HAL
open science

Kernel Approximation Methods for Speech Recognition

Avner May, Alireza Bagheri Garakani, Zhiyun Lu, Dong Guo, Kuan Liu, Aurélien Bellet, Linxi Fan, Michael Collins, Daniel Hsu, Brian Kingsbury, et al.

To cite this version:

Avner May, Alireza Bagheri Garakani, Zhiyun Lu, Dong Guo, Kuan Liu, et al.. Kernel Approximation Methods for Speech Recognition. Journal of Machine Learning Research, 2019, 20, pp.1 - 36. hal-02166422

HAL Id: hal-02166422

https://inria.hal.science/hal-02166422

Submitted on 26 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Journal of Machine Learning Research 20 (2019) 1-36 Submitted 1/17; Revised 9/17; Published 4/19

Kernel Approximation Methods for Speech Recognition

Avner May[1†] AVNERMAY@CS.STANFORD.EDU
Alireza Bagheri Garakani[2‡] BAGHERIG@USC.EDU
Zhiyun Lu[2‡] ZHIYUNLU@USC.EDU
Dong Guo[2‡] DONGGUO@USC.EDU
Kuan Liu[2,5‡] LIUKUAN@GOOGLE.COM
Aurélien Bellet[3] AURELIEN.BELLET@INRIA.FR
Linxi Fan[1] JIMFAN@CS.STANFORD.EDU
Michael Collins[4,5] MCOLLINS@CS.COLUMBIA.EDU
Daniel Hsu[4] DJHSU@CS.COLUMBIA.EDU
Brian Kingsbury[6] BEDK@US.IBM.COM
Michael Picheny[6] PICHENY@US.IBM.COM
Fei Sha[2] FEISHA@USC.EDU

[1] *Department of Computer Science, Stanford University, Stanford, CA 94305, USA*
[2] *Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA*
[3] *INRIA, 40 Avenue Halley, 59650 Villeneuve d'Ascq, France*
[4] *Department of Computer Science, Columbia University, New York, NY 10027, USA*
[5] *Google Inc, USA*
[6] *IBM Research AI, Yorktown Heights, NY 10598, USA*
[† ‡]: Contributed equally as the first and second co-authors, respectively

Editor: Benjamin Recht

Abstract

We study the performance of kernel methods on the acoustic modeling task for automatic speech recognition, and compare their performance to deep neural networks (DNNs). To scale the kernel methods to large data sets, we use the random Fourier feature method of Rahimi and Recht (2007). We propose two novel techniques for improving the performance of kernel acoustic models. First, we propose a simple but effective feature selection method which reduces the number of random features required to attain a fixed level of performance. Second, we present a number of metrics which correlate strongly with speech recognition performance when computed on the heldout set; we attain improved performance by using these metrics to decide when to stop training. Additionally, we show that the linear bottleneck method of Sainath et al. (2013a) improves the performance of our kernel models significantly, in addition to speeding up training and making the models more compact. Leveraging these three methods, the kernel methods attain token error rates between 0.5% better and 0.1% worse than fully-connected DNNs across four speech recognition data sets, including the TIMIT and Broadcast News benchmark tasks.

Keywords: kernel methods, deep neural networks, acoustic modeling, automatic speech recognition, feature selection, logistic regression

©2019 Avner May, Alireza Bagheri Garakani, Zhiyun Lu, Dong Guo, Kuan Liu, Aurélien Bellet, Linxi Fan, Michael Collins, Daniel Hsu, Brian Kingsbury, Michael Picheny, Fei Sha.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v20/17-026.html.

1. Introduction

Large-scale non-linear classification is an important and challenging problem in machine learning. In recent years, deep learning techniques have significantly advanced state-of-the-art performance on classification problems in various domains, including automatic speech recognition (ASR) (Seide et al., 2011a; Hinton et al., 2012; Mohamed et al., 2012; Xiong et al., 2017; Saon et al., 2017), computer vision (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; He et al., 2016), and natural language processing (NLP) (Mikolov et al., 2013; Sutskever et al., 2014; Andor et al., 2016). Deep neural networks (DNNs) are able to gracefully scale to large data sets, and can successfully leverage this additional data to achieve strong empirical performance. In contrast, kernel methods, which are attractive due to their high capacity, as well as for their theoretical learning guarantees and tractability (Schölkopf and Smola, 2002), do not scale well. In particular, with data sets of size N, the $\Theta(N^2)$ size of the kernel matrix makes training prohibitively slow, while the typical $\Theta(N)$ size of the resulting models (Steinwart, 2003) makes their deployment impractical.

An important technique for scaling kernel methods to large data sets is to use approximation. Kernel approximation methods construct explicit feature maps whose dot-products approximate the kernel function, and then learn linear models with these features. Notable approximation methods include the Nyström method (Williams and Seeger, 2000), and random Fourier features (RFFs) (Rahimi and Recht, 2007). Although these methods make it possible to apply kernel methods to large-scale tasks, there have been very few published attempts applying these methods to the challenging large-scale tasks on which deep learning techniques have truly shined.[1]

The primary contribution of this paper is to demonstrate, in a large-scale setting where deep learning techniques have been known to dominate, that kernel approximation methods can effectively compete with fully-connected DNNs. More specifically:

- We benchmark the performance of kernel approximation methods (RFFs) relative to fully-connected DNNs on the acoustic modeling problem for automatic speech recognition, on four data sets with millions of training points and hundreds/thousands of classes.[2]
- We propose three methods for improving the performance of the kernel acoustic models: a feature selection method, new early stopping criteria for training, and the use of a linear bottleneck layer (Sainath et al., 2013a). We show that using these techniques, the kernel methods attain token error rates (TER)[3] between 0.5% better and 0.1% worse than fully-connected DNNs on the four data sets.

This contribution is important for both practical and theoretical reasons. From a practical perspective, it suggests that kernel methods can be competitive with deep learning

1. See related work section for discussion.
2. 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).
3. For our Cantonese data set, 'token error rate' corresponds to 'character error rate.' For our Bengali and Broadcast News data sets, it corresponds to 'word error rate.' For TIMIT, it corresponds to 'phone error rate.'

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 et al., 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 kernel methods can match DNNs on a task this large and challenging gives an important new perspective.

As discussed above, we propose three methods to improve the performance of the kernel acoustic models. First, we propose a simple feature selection algorithm, which effectively reduces the number of random features required to attain a fixed level of performance. We iteratively select features from large pools of random features, using learned weights in the selection criterion. This has two clear benefits: (1) the subsequent training on the selected features is considerably faster than training on the entire pool of random features, and (2) the resulting model is also much smaller. For certain kernels, this feature selection approach—which is applied at the level of the random features—can be regarded as a non-linear method for feature selection at the level of the input features. We use this observation to motivate the design of a new kernel function, the "sparse Gaussian kernel," which performs well in conjunction with the feature selection algorithm.

Second, we present several frame-level metrics which correlate strongly with the TER. We show that we can attain notable gains in TER for both kernels and DNNs by monitoring these metrics on the heldout set during training to determine when to stop training.

Lastly, we demonstrate the importance of using a linear bottleneck (Sainath et al., 2013a) in the parameter matrix of our kernel models. Not only does this method improve the performance of our kernel models significantly, it also makes training faster, and reduces the size of the models learned.

In this paper we unify and extend the previous works of Lu et al. (2016) and May et al. (2016). The most significant additions are as follows: (1) we show that we can attain improved performance by combining the methods from both papers; (2) we present a more extensive set of experiments, including results on the TIMIT benchmark task, and a detailed ablation study to reveal the marginal improvements from each method; (3) we present a larger set of metrics which correlate strongly with TER, and show that we can attain improved performance by using these metrics during training to decide when to decay the learning rate and stop training.

The rest of the paper is organized as follows. We review related work in Section 2. We provide some background for kernel approximation methods, as well as for acoustic modeling, in Section 3. We present our feature selection algorithm in Section 4. In Section 5, we present several novel metrics which correlate strongly with TER, and discuss how they can be used during training to improve TER performance. In Section 6, we report extensive experiments comparing DNNs and kernel methods, including results using the methods discussed above. We conclude in Section 7.

2. Related Work

Scaling up kernel methods has been a long-standing and actively studied problem (Platt, 1998; DeCoste and Schölkopf, 2002; Tsang et al., 2005; Bottou et al., 2007; Clarkson, 2010). Approximating kernels by constructing explicit finite-dimensional feature representations, where dot products between these representations approximate the kernel function, has emerged as a powerful technique (e.g., Williams and Seeger, 2000; 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, 2000). For shift-invariant kernels, the RFF technique of Rahimi and Recht (2007) uses random projections 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 aim to make RFFs more computationally efficient. One line of work attempts to reduce the time and memory needed to compute the RFFs 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 the random features (Dai et al., 2014). For kernels with sparse feature expansions, Sonnenburg and Franc (2010) show how to scale kernel SVMs to data sets 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 DNNs on these tasks. Notable exceptions are the following: on image recognition problems, it has been shown that random Fourier features (Rahimi and Recht, 2007) can be used to replace the fully connected layers in the convolutional neural network (CNN) known as AlexNet (Krizhevsky et al., 2012), and achieve comparable performance on the ImageNet 2012 data set (Dai et al., 2014; Yang et al., 2015). Although these results suggest that kernel methods can replace the fully-connected layers of neural networks, the hybrid CNN-kernel model makes it difficult to disentangle the relative importance of these two components. In ASR, the only existing work applying kernel approximation methods has been quite limited in scope (Huang et al., 2014), using the relatively easy and small-scale TIMIT data set.[4] A detailed evaluation of kernel 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. Here, the ℓ_1-regularization may cause the learned solution

4. Here, we are excluding the results presented in this paper, some of which have already been published (May et al., 2016; Lu et al., 2016).

to only depend on a subset of the generated features. 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 the forward-backward splitting (FOBOS) algorithm of Duchi and Singer (2009) (using ℓ_1/ℓ_2 mixed-norm 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. 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.

Another improvement we propose alters the frame-level training of the acoustic model to improve the speech recognition performance of the final model. A set of methods, typically referred to as *sequence training* techniques, share our goal of tuning the acoustic model for the purpose of improving its recognition performance. 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. Our 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, there has been significant work on end-to-end training of ASR systems, which removes the need for frame-level training of acoustic models altogether (Graves et al., 2006; Amodei et al., 2016; Chan et al., 2016; Chiu et al., 2018). For future work, we would like to see how much kernel models can benefit from the various sequence training methods mentioned above, relative to DNNs.

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, 2016), speaker adaptation through the use of i-vectors (Dehak et al., 2011), training on large data sets (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., 2013c; Soltau et al., 2014; Simonyan and Zisserman, 2015; 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 feed-forward DNNs. The most recent state-of-the-art systems (Xiong et al., 2017; Saon 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 data set when switching from a sigmoid DNN architecture to an LSTM, while in Xiong et al. (2017) they see that a ResNet CNN (He et al., 2016) improves upon a ReLU DNN by 1.6%.

In the context of these recent advances, our results showing competitive performance with fully-connected DNNs are significant, for a number of reasons. First of all, while no longer being state-of-the-art, fully-connected DNNs still attain strong performance on the acoustic modeling task. Second, fully-connected 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 deep learning architectures (Simonyan and Zisserman, 2015; He et al., 2016). Additionally, we believe it should be important 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 simpler interpretation, and cleaner theoretical analysis, 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 3.3, 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 to generalize well, a problem known as the *curse of dimensionality* (Härdle et al., 2004; Bengio et al., 2007). In Ba and Caruana (2014), the authors show that the performance of shallow neural networks can be improved 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 paper adds credence to the argument that shallow models can compete with deep networks.

3. Background

In this section, we provide background on kernels and how to approximate them with random Fourier features (Rahimi and Recht, 2007), on acoustic modeling (using neural networks and kernels), and on the linear bottleneck method of Sainath et al. (2013a).

3.1. Kernel Methods and Random Fourier Features

Kernel methods, broadly speaking, are a set of machine learning techniques which either explicitly or implicitly map data from the input space $\mathcal{X}$ to some feature space $\mathcal{H}$, in which a linear model is learned. A kernel function $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is then defined[5] as the function which takes as input $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$, and returns the dot-product of the corresponding points in $\mathcal{H}$. If we let $\boldsymbol{\phi}\colon \mathcal{X} \to \mathcal{H}$ denote the map into the feature space, then $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle$. Standard kernel methods avoid inference in $\mathcal{H}$, because it is generally a very high-dimensional, or even infinite-dimensional, space. Instead, they solve the dual problem by using the N-by-N kernel matrix containing the values of the kernel function applied to all pairs of N training points. This method of working in the dual space is known as the "kernel trick," and it provides a nice computational advantage when $\dim(\mathcal{H})$ is far greater than N. However, when N is very large, the $\Theta(N^2)$ size of the kernel matrix makes training impractical.

Rahimi and Recht (2007) address this problem by leveraging Bochner's Theorem, a classical result in harmonic analysis, to provide a way to approximate any positive-definite shift-invariant kernel K with finite-dimensional features, known as random Fourier features. A kernel function K is shift-invariant if and only if $K(\boldsymbol{x}, \boldsymbol{x}') = \hat{K}(\boldsymbol{x} - \boldsymbol{x}')\ \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ 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 $K(\boldsymbol{x}, \boldsymbol{x}') = \hat{K}(\boldsymbol{x} - \boldsymbol{x}')$ on $\mathbb{R}^d$ is positive-definite if and only if $\hat{K}$ is the Fourier transform of a non-negative measure $\mu(\boldsymbol{\omega})$.*

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

$$\hat{K}(\boldsymbol{\delta}) = \int_{\mathbb{R}^d} \mu(\boldsymbol{\omega}) e^{-j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\delta}}\, d\boldsymbol{\omega}, \tag{1}$$

where

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

is the inverse Fourier transform[6] of $\hat{K}(\boldsymbol{\delta})$, and where $j = \sqrt{-1}$. By Bochner's theorem, $\mu(\boldsymbol{\omega})$ is a non-negative measure. As a result, if we let $Z = \int_{\mathbb{R}^d} \mu(\boldsymbol{\omega}) d\boldsymbol{\omega}$, then $p(\boldsymbol{\omega}) = \frac{1}{Z}\mu(\boldsymbol{\omega})$ is a proper probability distribution, and we get that

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

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

$$\hat{K}(\boldsymbol{\delta}) = \hat{K}(\boldsymbol{x} - \boldsymbol{x}') = \int_{\mathbb{R}^d} p(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}')}\, d\boldsymbol{\omega} = \mathbb{E}_{\boldsymbol{\omega}}\left[e^{j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}} e^{-j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}'} \right]. \tag{3}$$

5. It is also possible to define the kernel function prior to defining the feature map; then, for positive-definite kernel functions, Mercer's theorem guarantees that a corresponding feature map $\boldsymbol{\phi}$ exists such that $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle$.
6. There are various ways of defining the Fourier transform and its inverse. We use the convention specified in Equations (1) and (2), which is consistent with Rahimi and Recht (2007).

Kernel name	$\boldsymbol{K(x,y)}$	$\boldsymbol{p(\omega)}$	**Density name**
Gaussian	$e^{-\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2/2\sigma^2}$	$(2\pi(1/\sigma^2))^{-d/2}e^{-\frac{\|\boldsymbol{\omega}\|_2^2}{2(1/\sigma)^2}}$	Normal$(\mathbf{0}_d, \frac{1}{\sigma^2}\mathbb{1}_d)$
Laplacian	$e^{-\lambda\|\boldsymbol{x}-\boldsymbol{x}'\|_1}$	$\prod_{i=1}^d \frac{1}{\lambda\pi(1+(\boldsymbol{\omega}_i/\lambda)^2)}$	Cauchy$(\mathbf{0}_d, \lambda)$

Table 1: Gaussian and Laplacian Kernels, together with their sampling distributions $p(\boldsymbol{\omega})$.

This can be further simplified as

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

where $\boldsymbol{\omega}$ is drawn from $p(\boldsymbol{\omega})$, and b is drawn uniformly from $[0, 2\pi]$. See Appendix B for details on why this specific functional form is correct.

This motivates a sampling-based approach for approximating the kernel function. Concretely, we draw $\{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D\}$ independently from the distribution $p(\boldsymbol{\omega})$, and $\{b_1, \ldots, b_D\}$ independently from the uniform distribution on $[0, 2\pi]$. We then use these parameters to approximate the kernel as follows:

$$K(\boldsymbol{x},\boldsymbol{x}') \approx \frac{1}{D}\sum_{i=1}^{D}\sqrt{2}\cos(\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{x}+b_i)\cdot\sqrt{2}\cos(\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{x}'+b_i) = z(\boldsymbol{x})^{\mathrm{T}}z(\boldsymbol{x}'),$$

where $z_i(\boldsymbol{x}) = \sqrt{\frac{2}{D}}\cos(\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{x}+b_i)$ is the ith element of the D-dimensional random vector $z(\boldsymbol{x})$. In Table 1, we list two popular (properly-scaled) positive-definite kernels with their respective inverse Fourier transforms.

Using these random feature maps in conjunction with linear learning algorithms can yield huge gains in efficiency relative to standard kernel methods on large data sets. Learning with a representation $z(\cdot) \in \mathbb{R}^D$ is relatively efficient provided that D is far less than the number of training samples N. For example, in our experiments (Section 6), we have 2 million to 16 million training samples, while $D \approx 25{,}000$ often leads to good performance.

3.2. Neural Networks Acoustic Models

Neural network acoustic models provide a conditional probability distribution $p(y|\boldsymbol{x})$ over C possible acoustic states, conditioned on an acoustic frame $\boldsymbol{x}$ encoded in some feature representation. The acoustic states correspond to context-dependent phoneme states (Dahl et al., 2012), and in modern speech recognition systems, the number of such states is on the order of 10^3 to 10^4. The acoustic model is used within probabilistic systems for decoding speech signals into word sequences. Typically, the probability model used is a hidden Markov model (HMM), where the model's emission and transition probabilities are provided by an acoustic model together with a language model. We use Bayes' rule to compute the probability $p(\boldsymbol{x}|y)$ of emitting a certain acoustic feature vector $\boldsymbol{x}$ from state y, given the output $p(y|\boldsymbol{x})$ of the neural network:

$$p(\boldsymbol{x}|y) \quad = \quad \frac{p(y|\boldsymbol{x})p(\boldsymbol{x})}{p(y)}.$$

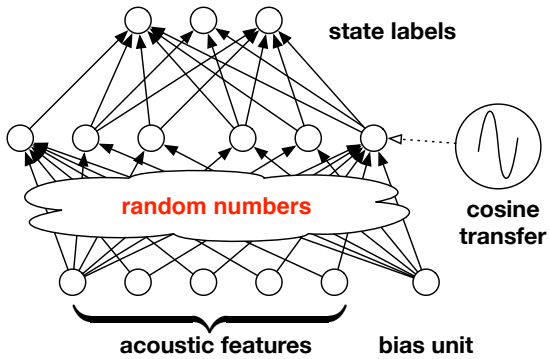

Figure 1: Kernel acoustic model seen as a shallow neural network.

Note that $p(x)$ can be ignored at inference time because it doesn't affect the relative scores assigned to different word sequences, and $p(y)$ is simply the prior probability of HMM state y. The Viterbi algorithm can then be used to determine the most likely word sequence (see Gales and Young (2007) for an overview of using HMMs for speech recognition).

3.3. Kernel Acoustic Models

To train kernel acoustic models, we use random Fourier features and simply plug the random feature vector $z(\boldsymbol{x})$ (for an acoustic frame $\boldsymbol{x}$) into a multinomial logistic regression model:

$$p(y|\boldsymbol{x}) = \frac{\exp\left(\langle \boldsymbol{\theta}_y,\ z(\boldsymbol{x})\rangle\right)}{\sum_{y'} \exp\left(\left\langle \boldsymbol{\theta}_{y'},\ z(\boldsymbol{x})\right\rangle\right)}. \tag{4}$$

The label y can take any value in $\{1, 2, \ldots, C\}$, each corresponding to a context-dependent phonetic state label, and the parameter matrix $\Theta = [\boldsymbol{\theta}_1 | \ldots | \boldsymbol{\theta}_C]$ is learned. Note that we also include a bias term in our model by appending a 1 to $z(\boldsymbol{x})$ in the equation above.

The model in Equation (4) can be seen as a shallow neural network, with the following properties: (1) the parameters from the inputs (i.e., acoustic feature vectors) to the hidden units are set randomly, and are not learned; (2) the hidden units use $\cos(\cdot)$ as their 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 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.

3.4. Linear Bottleneck

When the number of random features D and the number of phonetic state labels C are large, the $D \times C$ size of the kernel acoustic model parameter matrix Θ can lead to memory and computation issues during training. We can significantly reduce the number of parameters in Θ by using a low-rank factorization $\Theta = UV$; this is called a "linear bottleneck" (Sainath et al., 2013a). This 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 6.

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 is necessary to impose the low-rank decomposition during training, given GPU memory constraints.

4. Random Feature Selection

In this section, we first motivate and describe our proposed feature selection algorithm. We then introduce a new "sparse Gaussian kernel," which performs well in conjunction with the feature selection algorithm.

4.1. Proposed Feature Selection Algorithm

Our proposed random feature selection method, shown in Algorithm 1, is iterative. In each iteration, a model is trained on a set of features using a single pass of stochastic gradient descent (SGD) on a subset of the training data. Then, the features whose corresponding rows in the parameter matrix have the smallest ℓ_2 norms are discarded and replaced with a new set of random features.

This feature selection method has the following advantages: The overall computational cost is mild, as it requires just T passes of SGD through subsets of the data of size R (equivalent to TR/N full SGD epochs). In fact, in our experiments, we find it sufficient to use $R = O(D)$. Moreover, the method is able to explore a large number of non-linear features, while maintaining a compact model. If the number of features s_t selected in iteration t grows linearly each iteration ($s_t = Dt/T$), then the learning algorithm is exposed to $D(T+1)/2$ random features throughout the feature selection process; this s_t sequence is the selection schedule we use in all our experiments. We show in Section 6 that the acoustic models trained on the selected features generally outperform models trained on random 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 FOBOS 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-norms are below a threshold are set to 0. After training completes, the 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. As mentioned in the related work section, our attempts at using FOBOS for feature selection failed, because the magnitude of the regularization parameter needed to produce a sparse model was so large that it dominated the learning process; as a result, the learned models performed badly, and the selected features were essentially random.

Algorithm 1 Random feature selection

input Target number of random features D, data subset size R,
Integers $(s_1, \ldots, s_{T-1})$ such that $0 < s_1 < \cdots < s_{T-1} < D$, specifying selection schedule.

initialize set of selected indices $S := \emptyset$.
for $t = 1, 2, \ldots, T$ **do**
  for $i \in \{1, \ldots, D\} \backslash S$ **do**
    $\boldsymbol{\omega}_i \sim p(\boldsymbol{\omega})$.
    $b_i \sim \mathcal{U}(0, 2\pi)$.
  end for
  if $t \neq T$ **then**
    Initialize parameter matrix Θ.[7]
    Learn weights $\Theta \in \mathbb{R}^{D \times C}$ using a single pass of SGD over R randomly selected training examples, using the projection vectors $(\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D)$, and the biases $(b_1, \ldots, b_D)$, to generate the random Fourier features.
    $S := \{i \mid \Theta_i$ is amongst the s_t rows of Θ with highest ℓ_2 norm$\}$.[8]
  end if
end for
return The selected projection vectors $(\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D)$, and the selected biases $(b_1, \ldots, b_D)$.

One disadvantage of this method is that the selection criterion may misrepresent the features' actual predictive utilities. For instance, the presence of some random feature may increase or decrease the weights for other random features relative to what they would be if that feature were not present. 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 be significantly more computationally expensive. For example, it would require $O(D)$ passes through the data, relative to $O(T)$ passes, which would be prohibitive for large D values. 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.

4.2. A Sparse Gaussian Kernel

In Section 6 we will show that our proposed feature selection algorithm improves the performance of the Laplacian kernel models significantly more than the Gaussian kernel models. In this section, we leverage this insight in order to design a new kernel, the "sparse Gaussian kernel," which we will show also benefits significantly from the feature selection process.

Recall from Table 1 that for the Laplacian kernel, the sampling distribution used for the random Fourier features is the multivariate Cauchy density. Because the Cauchy distribution is fat-tailed, a d-dimensional Cauchy vector will typically contain some entries

7. See Section 6.3 for details on how Θ is initialized.
8. In the case where we are using a linear bottleneck to decompose the parameter matrix Θ into UV, we perform the SGD training using this decomposition. After we complete the training in a given iteration, we compute $\Theta = UV$, and then select features based on the ℓ_2 norms of the rows of Θ.

much larger than the rest. This property of the sampling distribution implies that many of the random features generated via projections with random Cauchy vectors will effectively concentrate on only a few of the input features. We can thus regard such random features as being non-linear combinations of a small number of the original input features. Thus, the proposed feature selection method effectively picks 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 actually choose a small number k of coordinates $F \subseteq \{1, 2, \ldots, d\}$, say, uniformly at random, and then choose the random vector so that it is always zero in positions outside of F. Compared to the random Fourier feature approximation to the Laplacian kernel, the random vectors chosen in this way are truly sparse. From a systems perspective, this sparsity can reduce the memory required for the random projection matrix, and make the random features more efficient to compute (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 $\boldsymbol{\omega}$ are drawn i.i.d. from $\mathcal{N}(0, \sigma^{-2})$, then the corresponding kernel is

$$K(\boldsymbol{x}, \boldsymbol{x}') = \binom{d}{k}^{-1} \sum_{\substack{F \subseteq \{1,\ldots,d\}, \\ |F|=k}} \exp\left(-\frac{\|\boldsymbol{x}_F - \boldsymbol{x}'_F\|_2^2}{2\sigma^2}\right), \tag{5}$$

where $\boldsymbol{x}_F$ is a vector composed of the elements $\boldsymbol{x}_i$ for $i \in F$. We call this kernel the "sparse Gaussian kernel." Note that this kernel puts equal emphasis on all input feature subsets F of size k. However, the feature selection process may effectively bias the distribution of the feature subsets to concentrate on some small family $\mathcal{F}$ of input feature subsets.

5. New Early Stopping Criteria

A challenge for training acoustic models is that the training criterion (e.g., cross-entropy) does not perfectly correlate with the true objective (TER). To partially address this problem, in this section we present several new metrics which we observe correlate with TER significantly better than cross-entropy does. In Section 6 we then show that we can attain improved TER performance by monitoring these metrics on the heldout set during training to decide when to decay the learning rate and stop training. Note that the reason we use these metrics as proxies for the TER, instead of directly using the TER, is that it is very expensive to compute the TER on the development set.

The common thread which unites all the metrics we will present is that they do not penalize very incorrect examples (meaning, examples for which a model assigns a probability very close to 0 to the correct label) as strongly as cross-entropy does. Notice, for instance, that the cross-entropy loss can approach infinity on a single incorrect example. Our metrics are more lenient. We present them now:

1. **Entropy Regularized Log Loss (ERLL)**:[9] This loss rewards models for being confident (low entropy), by considering a weighted sum of the cross entropy loss (CE) and the average entropy (ENT) of the model on the heldout data. Specifically, for any $\beta \in \mathbb{R}$, we define the loss as

$$CE + \beta \cdot ENT = -\frac{1}{N} \sum_{i=1}^{N} \sum_{y=1}^{C} [\mathbb{I}(y = y_i) + \beta \cdot p(y|\boldsymbol{x}_i)] \log p(y|\boldsymbol{x}_i).$$

   This metric encourages models to be more confident, even if it means having a worse cross-entropy loss as a result.

2. **Capped Log Loss**: For any value of $\lambda \geq 0$, we define the "capped log loss" as

$$-\frac{1}{N} \sum_{i=1}^{N} \log(p(y_i|\boldsymbol{x}_i) + \lambda).$$

   Effectively, this loss ensures that no single example contributes more than $-\frac{1}{N}\log(\lambda)$ to the loss. If λ is a small positive number, this loss is very similar to the normal log loss for values of $p(y_i|\boldsymbol{x}_i)$ close to 1, while affecting the loss dramatically for values close to 0 (for example, when $p(y_i|\boldsymbol{x}_i) < \lambda$).

3. **Top-k Log Loss**: For this loss, assume that the heldout examples $(\boldsymbol{x}_i, y_i)$ are sorted in descending order of their $p(y_i|\boldsymbol{x}_i)$ values. Then, for any positive integer $k \leq N$, we can define the "Top-k Log Loss" as

$$-\frac{1}{k} \sum_{i=1}^{k} \log p(y_i|\boldsymbol{x}_i).$$

   This metric judges a model based on how well it does on the k heldout examples to which it assigns highest probabilities.

Notice that for $\beta = 0$, $\lambda = 0$, and $k = N$, these metrics all simplify to the standard cross-entropy loss. In Figure 2, we show plots of the empirical correlations of these metrics with TER values, as a function of each metric's "hyperparameters," based on models we have trained. More specifically, we fully train a large number of kernel and DNN models; we then evaluate the TER performance of these models on the development set, as well as compute the heldout performance of these models in terms of the three metrics described above (for various settings of β, λ, and k). The precise set of models we used are those in Tables 4 and 5. We train models on four data sets: the Cantonese and Bengali data sets for the IARPA Babel program, a 50-hour subset of the broadcast news data set (BN-50), and the TIMIT benchmark task (see Section 6.1 for data set details). We then compute the empirical correlations between the TER values and the different metrics, and plot them as a function of each metric's hyperparameter. Note that for the top-k log loss, we plot the correlation with TER as a function of the fraction $1 - \frac{k}{N}$ of the heldout data set which is *ignored*.

9. In Lu et al. (2016) this metric was called "entropy regularized perplexity" (ERP) for $\beta = 1$.

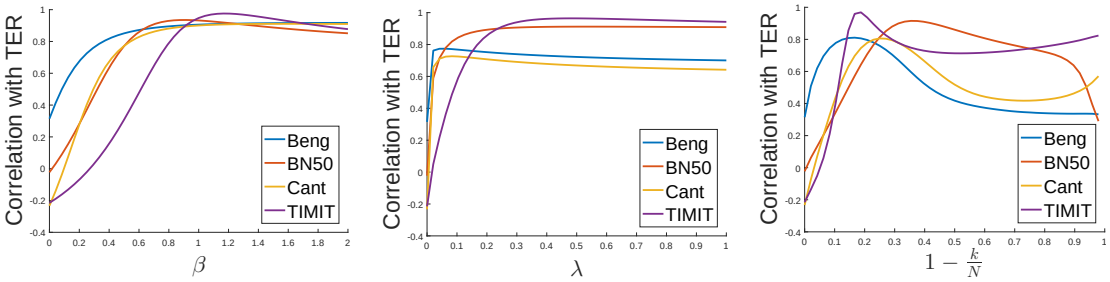

Figure 2: Empirical correlations of (left to right) heldout entropy regularized log loss, capped log loss, and top-k log loss with TER on the development set, as a function of β, λ, and $1 - \frac{k}{N}$, respectively.

As can be seen from these plots, for certain ranges of values of the metric hyperparameters, the correlation of these metrics with TER is quite high. For example, for $\beta = 1$, the correlations between entropy regularized log loss and TER are 0.91, 0.93, 0.90, and 0.95 for Bengali, BN-50, Cantonese, and TIMIT respectively. This is compared to correlations of 0.31, -0.02, -0.23, and -0.21 for the cross-entropy objective.

One conjecture for why these metrics attain higher correlation with ASR performance than heldout cross-entropy is because there is inherent noise in the labels on which the acoustic models are trained. The labels are noisy because they are generated via a forced alignment between the correct transcription and the audio using a GMM/HMM acoustic model, as discussed in Section 6.1. Thus, by not penalizing a model's mistakes on the incorrect labels as harshly, and instead focusing on the model's performance on the clean labels, these metrics better capture the quality of the model for the downstream ASR task. Better understanding this phenomenon is an interesting area for future work.

Because of the high correlation between these metrics and ASR performance, we propose to monitor these metrics on the heldout set during training to decide when to decay the learning rate and stop training (details in Section 6.3). In Section 6.4 we present results using this learning rate decay method with the ERLL metric with $\beta = 1$, and show that it leads to improvements in TER for both kernel and DNN models. We note that we could have also used the other metrics for this purpose, but chose to use ERLL with $\beta = 1$ since we observed that it attained high correlation values with TER across all four data sets.

6. Experiments

We now present our empirical results comparing the performance of fully-connected DNNs with kernel approximation methods for acoustic modeling, across four data sets. We leverage the three methods we have discussed—feature selection, the new early stopping criterion, and linear bottlenecks—and show that using these three methods the kernel models perform very similarly to the DNNs across the four test sets, attaining token error rates between 0.5% better and 0.1% worse than the DNNs.

In this section, we first provide a description of the data sets we use, and our evaluation criteria. We then give an overview of our training procedure, and provide details regarding

hyperparameter choices. We then present our experimental results comparing the performance of kernel approximation methods to DNNs. Lastly, we discuss the impact of the number of random features on performance, and take a deeper look at the dynamics of the feature selection process.

6.1. Data Sets

We train DNN and kernel acoustic models, as described in Section 3, on four data sets: the IARPA Babel Program Cantonese (IARPA-babel101-v0.4c) and Bengali (IARPA-babel103b-v0.4b) limited language packs, a 50 hour subset of the Broadcast News data set (BN-50) (Kingsbury, 2009; Sainath et al., 2011), and the TIMIT benchmark data set (Garofolo et al., 1993). Each data set is partitioned 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., the learning rate). We then run decoding on the development set using IBM's Attila speech recognition toolkit (Soltau et al., 2010), to select a small subset of models which perform best in terms of TER (the best kernel and DNN model per data set). We tune the acoustic model weight on the development set to optimize the relative contributions of the language model and the acoustic model to the final score assigned to a given word sequence. Finally, we evaluate the TER performance on the test set using the selected group of models (using, for each model, the best acoustic model weight on the development set), 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 Bengali and Cantonese Babel data sets both include training and development sets of approximately 20 hours, and an approximately 5 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 comparatively small.

For the Broadcast News data set, we use 45 hours of audio for training, and 5 hours as a heldout set. For the development set, we use the EARS Dev-04f data set (as described by Kingsbury, 2009), which consists of approximately three hours of broadcast news from various news shows. We use the DARPA EARS RT-03 English Broadcast News Evaluation Set (Fiscus et al., 2003) as our test set, consisting of 72 five minute conversations.

The TIMIT data set 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,

Data set	Train	Heldout	Dev	Test	# features	# classes
Beng.	21 hr (7.7M)	2.8 hr (1.0M)	20 hr (7.1M)	5 hr (1.7M)	360	1000
BN-50	45 hr (16M)	5 hr (1.8M)	2 hr (0.7M)	2.5 hr (0.9M)	360	5000
Cant.	21 hr (7.5M)	2.5 hr (0.9M)	20 hr (7.2M)	5 hr (1.8M)	360	1000
TIMIT	3.2 hr (2.3M)	0.3 hr (0.2M)	0.15 hr (0.1M)	0.15 hr (0.1M)	440	147

Table 2: Data set details. We report the size of each data set partition in terms of the number of hours of speech, and in terms of the number of acoustic frames (in parentheses).

and divisions of the data set as Huang et al. (2014) and Chen et al. (2016), which allows direct comparison of our results with theirs (Table 8).

The acoustic features, representing 25 ms acoustic frames with context, are real-valued dense vectors. A 10 ms shift is used between adjacent frames (except on TIMIT, where a 5 ms shift is used). For the Cantonese, Bengali, and Broadcast News data sets we use a standard 360-dimensional speaker-adapted representation used by IBM (Kingsbury et al., 2013); these feature vectors correspond to the concatenation of nine 40-dimensional vectors corresponding to the four frames before and after the current frame. For the TIMIT experiments, we use 40-dimensional feature space maximum likelihood linear regression (fMLLR) features (Gales, 1998), and concatenate the five neighboring frames in either direction, for a total of eleven frames and 440 features.

The state labels are obtained via forced alignment using a GMM/HMM system. This forced alignment is necessary because there is a mismatch between the ground truth available for training (a word sequence transcribing each utterance) and the actual labels needed for training (precise assignment of HMM states to frames). The Cantonese and Bengali data sets 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 data set has 147 context-independent labels, corresponding to the beginning, middle, and end of 49 phonemes.

The language models we use are all n-gram language models estimated using modified Kneser-Ney smoothing, with n values of 2, 4, 3, and 3 for Bengali, Broadcast News, Cantonese, and TIMIT, respectively. The TIMIT language model is a phone-level model. The Bengali and Cantonese language models are particularly small (approximately 60,000 bigrams and 136,000 trigrams, respectively), trained using only the provided audio transcripts. The Broadcast News model is small as well, containing only 3.3 million 4-grams.

In Table 2 we provide details on all the data sets, as well as on their number of features and classes. All our data sets have millions of training points, significantly exceeding the size of typical machine learning tasks tackled by kernel methods. Additionally, the large number of output classes for our data sets 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).

6.2. Evaluation Metrics

We use five metrics to evaluate the DNN and kernel acoustic models:

1. **Cross-entropy (CE)**: Given examples, $\{(\boldsymbol{x}_i, y_i), i = 1 \ldots N\}$, the cross-entropy is defined as
$$-\frac{1}{N}\sum_{i=1}^{N} \log p(y_i|\boldsymbol{x}_i). \tag{6}$$

2. **Average Entropy (ENT)**: The average entropy of a model is defined as
$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{C} p(y|\boldsymbol{x}_i) \log p(y|\boldsymbol{x}_i).$$
If a model has low average entropy, it is generally confident in its predictions.

3. **Entropy Regularized Log Loss (ERLL)**: Defined in Section 5. We use $\beta = 1$ unless specified otherwise.

4. **Classification Error (ERR)**: The classification error is defined as
$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[y_i \neq \operatorname*{arg\,max}_{y \in 1,2,\ldots,C} p(y|\boldsymbol{x}_i)\right].$$

5. **Token Error Rate (TER)**: We feed the predictions of the acoustic models, which correspond to probability distributions over the phonetic states, to the rest of the ASR pipeline and calculate the misalignment between the decoder's outputs and the ground-truth transcriptions. 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 data set, to its corresponding metric.

6.3. Details of Acoustic Model Training

We train all our kernel models with either the Laplacian, the Gaussian, or the sparse Gaussian (Section 4.2) kernels. These kernel models typically have three hyperparameters: the kernel bandwidth (σ for the Gaussian kernels, λ for the Laplacian kernel; see Table 1), the number of random projections, and the initial learning rate. We try various numbers of random features, ranging from 5000 to 400,000. 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. The sparse Gaussian kernel additionally has the hyperparameter k which specifies the sparsity of each random projection vector $\boldsymbol{\omega}_i$. For all experiments, we use $k = 5$.

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 four 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 3 we show how depth affects recognition performance on the Broadcast News data set. Additionally, all our DNNs use

	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 3: 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 set. All of these models were trained using a linear bottleneck for the output parameter matrix, and using ERLL for learning rate decay. The best result is in bold.

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 data sets.

For both DNN and kernel models, we use stochastic gradient descent (SGD) as our optimization algorithm for minimizing the training cross-entropy objective. We use mini-batches of size of 250 or 256 samples during training, and we use the heldout set to tune the other hyperparameters. We use the learning rate decay scheme described in (Morgan and Bourlard, 1990; Sainath et al., 2013a,b), which monitors performance on the heldout set 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. Instead of using the heldout cross-entropy, in some of our experiments we use the heldout ERLL to decide when to decay the learning rate. We terminate training once we have divided the learning rate 10 times.

As mentioned in Section 3.4, one effective way of reducing the number of parameters in our models is use a linear bottleneck, which is a low-rank factorization of the output parameter matrix (Sainath et al., 2013a). We use bottlenecks of size 1000, 250, 250, and 100 for BN-50, Bengali, Cantonese, and TIMIT, respectively. We train models both with and without this technique; the only exception is that we are unable to train BN-50 kernel models without the bottleneck of size 1000, due to memory constraints on our GPUs.

We initialize our DNN parameters uniformly at random within $[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, \frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}]$, 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 3, and we initialize the parameter matrix Θ as the zero matrix. When using a linear bottleneck to decompose the parameter matrix, we initialize the resulting two matrices randomly, like we do for our DNNs.

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), but ultimately we use all N training examples once the random features are selected. Thus, each iteration of feature selection has equivalent computational cost to a R/N fraction of an SGD epoch (roughly 1/7 or 2/7 for $D < 10^5$ and $D \geq 10^5$ respectively, on the Babel speech data sets). 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 seven (or fourteen) epochs of training on the Babel data sets. For the Broadcast News data set, it corresponds to the cost of approximately six full epochs of training (for $D \geq 10^5$).

All our training code is written in MATLAB, leveraging its GPU features, and executed on Amazon EC2 machines.

6.4. Results

In this section, we report the results from our experiments comparing kernel methods to DNNs on ASR tasks. We first discuss the ablation study showing the marginal improvements to the kernel model performance attained by feature selection, the new early stopping criterion, and the linear bottlenecks; we show analogous results for DNNs for the early stopping and linear bottleneck methods. We then compare for each data set the performance of the best kernel and DNN models from these detailed experiments; the most important result from these comparisons, shown in Table 7, is that the kernel models are competitive with the DNNs, attaining TER values between 0.5% better and 0.1% worse than the DNNs across our four test sets.

For our detailed experiments, for both DNN and kernel methods we train models with and without linear bottlenecks, and with and without using ERLL to determine the learning rate decay. For our kernel methods, we additionally train models with and without using feature selection. We run experiments with all three kernels (Laplacian, Gaussian, sparse Gaussian) and we use 100,000 random features on all data sets except for TIMIT, where we are able to use 200,000 random features (because the output dimensionality is lower). As mentioned in the previous section, for our DNN experiments, we train models with four hidden layers,[10] using the tanh activation function, and using either 1000, 2000, or 4000 hidden units per layer. We focus on comparing the performance of these methods in terms of TER, but we also report results for other metrics. Unless specified otherwise, all TER results are on the development set, and all cross-entropy, entropy, classification error, and ERLL results are on the heldout set.

In Tables 4 and 5, we show the TER results for our DNN and kernel models, respectively, across all data sets. There are many things to notice about these results. For our DNN models, linear bottlenecks almost always lower TER values, though in a few cases they have no effect on TER. Using ERLL 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 with 4000 hidden units, and TIMIT with 2000 and 4000). The DNNs with 4000 hidden units typically attain the best results, though on a couple of data sets they are matched or narrowly beaten by the 2000 models.

For our kernel models, we see that incorporating a linear bottleneck brings large drops in TER across the board.[11] Performing feature selection generally improves TER as well; we see that it 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

10. As mentioned in Section 6.3, we find that this is generally the best setting.
11. Recall that we are unable to train BN-50 kernel models *without* using a bottleneck because the resulting models would not fit on our GPUs.

instances in which feature selection hurts TER (see Section 6.6 for discussion). Second, we see that using heldout ERLL to determine when to decay the learning rate helps all our kernel models attain lower TER values. Next, 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. 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 6.6). In general, the kernel function which performed best, across the majority of settings, was the sparse Gaussian kernel.

To further improve the performance of the kernel models, we train models with up to 400k features on Broadcast News, our most challenging data set. 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 6. All of these models were trained with a linear bottleneck of size 1000, and using ERLL for learning rate decay. We include results using 100k random features in this table for comparison. As we can observe, our best kernel model on BN-50 now attains a TER of 16.4%, which 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 the 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 7, we compare for each data set the performance of the best DNN model with the best kernel model, across six 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 6 with 200k and 400k random features, along with those in Table 5, in the process of finding the best performing models. In terms of heldout cross-entropy, the kernels outperformed the DNNs on Cantonese and TIMIT, while the DNNs beat the kernels on Bengali and BN-50. With regard to classification error, the kernels beat the DNNs on all data sets 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 now discuss the results on the test sets. First of all, to avoid overfitting to the test sets, for each data set 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 7 thus contains all the test results we collected.[12] As one can see, the relative test performance

12. The only exception is on the Broadcast News data set. Prior to training the best performing 400k model, we had already evaluated the best 100k model, which attained a TER of 11.9% on the test set.

	1000				2000				4000			
	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: DNN TER Results (development set): This table shows TER results for 4-hidden layer DNNs with 1000, 2000, or 4000 hidden units per layer. 'NT' specifies that no "tricks" were used during training (no bottleneck, did not use ERLL for learning rate decay). A 'B' specifies that a linear bottleneck was used; an 'R' specifies that ERLL was used for learning rate decay (so 'BR' means both were used). The best result for each data set is in bold.

	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: Kernel TER Results (development set): This table shows TER results for our kernel experiments using either the Laplacian, Gaussian, or Sparse Gaussian kernels, with 100k random features (except for TIMIT, which uses 200k features). 'NT' specifies that no "tricks" were used during training (no bottleneck, did not use ERLL for learning rate decay). A 'B' specifies that a linear bottleneck was used; an 'R' specifies that ERLL was used for the learning rate decay (so 'BR' means both were used). '+FS' specifies that feature selection was used. The best result for each row is in bold.

	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 6: 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 ERLL for learning rate decay.

	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
ERLL	**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 7: Table comparing the Best DNN ('D') and kernel ('K') results, across four data sets and six metrics. The first four 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 6 are included in the set of models from which the best performing model is picked (for each metric). See Section 6.2 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 8: 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.

of the DNN and kernel models is quite similar to the development set results; the DNNs perform better by 0.5% on Cantonese, 0.1% on TIMIT and Broadcast News, and perform worse by 0.1% on Bengali. For direct comparison on the TIMIT data set, we include in Table 8 the test results for the best DNN and kernel models from Huang et al. (2014) and Chen et al. (2016). As mentioned in Section 6.1, 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 comparable. 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 all these results is that the kernels (narrowly) beat the DNNs on Broadcast News, our largest and most challenging data set, and one which has been used extensively in large scale speech recognition research. In Appendix A, we include more detailed tables comparing the various models we trained across all the metrics from Section 6.2.

6.5. Importance of the Number of Random Features

We will now illustrate the importance of the number of random features D on model performance. For this purpose, we trained kernel models on the BN-50 data set using $D \in \{5000, 10000, 25000, 50000, 100000\}$ for the three kernel functions, with and without feature selection, with a linear bottleneck of size 1000, and using heldout cross-entropy for learning rate decay. In Figure 3, we show how increasing the number of features dramat-

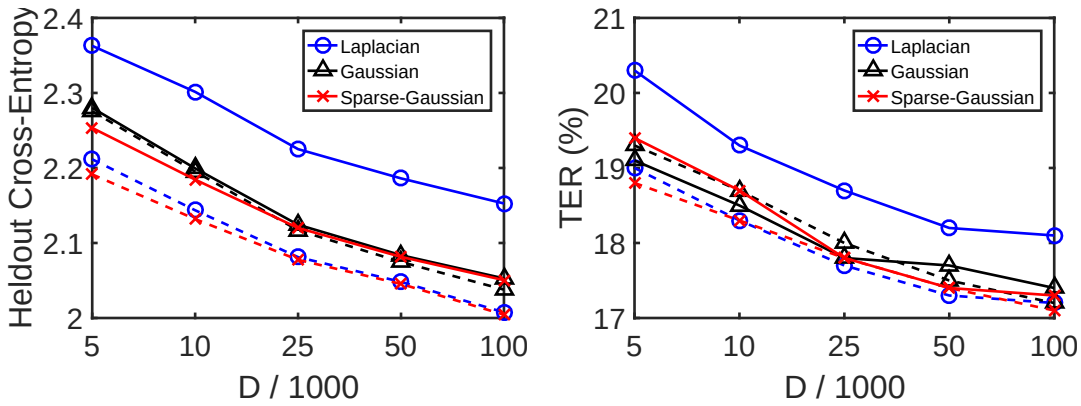

Figure 3: Heldout cross-entropy and development set TER performance of kernel acoustic models on BN-50, as a function of the number of random features D. Dashed lines mean feature selection was performed, while solid lines mean it was not.

ically 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 50,000 to 100,000. Interestingly, we can also observe in these plots that the benefits from feature selection (gap between the dashed and solid lines) are large for the Laplacian kernel, modest for the sparse Gaussian kernel, and insignificant for the Gaussian kernel. We provide insight into why these trends occur in the following section.

6.6. Dynamics of Random Feature Selection

We now explore the feature selection dynamics. We first show that features selected in early iterations are likely to be selected in all subsequent iterations, demonstrating that the selection criterion is consistent. Second, we show that the feature selection process can be seen as way of learning to upweight important input features. This effect is pronounced for the Laplacian and sparse Gaussian kernels, but not for the Gaussian kernel, helping to explain why the former methods benefit from feature selection more than the latter.

In Figure 4, we plot the fraction of the s_t features selected in iteration t that remain in the model after all T iterations ("survival rate"). We show results for kernel models trained on the Cantonese data set, without using linear bottlenecks, and using heldout cross-entropy for learning rate decay. In nearly all iterations and for all kernels, over half of the selected features survive to the final model. For instance, over 90% of the Laplacian kernel features selected at iteration 10 survive the remaining 40 rounds of selection. For comparison, we plot the expected survival rate if at each iteration the features were chosen uniformly at random; for $s_t = Dt/T$, the expected survival rate for features selected at iteration t is $T!/(t! \cdot T^{T-t})$, which is exponentially small in T when $t \leq \beta T$ for any fixed $\beta < 1$.[13] For example, at $t = 10$ the expected survival rate is approximately 9×10^{-11} with $T = 50$. Thus, our selection criterion is orders of magnitude more consistent than random selection.

13. This can be shown using Stirling's formula. See Jameson (2015) for a useful review.

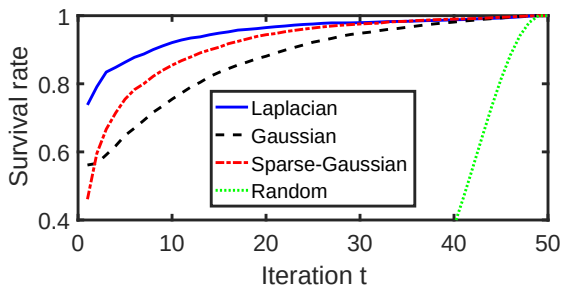

Figure 4: Fraction of the s_t features selected in iteration t that are in the final model (survival rate) for kernel models trained on the Cantonese data set.

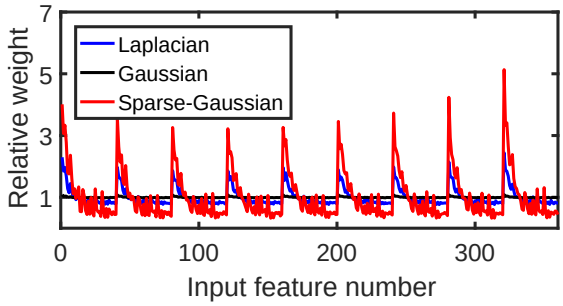

Figure 5: The relative weight of each input feature in the projection matrix Ω after feature selection completes, for kernel models trained on the Cantonese data set.

Finally, we consider how the feature selection process can be seen as a way of learning to assign more weight to the more important input features. Consider the final matrix of random vectors $\Omega := [\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D] \in \mathbb{R}^{d \times D}$ after feature selection, for the Cantonese models from Figure 4. 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 Ω. In Figure 5, we plot $\sum_{j=1}^{D} |\Omega_{i,j}|/Z$ for each input feature $i \in \{1, 2, \ldots, d\}$, where $Z = \frac{1}{d}\sum_{i,j} |\Omega_{i,j}|$ is a normalization term.[14] There is a strong periodic effect as a function of the input feature number. An examination of the feature pipeline from Kingsbury et al. (2013) reveals that this is because these features vectors are the concatenation of nine 40-dimensional acoustic feature vectors, where each set of 40 features is 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, for 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 $O(\sqrt{\log(d)})$. This observation helps explain why feature selection significantly improves the Laplacian and sparse Gaussian models, but not the Gaussian models.

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

7. Conclusion

In this paper, we explore the performance of kernel methods on large-scale ASR tasks, leveraging the kernel approximation technique of Rahimi and Recht (2007). We propose two new methods (feature selection, new early stopping criteria) which lead to large improvements in the performance of kernel acoustic models. We further show that using a linear bottleneck (Sainath et al., 2013a) to decompose the parameter matrices of these kernel models leads to significant improvements in performance as well. We validate these findings on four data sets, including the Broadcast News and TIMIT benchmark tasks. Using all these methods together, the kernel methods attain comparable speech recognition performance to DNNs across the four test sets; on Cantonese, TIMIT, and Broadcast News, the kernel models outperform the DNNs by 0.5%, 0.1%, and 0.1% TER absolute, respectively, whereas on Bengali, the DNN does better by 0.1% TER.

For future work, we are interested in continuing to test the limits of kernel methods on challenging empirical tasks. For example, can kernel methods be adapted to compete with convolutional and recurrent neural networks?

Acknowledgments

This research is supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD / ARL) contract number W911NF-12-C-0012. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

F.S. is grateful to Lawrence K. Saul (UCSD), Léon Bottou (Facebook), Alex Smola (Amazon), and Chris J.C. Burges (Microsoft Research) for many fruitful discussions and pointers to relevant work. Additionally, A.B.G. is partially supported by a USC Provost Graduate Fellowship. F.S. is partially supported by a NSF IIS-1065243, 1451412, 1139148 a Google Research Award, an Alfred. P. Sloan Research Fellowship, an ARO YIP Award (W911NF-12-1-0241) and ARO Award W911NF-15-1-0484. A.B. is partially supported by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020. Computation for the work described in this paper was partially supported by the University of Southern California's Center for High-Performance Computing (`http://hpc.usc.edu`).

Appendix A. Detailed Empirical Results

In this appendix, we include tables comparing the models we trained in terms of four different metrics (CE, ENT, ERR, and ERLL). The notation is the same as in Tables 4 and 5. For both DNN and kernel models, 'NT' specifies that no "tricks" were used during training (no bottleneck, did not use ERLL for learning rate decay). A 'B' specifies that a linear bottleneck was used for the output parameter matrix, while an 'R' specifies that ERLL was used for learning rate decay (so 'BR' means both were used). For kernel models, '+FS' specifies that feature selection was performed for the corresponding row. The best result for each metric and data set is in bold.

Some important things to take note of in these tables:

- The linear bottleneck typically causes large drops 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 ERLL 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 ERLL is typically lower for models that use this method (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 heldout ERLL across the board, as well as TER in the vast majority of cases.

	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 9: Kernel: Metric CE

	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 10: DNN: 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 11: Kernel: Metric ENT

	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 12: DNN: 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 13: Kernel: Metric ERLL

	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 14: DNN: Metric ERLL

	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 15: Kernel: Metric ERR

	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 16: DNN: Metric ERR

Appendix B. Derivation of Functional Form 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,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{\boldsymbol{\omega},b}\left[\sqrt{2}\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} + b) \cdot \sqrt{2}\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}' + b)\right], \tag{7}$$

where $\boldsymbol{\omega}$ is drawn from $p(\boldsymbol{\omega})$, the inverse Fourier transform of k, and b is drawn uniformly from $[0, 2\pi]$. We begin this proof using Equation (3) from Section 3.1:

$$\begin{aligned}
k(\boldsymbol{x}, \boldsymbol{x}') &= \int_{R^d} p(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}')}\, d\boldsymbol{\omega} \\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[e^{j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}} e^{-j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}'}\right] \\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[\big(\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}) + j\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\big)\big(\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}') - j\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\big)\right] \\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}') + \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\right] \\
&+ j \cdot \mathbb{E}_{\boldsymbol{\omega}}\left[\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}') - \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\right] \\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}') + \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\right]
\end{aligned} \tag{8}$$

Note the Equation (8) is true because we know that $k(\boldsymbol{x}, \boldsymbol{x}')$ 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 (7) is equal to this same expression:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{\omega},b}\left[\sqrt{2}\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} + b) \cdot \sqrt{2}\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}' + b)\right] \\
&\quad = 2 \cdot \mathbb{E}_{\boldsymbol{\omega},b}\Bigg[\Bigg(\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(b) - \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\sin(b)\Bigg) \cdot \\
&\qquad\qquad \Bigg(\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\cos(b) - \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\sin(b)\Bigg)\Bigg] \qquad (9) \\
&\quad = 2 \cdot \mathbb{E}_{\boldsymbol{\omega},b}\Bigg[\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\cos^2(b) \\
&\qquad\qquad - \cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\cos(b)\sin(b) \\
&\qquad\qquad - \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\cos(b)\sin(b) \\
&\qquad\qquad + \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\sin^2(b)\Bigg] \\
&\quad = 2 \cdot \mathbb{E}_{\boldsymbol{\omega}}\left[\frac{1}{2}\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}') + \frac{1}{2}\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\right] \qquad (10) \\
&\quad = \mathbb{E}_{\boldsymbol{\omega}}\left[\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\cos(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}') + \sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x})\sin(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}')\right] \\
&\quad = k(\boldsymbol{x}, \boldsymbol{x}')
\end{aligned}$$

Equation (9) is true by the cosine sum of angles formula, and Equation (10) 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. ■

References

Naman Agarwal, Zeyuan Allen Zhu, Brian Bullins, Elad Hazan, and Tengyu Ma. Finding approximate local minima faster than gradient descent. In *STOC*, 2017.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep Speech 2 : End-to-end speech recognition in English and Mandarin. In *ICML*, 2016.

Animashree Anandkumar and Rong Ge. Efficient approaches for escaping higher order saddle points in non-convex optimization. In *COLT*, 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 *ACL*, 2016.

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 *ICML*, 2017.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, 2014.

Lalit R Bahl, Peter F Brown, Peter V De Souza, and Robert L Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *ICASSP*, 1986.

Peter L. Bartlett. For valid generalization the size of the weights is more important than the size of the network. In *NIPS*, 1996.

Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, 34(5):1–41, 2007.

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.

Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston. *Large-Scale Kernel Machines*. MIT Press, 2007.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *ICASSP*, pages 4960–4964. IEEE, 2016.

Jie Chen, Lingfei Wu, Kartik Audhkhasi, Brian Kingsbury, and Bhuvana Ramabhadran. Efficient one-vs-one kernel ridge regression for speech recognition. In *ICASSP*, 2016.

Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Ekaterina Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition with sequence-to-sequence models. In *ICASSP*, 2018.

Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Trans. Algorithms*, 6(4):63:1–63:30, 2010.

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

George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Trans. Audio, Speech & Language Processing*, 20(1):30–42, 2012.

Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *NIPS*, 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 *NIPS*, 2014.

Dennis DeCoste and Bernhard Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002.

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.

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

Jonathan Fiscus, George Doddington, Audrey Le, Greg Sanders, Mark Przybocki, and David Pallett. 2003 NIST Rich Transcription evaluation data. *Linguistic Data Consortium*, 2003. URL `https://catalog.ldc.upenn.edu/LDC2007S10`.

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

Mark J. F. Gales and Steve J. Young. The application of hidden Markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2007.

John S. Garofolo, Lori F. Lamel, William M. Fisher, Jonathan G. Fiscus, David S. Pallett, Nancy L. Dahlgren, and Victor Zue. TIMIT acoustic phonetic continuous speech corpus. *Linguistic Data Consortium*, 1993. URL `https://catalog.ldc.upenn.edu/LDC93S1`.

Matthew Gibson and Thomas Hain. Hypothesis spaces for minimum Bayes risk training in large vocabulary speech recognition. In *INTERSPEECH*, 2006.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

Raffay Hamid, Ying Xiao, Alex Gittens, and Dennis DeCoste. Compact random feature maps. In *ICML*, 2014.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara 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, 2012.

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

Po-Sen Huang, Haim Avron, Tara N. Sainath, Vikas Sindhwani, and Bhuvana Ramabhadran. Kernel methods match deep neural networks on TIMIT. In *ICASSP*, 2014.

G.J.O. Jameson. A simple proof of Stirling's formula for the gamma function. *The Mathematical Gazette*, 99(544):68–74, 2015.

Janez Kaiser, Bogomir Horvat, and Zdravko Kacic. A novel loss function for the overall risk criterion based discriminative training of HMM models. In *INTERSPEECH*, 2000.

Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In *AISTATS*, 2012.

Brian Kingsbury. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *ICASSP*, 2009.

Brian Kingsbury, Jia Cui, Xiaodong Cui, Mark J. F. Gales, Kate Knill, Jonathan Mamou, Lidia Mangu, David Nolden, Michael Picheny, Bhuvana Ramabhadran, Ralf Schlüter, Abhinav Sethy, and Philip C. Woodland. A high-performance Cantonese keyword search system. In *ICASSP*, 2013.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Quoc V. Le, Tamás Sarlós, and Alexander J. Smola. Fastfood - computing Hilbert space expansions in loglinear time. In *ICML*, 2013.

Zhiyun Lu, Dong Guo, 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 *ICASSP*, 2016.

Avner May, Michael Collins, Daniel J. Hsu, and Brian Kingsbury. Compact kernel models for acoustic modeling via random feature selection. In *ICASSP*, 2016.

Charles A. Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *Journal of Machine Learning Research*, 7:2651–2667, 2006.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

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

Abdel-rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton. Acoustic modeling using deep belief networks. *IEEE Trans. Audio, Speech & Language Processing*, 20(1):14–22, 2012.

Guido F. Montúfar, Razvan Pascanu, KyungHyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *NIPS*, 2014.

N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *NIPS*, 1990.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *ICLR (Workshop)*, 2015.

Jeffrey Pennington and Yasaman Bahri. Geometry of neural network loss surfaces via random matrix theory. In *ICML*, 2017.

Jeffrey Pennington, Felix X. Yu, and Sanjiv Kumar. Spherical random features for polynomial kernels. In *NIPS*, 2015.

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

Daniel Povey and Brian Kingsbury. Evaluation of proposed modifications to MPE for large scale discriminative training. In *ICASSP*, 2007.

Daniel Povey and Philip C. Woodland. Minimum phone error and I-smoothing for improved discriminative training. In *ICASSP*, 2002.

Daniel Povey, Dimitri Kanevsky, Brian Kingsbury, Bhuvana Ramabhadran, George Saon, and Karthik Visweswariah. Boosted MMI for model and feature-space discriminative training. In *ICASSP*, 2008.

Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *INTERSPEECH*, 2016.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

Tara N. Sainath, Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novák, and Abdel-rahman Mohamed. Making deep belief networks effective for large vocabulary continuous speech recognition. In *ASRU*, 2011.

Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*, 2013a.

Tara N. Sainath, Brian Kingsbury, Hagen Soltau, and Bhuvana Ramabhadran. Optimization techniques to improve training speed of deep neural networks for large speech tasks. *IEEE Trans. Audio, Speech & Language Processing*, 21(11):2267–2276, 2013b.

Tara N. Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *ICASSP*, 2013c.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, 2014.

George Saon, Tom Sercu, Steven J. Rennie, and Hong-Kwang Jeff Kuo. The IBM 2016 English conversational telephone speech recognition system. In *INTERSPEECH*, 2016.

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.

B. Schölkopf and A. Smola. *Learning with kernels*. MIT Press, 2002.

Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *ASRU*, 2011a.

Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *INTERSPEECH*, 2011b.

Tom Sercu and Vaibhava Goel. Advances in very deep convolutional neural networks for LVCSR. In *INTERSPEECH*, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Hagen Soltau, George Saon, and Brian Kingsbury. The IBM Attila speech recognition toolkit. In *SLT*, 2010.

Hagen Soltau, George Saon, and Tara N. Sainath. Joint training of convolutional and non-convolutional neural networks. In *ICASSP*, 2014.

Sören Sonnenburg and Vojtech Franc. COFFIN: a computational framework for linear SVMs. In *ICML*, 2010.

Ingo Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In *NIPS*, 2003.

Nikko Ström. Sparse connection and pruning in large dynamic artificial neural networks. In *EUROSPEECH*, 1997.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *INTERSPEECH*, 2012.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

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.

V. Valtchev, J. J. Odell, Philip C. Woodland, and Steve J. Young. MMIE training of large vocabulary recognition systems. *Speech Communication*, 22(4):303–314, 1997.

Ewout van den Berg, Bhuvana Ramabhadran, and Michael Picheny. Training variance and performance evaluation of neural networks in speech. In *ICASSP*, 2017.

Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):480–492, 2012.

Karel Veselý, Arnab Ghoshal, Lukás Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *INTERSPEECH*, 2013.

Christopher K. I. Williams and Matthias W. Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, 2000.

Bo Xie, Yingyu Liang, and Le Song. Diverse neural network learns true target functions. In *AISTATS*, 2017.

Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Michael L. Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Toward human parity in conversational speech recognition. *IEEE/ACM Trans. Audio, Speech & Language Processing*, 25(12):2410–2423, 2017.

Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH*, 2013.

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alexander J. Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *ICCV*, 2015.

Ian En-Hsu Yen, Ting-Wei Lin, Shou-De Lin, Pradeep Ravikumar, and Inderjit S. Dhillon. Sparse random feature algorithm as coordinate descent in Hilbert space. In *NIPS*, 2014.

Felix X. Yu, Sanjiv Kumar, Henry A. Rowley, and Shih-Fu Chang. Compact nonlinear maps and circulant extensions. *arXiv preprint arXiv:1503.03893*, 2015.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.