x,x)| - |k(x,y) - k(x,x)| - |\tilde{k}(x,x) - \tilde{k}(x,y)|$. We then upper bound $|k(x,y) - k(x,x)|$ and $|\tilde{k}(x,x) - \tilde{k}(x,y)|$ using the fact that both k and $\tilde{k}$ are Lipschitz continuous functions (in each of their arguments). In order to prove the second part we show that it cannot be true that there exist two points y_1, y_2 within a distance ϵ of x such that $k(x,y_1) - \tilde{k}(x,y_1) > \epsilon$ and $k(x,y_2) - \tilde{k}(x,y_2) < -\epsilon$. Then we note that whenever $R(x,\epsilon) > 0$, it must be that $y = x$ satisfies $\|x-y\| = 0 < R(x,\epsilon)$, and thus that $k(x,y) - \tilde{k}(x,y) > \epsilon$. As a result, all $y \in B(x, \mathbb{R}(x,\epsilon))$ must satisfy $k(x,y) - \tilde{k}(x,y) > \epsilon$. We now dive into the full proof.

We will first prove that when $\|x-y\|_2 < R(x,\epsilon)$ holds, that $|k(x,y) - z_r(x)^T z_r(y)| > \epsilon$. Let $\tilde{k}(x,y) = \langle z_r(x), z_r(y)\rangle$. Let $a_1 = \|z_r(x)\|^2$, $a_2 = |\tilde{k}(x,x) - \tilde{k}(x,y)|$, and $a_3 = |k(x,y) - k(x,x)|$. By the triangle inequality we have:

$$\begin{aligned} |\tilde{k}(x,x) - k(x,x)| &\leq |\tilde{k}(x,x) - \tilde{k}(x,y)| + |\tilde{k}(x,y) - k(x,y)| + |k(x,y) - k(x,x)| \\ k(x,x) - a_1 &\leq a_2 + |\tilde{k}(x,y) - k(x,y)| + a_3 \\ \Rightarrow \quad |\tilde{k}(x,y) - k(x,y)| &\geq k(x,x) - a_1 - a_2 - a_3 \end{aligned}$$

So if $a_1 + a_2 + a_3 < k(x,x)$, then the Nyström method makes a (potentially very large) error estimating $k(x,y)$. a_2 and a_3 can be shown to be small by the fact that k and $\tilde{k}$ are continuous functions (in each of their entries). We will show this now.

If follows immediately from our assumption that all function of the form $k(x,\cdot)$ are C-Lipschitz that $a_3 = |k(x,x) - k(x,y)| \leq C\|x-y\|$. Now, we must simply bound a_2. In Lemma 1 below we show that $a_2 = |\tilde{k}(x,x) - \tilde{k}(x,y)| \leq \lambda_r^{-1} mC\|x-y\|$.

Thus, it immediately follows that:

$$\begin{aligned} |\tilde{k}(x,y) - k(x,y)| &\geq k(x,x) - \tilde{k}(x,x) - \lambda_r^{-1} mC\|x-y\| - C\|x-y\| \\ &= k(x,x) - \tilde{k}(x,x) - C\|x-y\|(\lambda_r^{-1} m + 1) \end{aligned}$$

Thus, if $k(x,x) - \tilde{k}(x,x) - C\|x-y\|(\lambda_r^{-1} m + 1) > \epsilon$, it must follow that $|\tilde{k}(x,y) - k(x,y)| > \epsilon$. We now rearrange the condition under which this holds.

$$\begin{aligned} k(x,x) - \tilde{k}(x,x) - C\|x-y\|(\lambda_r^{-1} m + 1) &> \epsilon \\ \Leftrightarrow \quad -C\|x-y\|(\lambda_r^{-1} m + 1) &> \epsilon - k(x,x) + \tilde{k}(x,x) \\ \Leftrightarrow \quad C\|x-y\|(\lambda_r^{-1} m + 1) &< k(x,x) - \tilde{k}(x,x) - \epsilon \\ \Leftrightarrow \quad \|x-y\| &< \frac{k(x,x) - \tilde{k}(x,x) - \epsilon}{C(\lambda_r^{-1} m + 1)} \end{aligned}$$

We will now prove that under this same condition on $\|x-y\|_2$, it must also be true that $k(x,y) - \tilde{k}(x,y) > 0$. This would mean that $k(x,y) - \tilde{k}(x,y) = |k(x,y) - \tilde{k}(x,y)| > \epsilon$ when these conditions holds. Thus, we are not only guaranteed that Nyström makes a large mistake on this x, y pair, but we know that Nyström will be *underestimating* the true value of $k(x,y)$.

Let $R(x,\epsilon) = \frac{k(x,x) - \tilde{k}(x,x) - \epsilon}{C(\lambda_r^{-1} m + 1)}$. We have already proven that all $y \in B(x, R(x,\epsilon))$ (the *open* ball of radius $R(x,\epsilon)$ around x) must satisfy $|k(x,y) - \tilde{k}(x,y)| > \epsilon$. Assume there exists a point

$y_1 \in B(x, R(x, \epsilon))$ satisfying $k(x, y_1) - \tilde{k}(x, y_1) > \epsilon$, and another point $y_2 \in B(x, R(x, \epsilon))$ for which $k(x, y_2) - \tilde{k}(x, y_2) < -\epsilon$. This immediately yields a contradiction because due to the continuity of the function $k(x, y) - \tilde{k}(x, y)$ in the variable y, there must exist a point along the line $\{ty_1 + (1 - t)y_2 \mid t \in [0, 1]\}$ for which $k(x, y) - \tilde{k}(x, y) = 0$. But $B(x, R(x, \epsilon))$ is a convex set, and thus every point along this line must live in this set as well. Thus, $k(x, y) - \tilde{k}(x, y) = 0$ would contradict the result we have proven that any $y \in B(x, R(x, \epsilon))$ must satisfy $|k(x, y) - \tilde{k}(x, y)| > \epsilon$. Thus, it cannot hold that there exist such points y_1,y_2.

In the case where $B(x, R(x, \epsilon)) \neq \emptyset$, it must hold that $y = x$ satisfies $y \in B(x, R(x, \epsilon))$. Thus, $\|x - y\|_2 = 0 < R(x, \epsilon) = \frac{k(x,x) - \tilde{k}(x,x) - \epsilon}{C(\lambda_r^{-1} m + 1)} \Rightarrow k(x, x) - \tilde{k}(x, x) > \epsilon$. Thus, it must hold that all $y \in B(x, R(x, \epsilon))$ also satisfy $k(x, x) - \tilde{k}(x, x) > \epsilon$. This finalizes the proof of Theorem 1.

□

Lemma 1: $|\tilde{k}(x, x) - \tilde{k}(x, y)| \leq \lambda_r^{-1} mC\|x - y\|$.

Proof: Let $h = k_y - k_x = [k(y, \hat{x}_1) - k(x, \hat{x}_1), \ldots, k(y, \hat{x}_m) - k(x, \hat{x}_m)]^T$, and thus $k_y = k_x + h$.

It follows that:

$$\begin{aligned}
|\tilde{k}(x,x) - \tilde{k}(x,y)| &= |z_r(x)^T z_r(x) - z_r(x)^T z_r(y)| \\
&= |\|z_r(x)\|^2 - k_x^T U_r \Sigma_r^{-1/2} \Sigma_r^{-1/2} U_r^T k_y| \\
&= |\|z_r(x)\|^2 - k_x^T U_r \Sigma_r^{-1} U_r^T (k_x + h)| \\
&= |\|z_r(x)\|^2 - \|z_r(x)\|^2 - k_x^T U_r \Sigma_r^{-1} U_r^T h)| \\
&= |k_x^T U_r \Sigma_r^{-1} U_r^T h| \\
&= |\left\langle U_r \Sigma_r^{-1} U_r^T k_x, h \right\rangle| \\
&\leq \|U_r \Sigma_r^{-1} U_r^T k_x\| \cdot \|h\| \\
&\leq \|U_r \Sigma_r^{-1} U_r^T\| \cdot \|k_x\| \cdot \|h\| \\
&\leq \lambda_r^{-1} \sqrt{m} \cdot \|h\| \\
&= \lambda_r^{-1} \sqrt{m} \sqrt{\sum_{i=1}^{m} \Big(k(y, \hat{x}_i) - k(x, \hat{x}_i) \Big)^2} \\
&\leq \lambda_r^{-1} \sqrt{m} \sqrt{\sum_{i=1}^{m} \big(C \|x - y\| \big)^2} \\
&= \lambda_r^{-1} \sqrt{m} \sqrt{m C^2 \|x - y\|^2} \\
&= \lambda_r^{-1} m C \|x - y\|
\end{aligned}$$

This proves the lemma. □

D.6.2 Theorem 5

We first give a short proof sketch: We prove this by noting that the expected difference between $k(x,x)$ and $\|z(x)\|^2$ is equal to $\|\varphi_{A^\perp}\|^2$, which is the squared distance of $\varphi(x)$ from the subspace A spanned by $\varphi(\hat{x}_1), \ldots, \varphi(\hat{x}_m)$. We can lower bound the expectation of this squared distance by the expected squared distance of the data from the affine subspace in $\mathcal{H}$ with the lowest expected squared distance. The affine subspace minimizing this expected squared distance is precisely $\mu + span(e_1^c, \ldots, e_m^c)$, and the expected squared distance from this affine subspace is equal to $\sum_{j=m+1}^{N^c} \lambda_j^c$.

We now give the full proof: Consider the i, j element of the uncentered, and potentially infinite dimensional covariance matrix of $\varphi'(x) = (\sqrt{\lambda_i} e_i(x))_{i=1}^N$:

$$\begin{aligned}
\Big[\mathbb{E}_X \left[\varphi'(X)\varphi'(X)^T\right]\Big]_{i,j} &= \mathbb{E}_X \left[\sqrt{\lambda_i} e_i(X) \sqrt{\lambda_j} e_j(X)\right] \\
&= \int_{x\in\mathcal{X}} \sqrt{\lambda_i \lambda_j} e_i(x) e_j(x) p(x) dx \\
&= \sqrt{\lambda_i \lambda_j} \int_{x\in\mathcal{X}} e_i(x) e_j(x) p(x) dx \\
&= \lambda_i \cdot \mathbb{1}[i = j].
\end{aligned}$$

Let $\{v_j\}_{j=1}^N$ be an orthonormal basis of $\ell_2([N^c])$ (e.g., $v_1 = (1, 0, 0, \ldots), v_2 = (0, 1, 0, \ldots)$, etc.). Let $U_m^* = span(v_1, \ldots, v_m)$. Let $z(x)$ be an m-dimensional Nyström representation, with landmark points denoted $\{\hat{x}_1, \ldots, \hat{x}_m\}$. Then it follows that:

$$\begin{aligned}
\mathbb{E}_X \left[k(X,X) - \|z(X)\|^2\right] &\geq \min_{\hat{x}_1,\ldots,\hat{x}_m \in \mathcal{X}} \mathbb{E}_X \left[k(x,x) - \|z(x)\|^2\right] \\
&= \min_{\hat{x}_1,\ldots,\hat{x}_m \in \mathcal{X}} \mathbb{E}_X \left[dist^2(\varphi'(x), span(k(\hat{x}_1,\cdot),\ldots,k(\hat{x}_m,\cdot)))\right] \\
&\geq \min_{U \subseteq \ell_2([N]), dim(U)=m} \mathbb{E}_X \left[dist^2(\varphi'(x), U)\right] \\
&\geq \min_{U \subseteq \ell_2([N]), dim(U)=m, u_0 \in \ell_2([N])} \mathbb{E}_X \left[dist^2(\varphi'(x), u_0 + U)\right] && \text{(D.1)} \\
&= \min_{U \subseteq \ell_2([N]), dim(U)=m} \mathbb{E}_X \left[dist^2(\varphi'(x), \mu + U)\right] && \text{(D.2)} \\
&= \min_{U \subseteq \ell_2([N]), dim(U)=m,} \mathbb{E}_X \left[dist^2(\varphi'(x) - \mu, U)\right] \\
&= \min_{U \subseteq \ell_2([N^c]), dim(U)=m,} \mathbb{E}_X \left[dist^2(\varphi^c(x), U)\right] \\
&= \mathbb{E}_X \left[dist^2(\varphi^c(x), U_m^*)\right] && \text{(D.3)} \\
&= \int_{x\in\mathcal{X}} dist^2(\varphi^c(x), U_m^*) p(x) dx \\
&= \int_{x\in\mathcal{X}} \sum_{j=m+1}^{N^c} \lambda_j^c e_j^c(x)^2 p(x) dx \\
&= \sum_{j=m+1}^{N^c} \lambda_j^c \int_{x\in\mathcal{X}} e_j^c(x)^2 p(x) dx \\
&= \sum_{j=m+1}^{N^c} \lambda_j^c.
\end{aligned}$$

Inequality D.1 holds because the expected distance of the data to the "closest" subspace U of dimension m must be greater than or equal to the expected distance of the data to the "closest" *affine* subspace $U + u_0 = \{u + u_0 \mid u \in U\}$ of dimension m. Inequality D.2 holds because the closest affine subspace must have $u_0 = \mu$ (see Theorem 5.3: `http://www.cs.columbia.edu/~djhsu/coms4772-f16/lectures/notes-pca.pdf`). Equality D.3 holds because the subspace of dimension m minimizing the expected distance to the centered data must be the subspace spanned by the m leading eigenvectors of the centered covariance matrix (this is precisely how we defined U_m^*). □

D.6.3 Theorem 6

Letting $f(X) = k(X,X) - \|z(X)\|^2 \in [0,1]$ and $\mathbb{E}_X[f(X)] = \bar{f} \geq R_m$, and assuming $0 \leq \epsilon \leq R_m$, we can apply Hoeffding's inequality as follows:

$$\begin{aligned} \mathbb{P}_X\Big[f(X) \leq \epsilon\Big] &\leq \mathbb{P}_X\Big[f(X) \leq \epsilon + (\bar{f} - R_m)\Big] \\ &= \mathbb{P}_X\Big[f(X) - \bar{f} \leq -(R_m - \epsilon)\Big] \\ &\leq \exp\Big(-2\big(R_m - \epsilon\big)^2\Big) \quad \text{(By Hoeffding's inequality)}. \\ \Rightarrow \quad \mathbb{P}_X\Big[k(X,X) - \|z(X)\|^2 \geq \epsilon\Big] &= \mathbb{P}_X\Big[f(X) \geq \epsilon\Big] \\ &= 1 - \mathbb{P}_X\Big[f(X) \leq \epsilon\Big] \\ &\geq 1 - \exp\Big(-2\big(R_m - \epsilon\big)^2\Big). \end{aligned}$$

□

D.6.4 Theorem 7

We first give a short sketch of the proof: Let $\varphi'_A(x)$ denote the component of $\varphi(x)$ in A which is orthogonal to μ_A, and let $\varphi'_{A^\perp}(x)$ denote the component of $\varphi(x)$ in $A^\perp$ which is orthogonal to $\mu_{A^\perp}$. We can always decompose $\varphi(x)$ into 4 orthogonal components: $\varphi(x) = \alpha_x\mu_A + \beta_x\mu_{A^\perp} + \varphi'_A(x) + \varphi'_{A^\perp}(x)$. By definition of μ, $\mathbb{E}_X[\varphi(X)] = \mu$, and we use this to show $\mathbb{E}_X[\alpha_X] = \mathbb{E}_X[\beta_X] = 1$, and that $\mathbb{E}_X[\varphi'_A(X)] = \mathbb{E}_X\big[\varphi'_{A^\perp}(X)\big] = 0$. It follows that $\mathbb{E}_X[z(X)] = \mu_A$, and thus that $\mathbb{E}_{X,Y}[k(X,Y) - \langle z(X), z(Y)\rangle] = \mathbb{E}_{X,Y}[\langle\varphi(X),\varphi(Y)\rangle - \langle z(X), z(Y)\rangle] = \|\mu\|^2 - \|\mu_A\|^2 = \|\mu_{A^\perp}\|^2$.

We now begin the full proof: Let $\varphi'_A(x)$ denote the component of $\varphi(x)$ in A which is orthogonal to μ_A, and let $\varphi'_{A^\perp}(x)$ denote the component of $\varphi(x)$ in $A^\perp$ which is orthogonal to $\mu_{A^\perp}$.

We can always decompose $\varphi(x)$ into 4 orthogonal components:

$$\varphi(x) = \alpha_x \mu_A + \beta_x \mu_{A^\perp} + \varphi'_A(x) + \varphi'_{A^\perp}(x).$$

We have defined $\mu = \mathbb{E}_X\left[\varphi(X)\right]$, so we know that

$$\begin{aligned} \mu &= \mathbb{E}_X\left[\varphi(X)\right] \\ &= \mathbb{E}_X\left[\alpha_X \mu_A + \beta_X \mu_{A^\perp} + \varphi'_A(X) + \varphi'_{A^\perp}(X)\right] \\ &= \mathbb{E}_X\left[\alpha_X\right]\mu_A + \mathbb{E}_X\left[\beta_X\right]\mu_{A^\perp} + \mathbb{E}_X\left[\varphi'_A(X)\right] + \mathbb{E}_X\left[\varphi'_{A^\perp}(X)\right]. \end{aligned}$$

Because by assumption μ_A, $\mu_{A^\perp}$, $\varphi'_A(x)$, and $\varphi'_{A^\perp}(x)$ are all mutually orthogonal (and thus linearly independent) for any x, the only way for this equality to hold is if $\mathbb{E}_X\left[\alpha_X\right] = \mathbb{E}_X\left[\beta_X\right] = 1$, and $\mathbb{E}_X\left[\varphi'_A(X)\right] = \mathbb{E}_X\left[\varphi'_{A^\perp}(X)\right] = 0$.

Now, consider $z(x) = \alpha_x \mu_A + \varphi'_A(x)$. It is clear that

$$\begin{aligned} \mathbb{E}_X\left[z(X)\right] &= \mathbb{E}_X\left[\alpha_X \mu_A + \varphi'_A(X)\right] \\ &= \mathbb{E}_X\left[\alpha_X\right]\mu_A + \mathbb{E}_X\left[\varphi'_A(X)\right] \\ &= \mu_A \end{aligned}$$

It follows from Lemma 7.1 below that

$$\begin{aligned} \mathbb{E}_{X,Y}\left[\langle z(X), z(Y)\rangle\right] &= \langle \mathbb{E}_X\left[z(X)\right], \mathbb{E}_Y\left[z(Y)\right]\rangle \\ &= \|\mu_A\|^2 \end{aligned}$$

Also by Lemma 7.1, we know that

$$\begin{aligned} \mathbb{E}_{X,Y}\left[k(X,Y)\right] &= \mathbb{E}_{X,Y}\left[\langle \varphi(X), \varphi(Y)\rangle\right] \\ &= \langle \mathbb{E}_X\left[\varphi(X)\right], \mathbb{E}_Y\left[\varphi(Y)\right]\rangle \\ &= \|\mu\|^2 \end{aligned}$$

Combine the above 2 equalities, we get that

$$\begin{aligned} \mathbb{E}_{X,Y}\left[k(X,Y) - \langle z(X), z(Y)\rangle\right] &= \|\mu\|^2 - \|\mu_A\|^2 \\ &= \|\mu_{A^\perp}\|^2 \end{aligned}$$

This completes the proof. □

Lemma 7.1. *Let W, Z be random variables over $\mathcal{H} = \ell_2([N])$, where $N \leq \infty$ is the dimension of $\mathcal{H}$. Then*

$$\mathbb{E}_{W,Z}\left[\langle W, Z\rangle\right] = \langle \mathbb{E}_W\left[W\right], \mathbb{E}_Z\left[Z\right]\rangle$$

Proof.

$$\begin{aligned}
\mathbb{E}_{W,Z}\left[\langle W, Z\rangle\right] &= \mathbb{E}_{W,Z}\left[\sum_{i=1}^{N} W_i Z_i\right] \\
&= \sum_{i=1}^{N} \mathbb{E}_W\left[W_i\right] \mathbb{E}_Z\left[Z_i\right] \\
&= \langle \mathbb{E}_W\left[W\right], \mathbb{E}_Z\left[Z\right]\rangle
\end{aligned}$$

□

D.7 Other ways of understanding the Nyström method

D.7.1 Nyström method as a projection onto a subspace

In Appendix D.5, we proved that the Nyström method can be understood as performing a projection of the datapoints in the RKHS onto the subspace corresponding to the landmark points. For more details, see that section of the Appendix.

D.7.2 Nyström method as a solution to an optimization problem

Another way of understanding the linear transformation performed by Nyström on top of $k'(x)$ is by viewing it as a solution to the following optimization problem:

$$\begin{aligned}
&\min_{L \in \mathbb{R}^{m \times m}} \quad \sum_{i,j=1}^{m} \Big(\langle L\, k'(\hat{x}_i), L\, k'(\hat{x}_j)\rangle - k(\hat{x}_i, \hat{x}_j)\Big)^2 \\
&\min_{L \in \mathbb{R}^{m \times m}} \quad \sum_{i,j=1}^{m} \Big(k'(\hat{x}_i)^T\, L^T L\, k'(\hat{x}_j) - k(\hat{x}_i, \hat{x}_j)\Big)^2 \\
&\min_{L \in \mathbb{R}^{m \times m}} \quad \|\hat{K}^T\, L^T L\, \hat{K} - \hat{K}\|_F^2
\end{aligned}$$

It is immediately clear from this objective function that the global minimizer satisfies

$$\begin{aligned} \hat{K}^T L^T L &= \mathbb{1} \\ \Rightarrow L^T L &= \hat{K}^{-1} \\ &= U\Lambda^{-1}U^T \\ \Rightarrow \quad L &= \Lambda^{-1/2}U^T, \end{aligned}$$

which is exactly the linear transformation performed by Nyström ($z(x) = \Lambda^{-1/2}U^T k'(x)$).

D.7.3 Nyström method as a preconditioner

Consider the kernel matrix $\hat{K} = U\Lambda U^T$ over the landmark points $\{\hat{x}_1, \ldots, \hat{x}_m\}$. Let's denote $k'(x) = k_x = [k(\hat{x}_1, x), \ldots, k(\hat{x}_m, x)]^T$. Thus, $\hat{K} = [k'(\hat{x}_1), \ldots, k'(\hat{x}_m)]$. We can consider $k'(x)$ as a feature representation for a point x, and from this perspective, $\hat{K}$ is a data matrix storing these representations as columns for all the landmark points. The (non-centered) sample covariance matrix of this data matrix is thus (proportional to) $H_k = \hat{K}\hat{K}^T = (U\Lambda U^T)(U\Lambda U^T) = U\Lambda^2 U^T$. Thus, if $\hat{K}$ is poorly conditioned, H_k will be VERY poorly conditioned! Specifically, the condition number of H_k is equal to the square of the condition number of $\hat{K}$: $\kappa(H_k) = \frac{\lambda_1^2}{\lambda_m^2} = \left(\frac{\lambda_1}{\lambda_m}\right)^2 = \kappa(\hat{K})^2$.

The Nyström method (partially) addresses this problem of learning on top of the $k'(x)$ feature representations by preconditioning these representations: $z(x) = \Lambda^{-1/2}U^T k'(x)$. Letting $Z = [z(\hat{x}_1), \ldots, z(\hat{x}_m)] = \Lambda^{-1/2}U^T\hat{K}$, we can consider the corresponding covariance matrix:

$$\begin{aligned} H_z &= ZZ^T \\ &= (\Lambda^{-1/2}U^T\hat{K})(\hat{K}U\Lambda^{-1/2}) \\ &= (\Lambda^{-1/2}U^T U\Lambda U^T)(U\Lambda U^T U\Lambda^{-1/2}) \\ &= \Lambda^{-1/2} \cdot \Lambda \cdot \Lambda \cdot \Lambda^{-1/2} \\ &= \Lambda \end{aligned}$$

Thus, the condition number for the Nyström covariance matrix H_z is $\kappa(H_z) = \frac{\lambda_1}{\lambda_m} = \kappa(\hat{K})$, which is a lot better than $\kappa(\hat{K})^2$.

D.7.4 Nyström method as eigenfunction approximator

This explanation is adapted from the original paper introducing the Nyström method [Williams and Seeger, 2001].

Consider the following linear operator $L : L_2(\mathcal{X}; p) \to L_2(\mathcal{X}; p)$:

$$L[f](y) = \int_{\mathcal{X}} k(y, x) f(x) p(x) dx$$

Recall that the inner product in $L_2(\mathcal{X}; p)$ is defined as:

$$\langle f, g \rangle_{L_2} = \int_{x \in \mathcal{X}} f(x) g(x) p(x) dx.$$

By Mercer's Theorem, it follows that $k(x, y)$ can be decomposed as follows:

$$k(x, y) = \sum_{j \in J} \lambda_j \phi_j(x) \phi_j(y),$$

where the λ_i and ϕ_i are the eigenvalues and eigenvectors of L, and the eigenvalues λ_i are strictly positive scalars, satisfying $\lambda_1 \geq \lambda_2 \geq \ldots > 0$. Note that this shows that there are at most countably infinitely many of them). Mercer's Theorem gives a (potentially) infinite dimensional representation $\varphi(x) = \{\sqrt{\lambda_j}\phi_j(x)\}_{j \in J} \in \ell_2(J)$ such that $k(x, y) = \langle \varphi(x), \varphi(y) \rangle_{\ell_2(J)} = \sum_{j \in J} \lambda_j \phi_j(x) \phi_j(y)$.

By the definition of eigenvalues/eigenfunctions in $L_2(\mathcal{X}; p)$, we know that

$$\begin{aligned} L[\phi_i] &= \lambda_i \phi_i \\ \Rightarrow \quad \int_{\mathcal{X}} k(y, x) \phi_i(x) p(x) dx &= \lambda_i \phi_i(y). \end{aligned}$$

We now consider a Monte Carlo approximation to the above integral, with i.i.d. sample $\{x_1, \ldots, x_m\}$ from $p(x)$:

$$\frac{1}{m} \sum_{k=1}^{m} k(y, x_k) \phi_i(x_k) \approx \lambda_i \phi_i(y) \tag{D.4}$$

These eigenfunctions are by definition "p-orthogonal", meaning:

$$\int_{\mathcal{X}} \phi_i(x) \phi_j(x) p(x) dx = \delta_{ij},$$

where $\delta_{ij} = \mathbb{1}[i = j]$. Once again using monte-carlo approximation of this integral, this becomes:

$$\Rightarrow \quad \frac{1}{m} \sum_{k=1}^{m} \phi_i(x_k) \phi_j(x_k) \approx \delta_{ij}$$

Equation $D.4$ can be rewritten in matrix notation by considering $y \in \{x_1, \ldots, x_m\}$.

$$\begin{aligned} \frac{1}{m}\sum_{k=1}^{m} k(x_j, x_k)\phi_i(x_k) &\approx \lambda_i\phi_i(x_j) \\ \Rightarrow \quad K^{(m)} \cdot \Phi^{(m)} \cdot \left(\frac{1}{m}\mathbb{1}\right) &\approx \Phi^{(m)} \cdot \Lambda, \end{aligned}$$

where $K^{(m)}$ is the m by m matrix whose j^{th} row is $k_{x_j}^{(m)T} = [k(x_j, x_1), \ldots, k(x_j, x_m)]$, $\Phi^{(m)}$ is the m by m matrix whose i^{th} column is $\bar{\phi}_i^{(m)} = [\phi_i(x_1), \ldots, \phi_i(x_m)]^T$, and Λ is the diagonal matrix with λ_i as the i^{th} term on the diagonal. Note that the ℓ_2 norm of $\bar{\phi}_i^{(m)}$ is approximately $\sqrt{m}$ by the "p-orthogonality" of the eigenvectors, and thus if we take $\Phi^{(m)} \cdot \left(\frac{1}{\sqrt{m}}\mathbb{1}\right)$, the columns of this matrix are approximately orthogonal. Thus, by multiplying both sides of the above equation by $\sqrt{m}\mathbb{1}$, we can rearrange it as follows (using the fact that the matrices of the form $c\mathbb{1}$ for any constant c commute with all matrices):

$$\begin{aligned} K^{(m)} \cdot \Phi^{(m)} \cdot \left(\frac{1}{\sqrt{m}}\mathbb{1}\right) &\approx \left(\sqrt{m}\mathbb{1}\right) \cdot \Phi^{(m)} \cdot \Lambda \\ K^{(m)} \cdot \Phi^{(m)} \cdot \left(\frac{1}{\sqrt{m}}\mathbb{1}\right) &\approx \Phi^{(m)} \cdot \left(\frac{1}{\sqrt{m}}\mathbb{1}\right) \cdot \Lambda \cdot \left(m\mathbb{1}\right) \\ K^{(m)} \cdot \bar{U}^{(m)} &\approx \bar{U}^{(m)} \cdot \bar{\Lambda}^{(m)}, \end{aligned}$$

where $\bar{U}^{(m)} = \Phi^{(m)} \cdot \left(\frac{1}{\sqrt{m}}\mathbb{1}\right)$ is an (approximately) orthogonal matrix whose i^{th} column is $\frac{1}{\sqrt{m}}\bar{\phi}_i^{(m)}$, and where $\bar{\Lambda}^{(m)} = \Lambda \cdot \left(m\mathbb{1}\right)$ is a diagonal matrix whose i^{th} term on the diagonal is $m\lambda_i$. This motivates computing the (exact) eigendecomposition of $K^{(m)} = U^{(m)}\Lambda^{(m)}U^{(m)T}$, and treating $U^{(m)}$ as an approximation of $\bar{U}^{(m)}$, and $\Lambda^{(m)}$ as an approximation of $\bar{\Lambda}^{(m)}$. Letting $U_i^{(m)}$ denote the i^{th} column of $U^{(m)}$, $U_{k,i}^{(m)}$ denote the k^{th} term in this column, $\lambda_i^{(m)} = \Lambda_{i,i}^{(m)}$, and $\bar{\lambda}_i^{(m)} = \bar{\Lambda}_{i,i}^{(m)}$, we get that:

$$\begin{aligned} U_i^{(m)} &\approx \bar{U}_i^{(m)} \\ &= \frac{1}{\sqrt{m}}\bar{\phi}_i^{(m)} \\ \Rightarrow \quad \bar{\phi}_i^{(m)} &\approx \sqrt{m} \cdot U_i^{(m)} \\ \Rightarrow \quad \phi_i(x_k) &\approx \sqrt{m} \cdot U_{k,i}^{(m)} \\ \lambda_i^{(m)} &\approx \bar{\lambda}_i^{(m)} \\ &= m\lambda_i \\ \Rightarrow \quad \lambda_i &\approx \frac{1}{m}\lambda_i^{(m)} \end{aligned}$$

Plugging these identities back into Equation D.4 gives us an approximation for the i^{th} eigenfunction of L:

$$\begin{aligned}
\frac{1}{m}\sum_{k=1}^{m} k(y,x_k)\phi_i(x_k) &\approx \lambda_i\phi_i(y) \\
\frac{1}{m}\sum_{k=1}^{m} k(y,x_k)\sqrt{m}\cdot U_{k,i}^{(m)} &\approx \frac{1}{m}\lambda_i^{(m)}\phi_i(y) \\
\frac{\sqrt{m}}{\lambda_i^{(m)}}\sum_{k=1}^{m} k(y,x_k)\cdot U_{k,i}^{(m)} &\approx \phi_i(y) \\
\Rightarrow \quad \phi_i(y) &\approx \frac{\sqrt{m}}{\lambda_i^{(m)}}U_i^{(m)T}k_y^{(m)},
\end{aligned}$$

where $k_y^{(m)} = [k(y,x_1),\ldots,k(y,x_m)]^T$. Furthermore, we already saw above the (approximate) mapping between the eigenvalues $\lambda_i^{(m)}$ of $K^{(m)}$ and the eigenvalues λ_i of L.

$$\lambda_i \approx \frac{1}{m}\lambda_i^{(m)}$$

Thus, we can see that using the monte-carlo estimate in equation D.4 reveals an (approximate) correspondence between eigenvalues/vectors of $K^{(m)}$ and the eigenvalues and eigenfunctions of the linear operator L corresponding to the kernel k. This correspondence allows us to approximate the *full* kernel matrix $K^{(n)}$, using the SVD of $K^{(m)}$, in the following manner:

$$\begin{aligned}
\lambda_i &\approx \frac{1}{m}\lambda_i^{(m)} \\
&\approx \frac{1}{n}\lambda_i^{(n)} \\
\Rightarrow \quad \lambda_i^{(n)} &\approx \frac{n}{m}\lambda_i^{(m)}
\end{aligned}$$

Thus, we can use the eigenvalues of $K^{(m)}$ to approximate the eigenvalues of $K^{(n)}$. Computing the approximate eigenvectors of $K^{(n)}$ is a tiny bit trickier. Using $\phi_i(y) \approx \frac{\sqrt{m}}{\lambda_i^{(m)}}U_i^{(m)T}k_y^{(m)}$, we can plug in all $y \in \{x_1,\ldots,x_n\}$, and get:

$$\phi_i(x_k) \approx \frac{\sqrt{m}}{\lambda_i^{(m)}}U_i^{(m)T}k_{x_k}^{(m)}$$

Letting $\bar{\phi}_i^{(n)} = [\phi_i(x_1), \ldots, \phi_i(x_n)]^T$, and letting $K_{m,n}$ be the m by n matrix whose k^{th} columns is $k_{x_k}^{(m)}$, we can see that

$$\begin{aligned} \bar{\phi}_i^{(n)} &\approx \left(\frac{\sqrt{m}}{\lambda_i^{(m)}} U_i^{(m)T} K_{m,n}\right)^T \\ \Rightarrow \quad \bar{\phi}_i^{(n)} &\approx \frac{\sqrt{m}}{\lambda_i^{(m)}} K_{m,n}^T U_i^{(m)}. \end{aligned}$$

We also know by applying our results from $K^{(m)}$ directly to $K^{(n)}$ that

$$\bar{\phi}_i^{(n)} \approx \sqrt{n} \cdot U_i^{(n)}.$$

Thus, combining the above 2 equations, we can see that

$$U_i^{(n)} \approx \sqrt{\frac{m}{n}} \frac{1}{\lambda_i^{(m)}} K_{m,n}^T U_i^{(m)}.$$

This last equation gives us an approximation to the top m eigenvectors of $K^{(n)}$ using the m eigenvectors of $K^{(m)}$.

To summarize, we have shown that we can approximate the top m eigenvalues and eigenvectors of $K^{(n)}$ using the eigenvalues and eigenvectors of $K^{(m)}$, as follows:

$$\begin{aligned} \lambda_i^{(n)} &\approx \frac{n}{m} \lambda_i^{(m)} \\ \Rightarrow \quad \tilde{\Lambda}^{(n)} &\triangleq \frac{n}{m} \Lambda^{(m)} \\ U_i^{(n)} &\approx \sqrt{\frac{m}{n}} \frac{1}{\lambda_i^{(m)}} K_{m,n}^T U_i^{(m)} \\ \Rightarrow \quad \tilde{U}^{(n)} &\triangleq \sqrt{\frac{m}{n}} K_{m,n}^T U^{(m)} (\Lambda^{(m)})^{-1} \end{aligned}$$

where we let $\tilde{U}^{(n)}$ be the n by m matrix containing the above approximations for $U_1^{(n)}$ through $U_m^{(n)}$ as columns, and $\tilde{\Lambda}^{(n)}$ be the m by m matrix containing the above approximation for $\lambda_1^{(n)}$ through $\lambda_m^{(n)}$ on the diagonal. We can now use $\tilde{U}^{(n)}$ and $\tilde{\Lambda}^{(n)}$ to construct the following low-rank approximation for $K^{(n)}$:

$$\begin{aligned}
K^{(n)} &\approx \tilde{U}^{(n)}\tilde{\Lambda}^{(n)}\tilde{U}^{(n)T} \\
&= \left(\sqrt{\frac{m}{n}}K_{m,n}^{T}U^{(m)}(\Lambda^{(m)})^{-1}\right)\left(\frac{n}{m}\Lambda^{(m)}\right)\left(\sqrt{\frac{m}{n}}K_{m,n}^{T}U^{(m)}(\Lambda^{(m)})^{-1}\right)^{T} \\
&= K_{m,n}^{T}U^{(m)}(\Lambda^{(m)})^{-1}\Lambda^{(m)}(\Lambda^{(m)})^{-1}U^{(m)T}K_{m,n} \\
&= K_{m,n}^{T}U^{(m)}(\Lambda^{(m)})^{-1}U^{(m)T}K_{m,n} \\
&= K_{m,n}^{T}(K^{(m)})^{-1}K_{m,n}.
\end{aligned}$$

This is precisely the approximate low-rank decomposition the Nyström method provides for the n by n kernel matrix $K^{(n)}$.