How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets

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

HAL Id: hal-01666934

https://inria.hal.science/hal-01666934

Submitted on 18 Dec 2017

How to Scale Up Kernel Methods
to Be As Good As Deep Neural Nets

Zhiyun Lu[1†], Avner May[2†]
Kuan Liu[1‡], Alireza Bagheri Garakani[1‡], Dong Guo[1‡], Aurélien Bellet[4‡*]
Linxi Fan[2], Michael Collins[2], Brian Kingsbury[3], Michael Picheny[3], Fei Sha[1¶]

[1] Dept. of Computer Science, U. of Southern California, Los Angeles, CA 90089
{zhiyunlu, kuanl, bagherig, dongguo, feisha}@usc.edu

[2] Dept. of Computer Science, Columbia University, New York, New York 10027
{avnermay, mcollins}@cs.columbia.edu, lf2422@columbia.edu

[3] IBM T. J. Watson Research Center, Yorktown Heights, NY 10598
{bedk, picheny}@us.ibm.com

[4] LTCI UMR 5141, Télécom ParisTech & CNRS, France
aurelien.bellet@telecom-paristech.fr

[†] and [‡]: shared first and second co-authorships, respectively

[¶]: to whom questions and comments should be sent

December 18, 2017

Abstract

The computational complexity of kernel methods has often been a major barrier for applying them to large-scale learning problems. We argue that this barrier can be effectively overcome. In particular, we develop methods to scale up kernel models to successfully tackle large-scale learning problems that are so far only approachable by deep learning architectures. Based on the seminal work by [38] on approximating kernel functions with features derived from random projections, we advance the state-of-the-art by proposing methods that can efficiently train models with hundreds of millions of parameters, and learn optimal representations from multiple kernels. We conduct extensive empirical studies on problems from image recognition and automatic speech recognition, and show that the performance of our kernel models matches that of well-engineered deep neural nets (DNNs). To the best of our knowledge, this is the first time that a direct comparison between these two methods on large-scale problems is reported. Our kernel methods have several appealing properties: training with convex optimization, cost for training a single model comparable to DNNs, and significantly reduced total cost due to fewer hyperparameters to tune for model selection. Our contrastive study between these two very different but equally competitive models sheds light on fundamental questions such as how to learn good representations.

---

[*]Most of the work in this paper was carried out while the author was affiliated with Department of Computer Science, University of Southern California.

1 Introduction

Deep neural networks (DNNs) and other types of deep learning architecture have made significant advances [3, 4]. In both well-benchmarked tasks and real-world applications, such as automatic speech recognition [21, 34, 44] and image recognition [29, 48], deep learning architectures have achieved an unprecedented level of success and have generated major impact.

Arguably, the most instrumental factors contributing to their success are: (1) learning from a huge amount of training data for highly complex models with millions to billions of parameters; (2) adopting simple but effective optimization methods such as stochastic gradient descent; (3) combatting overfitting with new schemes such as drop-out [23]; and (4) computing with massive parallelism on GPUs. New techniques as well as "tricks of the trade" are frequently invented and added to the toolboxes for machine learning researchers and practitioners.

In stark contrast, there have been many fewer publicly known successful applications of kernel methods (such as support vector machines) to problems at a scale comparable to the speech and image recognition problems tackled by DNNs. This is a surprising chasm, noting that kernel methods have been extensively studied both theoretically and empirically for their power of modeling highly nonlinear data [43]. Moreover, the connection between kernel methods and (infinite) neural networks has also been long noted [35, 51, 11].

Nonetheless, a common misconception is that it may be difficult, if not impossible, for kernel methods to catch up with deep learning methods in addressing large-scale learning problems. In particular, many kernel-based algorithms scale quadratically in the number of training samples. This barrier in computational complexity makes it especially challenging for kernel methods to reap the benefits of learning from a very large amount of data, while deep learning architectures are especially adept at it.

We contend that this skepticism can be sufficiently attenuated. Concretely, in this paper, we investigate and propose new ideas tailored for kernel methods, with the aim of scaling them up to take on challenging problems in computer vision and automatic speech recognition.

To this end, we build on the work by [38] on approximating kernel functions with features derived from random projections. Our innovation is, however, to advance the state-of-the-art to a much larger scale. Concretely, we propose fast training methods for models with hundreds of millions of parameters — these methods are necessary for classifiers using hundreds of thousands of features to recognize thousands of categories. We also propose scalable methods for combining multiple kernels as ways of learning feature representations. Interestingly, we show multiplicative combination of kernels scale better than additive combinations.

We validate our approaches with extensive empirical studies. We contrast kernel models to DNNs on 4 large-scale benchmark datasets, some of which are often used to demonstrate the effectiveness of DNNs. We show that the performance of large-scale kernel models approaches or surpasses their deep learning counterparts, which are either exhaustively optimized by us or are well-accepted as yardsticks in industry standards.

While providing a recipe to obtain state-of-the-art large-scale kernel models, another important contribution of our work is to shed light on new perspectives and opportunities for future study. The techniques we have developed are easy to implement, readily reproducible, and incur much less computational cost (for hyperparameter tuning and model selection) than deep learning architectures. Thus, they are valuable tools, tested and verified to be effective for constructing comparative systems.

Comparative studies enabled by such systems will, in our view be indispensable in pursuing the higher goal of exploring and acquiring an understanding of how the two camps of methods differ, for instance in learning new representations of the original data[1]. As an example, we show that combining kernel models and DNNs improves over either individual model, suggesting that the two paradigms learn different yet complementary representations from the data. We believe that research in this line will offer deep insights, and broaden the theory and practice of designing alternative methods to both DNNs and kernel methods for large-scale learning.

---

[1]Note this inquiry would be informative only if both kernel methods and deep learning methods attain similar performance yet exploit different aspects of data.

The rest of the paper is organized as follows. We briefly review related work in section 2. In section 3, we give a brief account of [38]. We describe our approaches in section 4. In section 5, we report extensive experiments comparing DNNs and kernel methods on the problems in image and automatic speech recognition. We conclude and discuss future directions in section 6.

2   Related work

The computational complexity of kernel methods, such as support vector machines, depends quadratically on the number of training examples at training time and linearly on the number of training examples at the testing time. Hence, scaling up kernel methods has been a long-standing and actively studied problem. [8] summarizes several earlier efforts in this vein.

With clever implementation tricks such as computation caching (for example, keeping only a small portion of the very large kernel matrix inside the memory), earlier kernel machines can cope with hundreds of thousands of samples [46, 17]. [7] provides an excellent account of various design considerations.

To further reduce the dependency on the number of training samples, a more effective strategy is to actively select training samples [6]. An early version of this idea was reflected in the Sequential Minimal Optimization (SMO) algorithm [37]. With more sophistication, this technique was extended to enable training SVMs on 8 million samples [33]. Alternative approaches exploit the equivalence between SVMs and sparse greedy approximation and solve SVMs approximately with a smaller subset of examples called coresets[2] [49, 12]. Exploiting structures of the kernel matrix can scale kernel methods to 2 million to 50 million training samples [47]. Note that at the time of publication, none of the above-mentioned methods had been directly compared to DNNs.

Instead of reducing the number of training samples, we can reduce the dimensionality of kernel features. In theory, those features are infinite dimensional. But for any practical problem, the dimensionality is bounded above by the number of training samples. The main idea is then to directly use such features, after dimensionality reduction, to construct classifiers (i.e., solving the optimization problem of SVM in the primal space).

Thus far, approximating kernels with finite-dimensional features has been recognized as a promising way of scaling up kernel methods. The most relevant one to our paper is the early observation by Rahimi and Recht that inner products between features derived from random projections can be used to approximate translation-invariant kernels, a direct result of spectral analysis of positive functions [5, 43, 38]. Their follow-up work of using those random features — weighted random kitchen sink [39] — is a major inspiration to our work.

Since, there has been a growing interest in using random projections to approximate different kernels [25, 19, 31, 50]. For example, [15] studied how to use random features for online learning. We note that the amount of time for such classifiers to make a prediction depends linearly on the number of training samples. This could be a concern when the number of training samples is large.

In spite of these progresses, there have been only a few reported large-scale empirical studies of those techniques on challenging tasks from speech recognition and computer vision, on which DNNs have been highly effective. In the context of automatic speech recognition, examples of directly using kernel methods were reported [18, 9, 24]. However, the tasks were fairly small-scale (for instance, on the TIMIT dataset). Moreover, none of them explores kernel learning as a way of learning new representations. In contrast, one major aspect of our work is to use multiple kernel learning to arrive at new representations so as to reduce the gap between DNNs and kernel methods, cf. section 4.2.

3   Features from random projections

In what follows, we describe the basic idea we have built upon to scale up kernel methods. The technique is based on explicitly and efficiently constructing features — they are generated randomly — whose inner

---

[2]We also experimented those techniques though we were not able to identify significant empirical success.

products then approximate kernel functions. Once such features are constructed, they can be used as inputs by any classifier.

3.1 Generate features by random projections

Given a pair of data points $\boldsymbol{x}$ and $\boldsymbol{z}$, a positive definite kernel function $k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ defines an inner product between the images of the two data points under a (nonlinear) mapping $\boldsymbol{\phi}(\cdot) : \mathbb{R}^d \to \mathbb{R}^M$,

$$k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{z}) \tag{1}$$

where the dimensionality M of the resulting mapping $\boldsymbol{\phi}(\boldsymbol{x})$ can be infinite (in theory).

Kernel methods avoid inference in $\mathbb{R}^M$. Instead, they rely on the kernel matrix over the training samples. When M is far greater than N, the number of training samples, this trick provides a nice computational advantage. However, when N is exceedingly large, this complexity at the quadratic order of N becomes impractical.

Rahimi and Recht leverage a classical result in harmonic analysis and provide a fast way to approximate $k(\cdot, \cdot)$ with *finite*-dimensional features [38]:

Theorem 1. *(Bochner's theorem, adapted from [38]) A continuous kernel $k(\boldsymbol{x}, \boldsymbol{z}) = k(\boldsymbol{x} - \boldsymbol{z})$ is positive definite if and only if $k(\boldsymbol{\delta})$ is the Fourier transform of a non-negative measure.*

More specifically, for shift-invariant kernels such as Gaussian RBF and Laplacian kernels,

$$k^{\mathsf{rbf}} = e^{-\|\boldsymbol{x}-\boldsymbol{z}\|_2^2 / 2\sigma^2}, \quad k^{\mathsf{lap}} = e^{-\|\boldsymbol{x}-\boldsymbol{z}\|_1 / \sigma} \tag{2}$$

the theorem implies that the kernel function can be expanded with harmonic basis, namely

$$k(\boldsymbol{x} - \boldsymbol{z}) = \int_{R^d} p(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{z})} \, d\boldsymbol{\omega} = \mathbb{E}_{\boldsymbol{\omega}} \left[e^{j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}} e^{-j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{z}} \right] \tag{3}$$

where $p(\boldsymbol{\omega})$ is the density of a d-dimensional probability distribution. The expectation is computed on complex-valued functions of $\boldsymbol{x}$ and $\boldsymbol{z}$. For real-valued kernel functions, however, they can be simplified to the cosine and sine functions, see below.

For Gaussian RBF and Laplacian kernels, the corresponding densities are Gaussian and Cauchy distributions:

$$p^{\mathsf{rbf}}(\boldsymbol{\omega}) = N\left(0, \frac{1}{\sigma}\boldsymbol{I}\right), \quad p^{\mathsf{lap}}(\boldsymbol{\omega}) = \prod_d \frac{1}{\pi(1 + \sigma^2\omega_d^2)} \tag{4}$$

The harmonic decomposition suggests a sampling-based approach of approximating the kernel function. Concretely, we draw $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_D\}$ from the distribution $p(\boldsymbol{\omega})$ and use the sample mean to approximate

$$k(\boldsymbol{x}, \boldsymbol{z}) \approx \frac{1}{D} \sum_{i=1}^{D} \phi_{\boldsymbol{\omega}_i}(\boldsymbol{x}) \phi_{\boldsymbol{\omega}_i}(\boldsymbol{z}) = \hat{\boldsymbol{\phi}}(\boldsymbol{x})^{\mathrm{T}} \hat{\boldsymbol{\phi}}(\boldsymbol{z}) \tag{5}$$

The **random feature vector** $\hat{\boldsymbol{\phi}}$ is thus composed of scaled cosines of random projections

$$\phi_{\boldsymbol{\omega}_i}(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \cos(\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{x} + b_i) \tag{6}$$

where b_i is a random variable, uniformly sampled from $[0, 2\pi]$. Details on the convergence property of this approximation can be found in [38].

A key advantage of using approximate features over standard kernel methods is its scalability to large datasets. Learning with a representation $\hat{\boldsymbol{\phi}}(\cdot) \in \mathbb{R}^D$ is relatively efficient provided that D is far less than the number of training samples. For example, in our experiments (cf. section 5), we have 7 million to 10 million training samples, while $D = 50,000$ often leads to good performance.

4

3.2   Use random features in classifiers

Just as the standard kernel methods (SVMs or kernelized linear regression) can be seen as fitting data with linear models in kernel-induced feature spaces, we can plug in the random feature vector $\hat{\phi}(x)$ in just about any (linear) model. In this paper, we focus on using them to construct multinomial logistic regression. Specifically, our model is a special instance of the *weighted sum of random kitchen sinks* [39]

$$p(y = c | \boldsymbol{x}) = \frac{e^{\boldsymbol{w}_c^{\mathrm{T}} \hat{\boldsymbol{\phi}}(\boldsymbol{x})}}{\sum_c e^{\boldsymbol{w}_c^{\mathrm{T}} \hat{\boldsymbol{\phi}}(\boldsymbol{x})}} \tag{7}$$

where the label y can take any value from $\{1, 2, \ldots, C\}$. We use multinomial logistic regression mainly because it can deal with a large number of classes and provide posterior probability assignments, needed by the application task (i.e., the speech recognition systems, in order to combine with components such as language models).

4   Our Approaches

To scale up kernel methods, we address two challenges: (1) how to train large-scale models in the form of eq. (7); (2) how to learn optimal kernel functions adapted to the data. We tackle the former with a parallel optimization algorithm and the latter by extending the construction of random features initially proposed in [38].

4.1   Parallel optimization for large-scale kernel models

While random features and weighted sum of random kitchen sinks have been investigated before, there are few reported cases of scaling up to the problems commonly seen in automatic speech recognition and other domains. For example, in our empirical studies of acoustic modeling (cf. section 5.3), the number of classes is $C = 1000$ and we often use more than $D = 100,000$ random features to compose $\hat{\phi}(x)$. Thus, the linear model eq. (7) has a large number of parameters (about $C \times D = 10^8$).

We have developed two major strategies to overcome this challenge. First, we leverage the observation that fitting multinomial logistic regression is a convex optimization problem and adopt the method of stochastic averaged gradient (SAG) for its faster convergence, both theoretically and empirically, over stochastic gradient descent (SGD) [42]. Note that while SGD is widely applicable to both convex and non-convex optimization problems, SAG is specifically designed for convex optimization and thus well-suited to our learning setting.

Secondly, we leverage the property that random projections are just *random* – that is, given a D-dimensional $\hat{\phi}(x)$, any random subset of it is still random. Our idea is then to train a model on each subset of features *in parallel* and then assemble them together to form a large model.

Specifically, for large D (say $\geq 100,000$), we partition D into B blocks $\hat{\phi}_b(x)$ with each block having a size of D_0 (say $= 25,000$). Note that each block corresponds to a different set of random projections sampled from the density $p(\boldsymbol{\omega})$. We train B multinomial logistic regression models and obtain B sets of parameters for each class, ie., $\{\boldsymbol{w}_c^b, c = 1, 2, \ldots, C\}_{b=1}^B$. To assemble them, we combine in the spirit of *geometric mean* of the probabilities (or arithmetic mean of the log probabilities)

$$p(y = c | \boldsymbol{x}) \propto \exp\left(\frac{1}{B} \sum_{b=1}^{B} \hat{\boldsymbol{\phi}}_b(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{w}_c^b \right)$$

$$= \sqrt[B]{\prod_b \exp\left(\hat{\boldsymbol{\phi}}_b(\boldsymbol{x})_b^{\mathrm{T}} \boldsymbol{w}_c^b \right)} \tag{8}$$

5

Note that this assembled model can be seen as a D-dimensional model with parameters of $\{\frac{1}{B}\boldsymbol{w}_c^b, c = 1, 2, \ldots, C\}_{b=1}^B$.

We sketch the main argument for the validity of this parallel training procedure, leaving a rigorous proof for future work. The parameters of the weighted sum of random kitchen sink converges in $O(1/\sqrt{D})$ to the true risk minimizer [39]. Thus, for each model of size D_0, the pre-softmax activations (i.e., the logits) converge in $O(1/\sqrt{D_0})$. For B such models, the arithmetic mean of logits converge in $O(1/(\sqrt{B}\sqrt{D_0}))$ thus matching up the rate for a D-dimensional model. Our extensive empirical studies have supported our argument — in virtually all training settings, the assembled models cannot be improved further, attaining the optimum of the corresponding D-dimensional model.

4.2 Learning kernel features

Another advantage of using kernels is to sidestep the problem of feature engineering, i.e., how to select the best basis functions for a task at hand. Essentially, determining what kernel function to use implicitly specifies the basis functions. But then the question becomes: how to select the best kernel function?

One popular paradigm to address the latter problem is multiple kernel learning (MKL) [30, 1, 13, 27]. That is, starting from a collection of base kernels, the algorithm identifies the best subset of them and combines them together to best adapt to the training data, analogous to designing the best features according to the data.

In the following, we show how a few common MKL ideas can benefit from the previously described large-scale learning techniques (cf. section 3). While many MKL algorithms are formulated with kernel matrices (and thus are not easily scalable to large problems), we demonstrate how they can be efficiently implemented with the general recipe of random feature approximation. Among them, we show an interesting and novel result on combining kernels with Hadamard products, where the random feature approximation is especially computationally advantageous.

In our empirical studies (detailed in Supplementary Material), we will show that MKL improves methods using a single kernel, and eventually approaches the performance of deep neural networks. Thus, MKL presents an effective and computational tractable alternatives to DNNs, even for large-scale problems.

Additive Kernel Combination   Given a collection of base kernels $\{k_i(\cdot, \cdot), i = 1, 2, \ldots, L\}$, their non-negative combination

$$k(\cdot, \cdot) = \sum_i \alpha_i k_i(\cdot, \cdot) \tag{9}$$

is also a kernel function, provided $\alpha_i \geq 0$ for any i.

Suppose each kernel $k_i(\cdot, \cdot)$ is approximated with a D-dimensional random feature vector $\hat{\boldsymbol{\phi}}_i(\cdot)$, as in eq. (5). Then, given the linearity of the combination, the kernel function $k(\cdot, \cdot)$ can be approximated by

$$k(\cdot, \cdot) \approx \sum_i \alpha_i \hat{\boldsymbol{\phi}}_i(\cdot)^{\mathrm{T}} \hat{\boldsymbol{\phi}}_i(\cdot) = \hat{\boldsymbol{\phi}}(\cdot)^{\mathrm{T}} \hat{\boldsymbol{\phi}}(\cdot) \tag{10}$$

where $\hat{\boldsymbol{\phi}}(\cdot)$ is just the concatenation of the $\sqrt{\alpha_i}$-scaled $\hat{\boldsymbol{\phi}}_i(\cdot)$. Note that the dimensionality of $\hat{\boldsymbol{\phi}}(\cdot)$ would be $L \times D$.

There are several ways to exploit this approximation. The first way is to straightforwardly plug $\hat{\boldsymbol{\phi}}(\cdot)$ into the multinomial logistic regression eq. (7) and optimize over $L \times D$ features. The second way is more scalable. For each $\hat{\boldsymbol{\phi}}_i(\cdot)$, we learn an optimal model with parameters $\boldsymbol{w}_c^i$ for each class c. We then learn a set of combination coefficients α_i by optimizing the likelihood model

$$p(y = c|\boldsymbol{x}) \propto \exp\left(\sum_i \sqrt{\alpha_i} \hat{\boldsymbol{\phi}}_i(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{w}_c^i\right) \tag{11}$$

while holding the other parameters fixed. This is a convex optimization with (presumably) a small set of parameters.

6

While the first approach is more general, however, empirically, we do not observe a strong difference and have adopted the second approach for its scalability.

Multiplicative Kernel Combination   Kernels can also be multiplicatively combined from base kernels:

$$k(\cdot, \cdot) = \prod_{i=1}^{L} k_i(\cdot, \cdot) \tag{12}$$

Note that this is a highly nonlinear combination [13]. Unlike the additive combination, to approximate the multiplicative combination of kernels, there does not exist a simple form (such as concatenating) of composing with the approximate features of individual kernels. Nonetheless, We have proved the following theorem as a way to constructing the approximate features for $k(\cdot, \cdot)$ efficiently.

Theorem 2. *Suppose all $k_i(\cdot, \cdot)$ are translation-invariant kernels such that*

$$k_i(\boldsymbol{x} - \boldsymbol{z}) = \int_{R^d} p_i(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{z})} \, d\boldsymbol{\omega} \tag{13}$$

Then $k(\cdot, \cdot)$ is also translation-invariant such that

$$k(\boldsymbol{x} - \boldsymbol{z}) = \int_{R^d} p(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{z})} \, d\boldsymbol{\omega} \tag{14}$$

where the probability measure $p(\boldsymbol{\omega})$ is given by the convolution of all $p_i(\boldsymbol{\omega})$

$$p(\boldsymbol{\omega}) = p_1(\boldsymbol{\omega}) * p_2(\boldsymbol{\omega}) * \cdots * p_L(\boldsymbol{\omega}) \tag{15}$$

Moreover, let $\boldsymbol{\omega}_i \sim p_i(\boldsymbol{\omega})$ be a random variable drawn from the corresponding distribution, then

$$\boldsymbol{\omega} = \sum_i \boldsymbol{\omega}_i \sim p(\boldsymbol{\omega}) \tag{16}$$

Namely, to approximate $k(\cdot, \cdot)$, one needs only to draw random variables from each individual component kernel's corresponding density, and use the sum of those variables to compute random features.

The proof of the theorem is in the Suppl.. We note that $\boldsymbol{\omega}$ and $\boldsymbol{\omega}_i$ have the same dimensionality. Thus, the number of approximating features is independent of the number of kernels, leading to a computational advantage over additive combination.

Kernel composition   Kernels can also be composited. Specifically, if $k_2(\boldsymbol{x}, \boldsymbol{z})$ is a kernel function that depends on only the inner products of its arguments, then $k = k_2 \circ k_1$ is also a kernel function. A concrete example is when k_2 is the Gaussian RBF kernel and $k_1(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\phi}_1(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\phi}_1(\boldsymbol{z})$ for some mapping $\boldsymbol{\phi}_1(\cdot)$

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\{-\|\boldsymbol{\phi}_1(\boldsymbol{x}) - \boldsymbol{\phi}_1(\boldsymbol{z})\|_2^2/\sigma^2\}$$
$$= \exp\{-[k_1(\boldsymbol{x}, \boldsymbol{x}) + k_1(\boldsymbol{z}, \boldsymbol{z}) - 2k_1(\boldsymbol{x}, \boldsymbol{z})]/\sigma^2\}$$

If we approximate both k_1 and k_2 using the random feature approximation of eq. (5), the composition would be (graphically) equivalent to the following mapping,

$$\boldsymbol{x} \xrightarrow{\boldsymbol{\omega} \sim p_1(\boldsymbol{\omega})} \hat{\boldsymbol{\phi}}_1(\boldsymbol{x}) \xrightarrow{\boldsymbol{\omega} \sim p_2(\boldsymbol{\omega})} \hat{\boldsymbol{\phi}}_2(\hat{\boldsymbol{\phi}}_1(\boldsymbol{x})) \tag{17}$$

namely, a one-hidden-layer neural networks with the weight parameters in the layers being completely random. As before, the result of the composite mapping $\hat{\boldsymbol{\phi}}_2 \circ \hat{\boldsymbol{\phi}}_1$ can be used in any classifier as input features.

Table 1: Handwritten digit recognition error rates (%)

Model	kernel		DNN	
Augment training data	no	yes	no	yes
On validation	0.97	0.77	0.71	0.62
On test	1.09	0.85	0.69	0.77

We also generalize this operation to introduce a linear projection to reduce dimensionality, serving as information bottleneck: $\hat{\phi}_2 \circ \boldsymbol{P} \circ \hat{\phi}_1$. We experimented on two choices.

First, $\boldsymbol{P}$ performs PCA (using the sample covariance matrix) on $\hat{\phi}_1(\boldsymbol{x})$. Note that this implies $\boldsymbol{P} \circ \hat{\phi}_1$ is an approximate kernel PCA on the original feature space $\boldsymbol{x}$, using the kernel k_1. Secondly, $\boldsymbol{P}$ performs *supervised* dimensionality reduction. One simple choice is to implement Fisher discriminant analysis (FDA) on $\hat{\phi}_1(\boldsymbol{x})$, which is equivalent to kernel (FDA) on $\boldsymbol{x}$. In our experiments, we have used a different procedure in a similar spirit. Specifically, In particular, we first use $\hat{\phi}_1(\boldsymbol{x})$ as input features to build a multinomial logistic regression to predict its labels. We then perform PCA on the log-posterior probabilities. Our choice here is largely due to the consideration of re-using the computations as we often need to estimate the performance of $k_1(\cdot, \cdot)$ alone, thus the multinomial classifier built with $k_1(\cdot, \cdot)$ is readily usable.

5   EXPERIMENTAL RESULTS

We validate our approaches of scaling up kernel methods on challenging problems in computer vision and automatic speech recognition (ASR). We conduct extensive empirical studies comparing kernel methods to deep neural networks (DNNs), which perform well in computer vision, and are state-of-the-art in ASR. We show that kernel methods attain similarly competitive performance as DNNs – details in section 5.2 and 5.3 (as well as Supplementary Material).

What can we learn from two very different, yet equally competitive, learning models? We report our initial findings on this question (section 5.5). We show that kernel methods and DNNs learn different yet complementary representation of the data. As such, a direct application of this observation is to combine them to obtain better models than either independently.

5.1   General setup

For all kernel-based models, we tune only three hyperparameters: the bandwidths for Gaussian or Laplacian kernels, the number of random projections, and the step size of the (convex) optimization procedure (as adjusting it has a similar effect as early-stopping).

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, the learning rate, the rate decay, the momentum, regularization, etc. We also use unsupervised pre-training and tune hyperparameters for that phase too.

Details about tuning those hyperparameters are described in the Supplementary Material as they are often dataset-specific. In short, model selection for kernel models has significantly lower computational cost. We give concrete measures to support this observation in section 5.4.

5.2   Computer vision tasks

We experiment on two problems: handwritten digit recognition and object recognition.

Handwritten digit recognition   We extract a dataset MNIST-6.7M, from the dataset MNIST-8M [33]. MNIST-8M is a transformed version of the original MNIST dataset [32]. Concretely, we randomly select 50,000 out of 60,000 images from the MNIST's training set and extracted the corresponding samples (the

Table 2: Object recognition error (%)

Model	kernel		DNN	
Augment training data	no	yes	no	yes
On validation	43.2	41.4	42.9	43.2
On test	43.9	42.2	43.3	44.0

original as well as transformed/distorted ones) in MNIST-8M, resulting in a total of 6.75 million samples in total, as our training set. We use the remaining 10,000 images from the original training set as a validation set — we purposely avoid using any transformed versions of those 10,000 images as a validation dataset to avoid potential overfitting. We report test error rate on the standard 10,000 MNIST test set.

We also experimented with a data augmentation trick to increase the number of training samples *during training*. Whenever we encounter a training sample, we corrupt it with masking noise (randomly flipping 1 to 0 in the binary image). We crudely tune the mask-out rates, which are either 0.1, 0.2 or 0.3.

Table 1 compares the performance of the best single-kernel based classifier to that of the best DNN. The kernel classifier uses Gaussian kernel, and 150,000 random projections. The best DNN has 4 hidden layers with 1000, 2000, 2000, and 2000 hidden units respectively.

The difference between the kernel model and the DNN is small – about 16 (out of 10,000) misclassified images. Interestingly, on the test data, the kernel model benefits from the data augmentation trick while the DNN does not. Possibly, the DNN overfits to the validation dataset.

Object recognition For this task, we experiment on the database CIFAR-10 [28]. The dataset contains 50,000 training samples and 10,000 test samples. Each sample is a RGB image of 32×32 pixels in one of 10 object categories. We randomly picked 10,000 images from the training set for validation, keeping the remaining 40,000 images for training. We did not perform any preprocessing to the images as we want to relate our findings to previously published results which often do not preprocess data or do not report all specific details in preprocessing. We also experimented with an augmented version of the dataset, by injecting Gaussian noise to images during training.

Table 2 compares the performance of the best single-kernel based classifier to that of the best DNN. The kernel classifier uses Gaussian kernel, and 4,000,000 random projections. The best DNN has 3 hidden layers with 4000, 2000, 2000 hidden units respectively.

The best kernel model performs slightly better than the DNN. They both outperform previously reported DNN results on this dataset, whose error rates are between 44.4% and 48.5% [41, 28, 40].

Convolutional neural nets (CNNs) can significantly outperform DNNs on this dataset. However, we do not compare to CNNs as our kernel models (as well as DNNs) do not construct feature extractions with prior knowledge, while CNNs are designed especially for object recognition.

5.3 Automatic speech recognition (ASR)

Task and evaluation metric Deep neural nets (DNNs) have been very successfully applied to ASR. There, DNNs perform the task of acoustic modeling. Acoustic modeling is analogous to the conventional multi-class classification, that is, to learn a predictive model to assign phoneme context-dependent phoneme state labels y to short segments of speech, called frames, represented as acoustic feature vectors $\boldsymbol{x}$. Acoustic feature vectors are extracted from frames and their context windows (i.e., neighboring frames in temporal proximity).

Analogously, kernel-based multinomial logistic regression models, as described in section 3.2, are also used as acoustic models and compared to DNNs.

Acoustic models are often evaluated in conjunction with other components of ASR systems. In particular, speech recognition is inherently a sequence recognition problem. Thus, perplexity and classification accuracies — commonly used for conventional multi-class classification problems — provide only a proxy (and intermediate goals) to the sequence recognition error. To measure the latter, a full ASR pipeline is

Table 3: Best Token Error Rates on Test Set (%)

Model	Bengali	Cantonese
ibm	70.4	67.3
rbm	69.5	66.3
best kernel model	70.0	65.7

necessary where the posterior probabilities of the phoneme states are combined with the probabilities of the language models (of the interested linguistic units such as words) to yield the most probable sequence of those units. A best alignment with the ground-truth sequence is computed, yielding token error rates (TER).

Given the inherent complexity, in what follows, we summarize the empirical studies of applying both paradigms to the acoustic modeling task. We will report TER on two different languages. Details are presented in the Supplementary Material, including comparisons in terms of both perplexity and accuracy for different models. We begin by describing the datasets, followed by a brief description of various kernel and DNN models we have experimented with.

Datasets We use two datasets: the IARPA Babel Program Cantonese (IARPA-babel101-v0.4c) and Bengali (IARPA-babel103b-v0.4b) limited language packs. Each pack contains a 20-hour training, and a 20-hour test sets. We designate about 10% of the training data as a held-out set to be used for model selection and tuning.

We follow the same procedure to extract acoustic features from raw audio data as in the previous work using DNNs for ASR [26]. In particular, we have used IBM's proprietary system Attila which is adapted for the above-mentioned Babel language packs. The acoustic features are 360-dimensional real-valued dense vectors. There are 1000 (non-overlapping) phoneme context-dependent state labels for each language pack. For Cantonese, there are about 7.5 million data points for training, 0.9 million for held-out, and 7.2 million for test, and on Bengali, 7.7 million for training, 1.0 million for held-out and 7.1 million for test. For Bengali, the TER metric is the word-error-rate (WER) and for Cantonese, it is character-error-rate (CER).

Various of models being experimented IBM's Attila ASR system has a DNN acoustic model that contains five hidden-layers, each of which contains 1,024 units with logistic nonlinearities. We refer to this system as ibm. We have developed another version of DNN, following the original Restricted Boltzman Machine (RBM)-based training procedure for learning DNNs[22]. Specifically, the pre-training is unsupervised. We have trained DNNs with 1, 2, 3 and 4 hidden layers, and 500, 1000, and 2000 hidden units per layer (thus, 12 architectures per language). We refer to this system as rbm.

For kernel-based acoustic models, we used Gaussian RBF, Laplacian kernels or some forms of combinations. The only hyper-parameter there to tune is the kernel bandwidth, which ranges from 0.3 - 5 median of the pairwise distances in the data (Typically, the median works well.), the number of random projections ranging from 2,000 to 400,000 (though a stable performance is often observed at 25,000 or above). For training with very large number of features, we used the parallel training procedure, described in section 4.1. For optimization, we used the stochastic average gradient and tune the step size loosely from 4 values $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$.

Details about these systems are in Supplementary Material.

Results Table 3 reports the best performing models measured in TER. The RBM-trained DNN (rbm), which has 4 hidden layers and 2000 units in each layer, performs the best on Bengali. But our best kernel model, which uses Gaussian RBF kernel and $150,000 - 200,000$ random projections, performs the best on Cantonese. Both perform better than IBM's DNN. On Cantonese, the improvement of the kernel model over ibm is noticeably substantial (1.6% reduction in absolute).

Table 4: Combining best performing kernel and DNN models

Dataset	MNIST-6.7	CIFAR-10	Bengali	Cantonese
Best single	0.69	42.2	69.5	65.7
Combined	**0.61**	**40.3**	**69.1**	**64.9**

5.4 Computational efficiency

In contrast to DNNs, kernel models can be more efficiently developed. We illustrate this on two aspects: the computational cost of training a single model and the cost of model selection (i.e., hyperparameter tuning)

Cost of training a single model   While the amount of training time depends on several factors, including the volume and the dimensionality of the dataset, the choice of hyperparameters and their effect on convergence, implementation details, etc. We give a rough picture after controlling those extraneous factors as much as possible.

We implement both methods with highly optimized Matlab codes (comparable to our CUDA C implementation) and utilize a single GPU (NVidia Tesla K20m). The timing results reported below are obtained from training acoustic models on the Bengali language dataset.

For a kernel model with 25,000 random projections (25 million model parameters), convergence is reached in less than 20 epochs, with an average of 15 minutes per epoch. In contrast, a competitive deep model with four hidden layers of 2,000 hidden units (15 million parameters), if initialized with pretrained parameters, reaches convergence in roughly 10 epochs, with an average of 28 minutes per epoch. (The pretraining requires additional 12 hours.)

Thus, the training time for a single kernel model is about the same as that for a DNN. This holds for a range of datasets and configurations of hyperparameters.

Cost of model selection   The number of kernel models to be tuned, is significantly (at least one order in magnitude) less than DNNs. There are only two hyperparameters to search when selecting kernel models: the kernel bandwidth and the learning rate. Generally, the higher the number of random projections, the better the performance is.

For DNNs, the number of hyperparameters needed to be tuned is substantially more. As previously mentioned, in our experiments, we tuned those related to the network architecture and optimization procedure, for both pre-training and fine-tuning. As such, it is fairly common to select the best DNN among hundreds to thousands of them.

Combining both factors, kernel models are especially appealing when they are used to tackle new problem settings where there is only a weak knowledge about what the optimal hyperparameters are or what the proper ranges for those parameters are. To develop DNNs in this scenario, one would be forced to combinatorially adjust many knobs while kernel approaches are simple and straightforward.

5.5 Do kernel and deep models learn the same thing?

Given their matching performances, do kernel and DNN models learn the same knowledge from data?

We report in the following our initial findings. We first combine the best performing models from each paradigm — we use weighted sum of pre-softmax activations (i.e., logits). Table 4 summarizes those results across 4 tasks we have studied in the previous sections. In the row of "best single", blue color indicates the number is from a kernel model while red from a DNN. Clearly, combining the two improves either one independently.

These improvements suggest that despite being close in error rates, the two models are still different enough. We gain more intuitive understanding by visualizing what are being learnt by each model. To this end, we project each model's pre-softmax activations onto the 2-D plane with t-SNE.
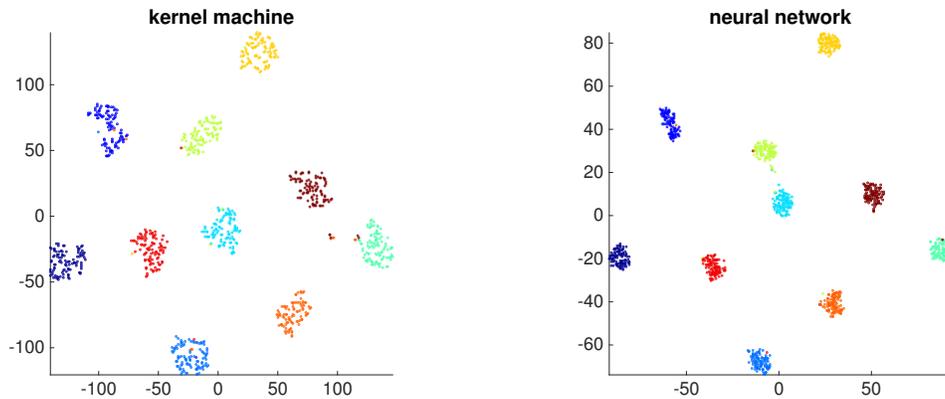
Figure 1: t-SNE embeddings of data representation learnt by kernel (Left) and by DNN (Right). The relative positioning of those well-separated clusters are different, suggesting that these two models learn to represent data in quite different ways (Kernel embedding is computed using DNN's embedding as initialization to avoid the randomness in t-SNE.)

Fig. 1 contrasts the embeddings for 1000 samples from MNIST-6.7M's test set. Each data point is a dot and the color encodes its label. Due to the low classification error rates, it is not surprising to find that there are 10 well separated clusters, one for each of the 10 digits. However, the relative positioning of those clusters is noticeably different between DNN and kernel. It is not clear the embeddings can be transformed into each other with linear transformations. This seems to suggest each method has its unique way of *nonlinearly* embedding data. Elucidating more precisely is our future research direction.

6   Conclusion

We propose techniques to scale up kernel methods to large learning problems that are commonly found in speech recognition and computer vision. We have shown that the performance of those large kernel models approaches or surpasses their deep neural networks counterparts, which have been regarded as the state-of-the-art. Future direction of our research include understanding the difference of these two camps of methods, for instance, in learning new representations of data.

Acknowledgement

References

[1] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proc. of the 21th Intl. Conf. on Mach. Learn. (ICML)*, 2004.

[2] Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors. *Advances in Neural Information Processing Systems 22*, 2009.

[3] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, January 2009.

[4] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: a review and new perspectives. *IEEE Trans. on Pattern Anal. & Mach. Intell.*, 35(8):1798–1828, 2013.

[5] Christian Berg, Jens Peter Reus Christensen, and Paul Ressel. *Harmonic Analysis on Semigroups*. Springer, 1984.

[6] Léon Bottou. Personal communication, 2014.

[7] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. In Bottou et al. [8].

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

[9] Chih-Chieh Cheng and B. Kingsbury. Arccosine kernels: Acoustic modeling with infinite neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5200–5203, 2011.

[10] K. Cho, A. Ilin, and T. Raiko. Improved learning of gaussian-bernoulli restricted boltzmann machines. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2011)*, pages 10–17, 2011.

[11] Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In Bengio et al. [2], pages 342–350.

[12] Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Trans. Algorithms*, 6(4):63:1–63:30, 2010.

[13] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Learning non-linear combinations of kernels. In Bengio et al. [2], pages 396–404.

[14] Corinna Cortes, Neil Lawrence, and Kilian Weinberger, editors. *Advances in Neural Information Processing Systems 27*, 2014.

[15] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In Cortes et al. [14].

[16] Sanjoy Dasgupta and David McAllester, editors. *Proc. of the 30th Int. Conf. on Mach. Learn. (ICML)*, volume 28 of *JMLR W & CP*, 2013.

[17] Dennis DeCoste and Bernhard Schölkopf. Training invariant support vector machines. *Mach. Learn.*, 46:161–190, 2002.

[18] Li Deng, Gökhan Tür, Xiaodong He, and Dilek Z. Hakkani-Tür. Use of kernel deep convex networks and end-to-end learning for spoken language understanding. In *2012 IEEE Spoken Language Technology Workshop (SLT), Miami, FL, USA, December 2-5, 2012*, pages 210–215, 2012.

[19] Raffay Hamid, Ying Xiao, Alex Gittens, and Dennis DeCoste. Compact random feature maps. In Dasgupta and McAllester [16], pages 19 – 27.

[20] G. E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8): 1771–1800, 2002.

[21] Geoffrey Hinton, Li Deng, Dong Yu, George E 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. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[22] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neual Comp.*, 18(7):1527–1554, 2006.

[23] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, July 2012. URL http://arxiv.org/abs/1207.0580.

[24] Po-Sen Huang, Haim Avron, Tara N Sainath, Vikas Sindhwani, and Bhuvana Ramabhadran. Kernel methods match deep neural networks on TIMIT. In *Proc. of the 2014 IEEE Intl. Conf. on Acou., Speech and Sig. Proc. (ICASSP)*, volume 1, page 6, 2014.

[25] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In *Proc. of the 29th Intl. Conf. on Mach. Learn. (ICML)*, 2012.

[26] 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 Phlip C. Woodland. A high-performance Cantonese keyword search system. In *Proc. of the 2013 IEEE Intl. Conf. on Acou., Speech and Sig. Proc. (ICASSP)*, pages 8277–8281, 2013.

[27] Marius Kloft, Ulf Brefeld, Sören Sonnenburg, and Alexander Zien. l_p-norm multiple kernel learning. *Journal of Machine Learning Research*, 12:953–997, 2011.

[28] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Pereira et al. [36], pages 1097–1105.

[30] Gert R. G. Lanckriet, Nello Cristianini, Peter L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[31] Quoc Viet Le, Tamás Sarlós, and Alexander Johannes Smola. Fastfood: Approximating kernel expansions in loglinear time. In Dasgupta and McAllester [16].

[32] Y. LeCun and C. Cortes. The mnist database of handwritten digits, 1998.

[33] Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. In Bottou et al. [8].

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

[35] R. Neal. Priors for infinite networks. Technical Report CRG-TR-94-1, Dept. of Computer Science, University of Toronto, 1994.

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

[37] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[38] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, pages 1177–1184, 2007.

[39] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21*, pages 1313–1320, 2008.

[40] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *International Conference on Artificial Intelligence and Statistics*, pages 924–932, 2012.

[41] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840, 2011.

[42] Nicolas L. Roux, Mark Schmidt, and Francis R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In Pereira et al. [36], pages 2663–2671.

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

[44] Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 24–29, 2011.

[45] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Proc. of Interspeech*, pages 437–440, 2011.

[46] Alex Smola. Personal communication, 2014.

[47] Sören Sonnenburg and Vojtech Franc. COFFIN: A computational framework for linear SVMs. In *Proc. of the 27th Intl. Conf. on Mach. Learn. (ICML)*, pages 999–1006, Haifa, Israel, 2010. URL http://www.icml2010.org/papers/280.pdf.

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Cortes et al. [14].

[49] 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. URL http://www.jmlr.org/papers/v6/tsang05a.html.

[50] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. on Pattern Anal. & Mach. Intell.*, 34(3):480–492, 2012.

[51] C. K. I. Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems 19*, pages 599–621, 1996.

A   Proof of the Theorem 2

Theorem 2. *Suppose all $k_i(\cdot, \cdot)$ are translation-invariant kernels such that*

$$k_i(\boldsymbol{x} - \boldsymbol{z}) = \int_{R^d} p_i(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{z})} \, d\boldsymbol{\omega}$$

Then $k(\cdot, \cdot) = \prod_{i=1}^{L} k_i(\cdot, \cdot)$ is also translation-invariant such that

$$k(\boldsymbol{x} - \boldsymbol{z}) = \int_{R^d} p(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{z})} \, d\boldsymbol{\omega}$$

where the probability measure $p(\boldsymbol{\omega})$ is given by the convolution of $p_i(\boldsymbol{\omega})s$

$$p(\boldsymbol{\omega}) = p_1(\boldsymbol{\omega}) * p_2(\boldsymbol{\omega}) * \cdots * p_L(\boldsymbol{\omega})$$

Moreover, let $\boldsymbol{\omega}_i \sim p_i(\boldsymbol{\omega})$ be a random variable drawn from the corresponding distribution, then

$$\boldsymbol{\omega} = \sum_i \boldsymbol{\omega}_i \sim p(\boldsymbol{\omega})$$

Namely, to approximate $k(\cdot, \cdot)$, one needs only to draw random variables from each individual component kernel's corresponding density, and use the sum of those variables to compute random features.

Proof. Denote $\mathbf{\Delta} = \boldsymbol{x} - \boldsymbol{z}$.

For translation-invariant kernel, we have

$$k_i(\boldsymbol{x}, \boldsymbol{z}) = k_i(\mathbf{\Delta}) \quad = \quad \int_{\boldsymbol{\omega}_i} p_i(\boldsymbol{\omega}_i) e^{j\boldsymbol{\omega}_i^{\mathrm{T}} \mathbf{\Delta}} \mathrm{d}\boldsymbol{\omega}_i$$

The product of the kernels is,

$$k(\cdot, \cdot) = \prod_{i=1}^{L} k_i(\cdot, \cdot) = \prod_{i=1}^{L} k_i(\mathbf{\Delta}) = k(\mathbf{\Delta}),$$

which is also translation-invariant.

$$
\begin{aligned}
k(\mathbf{\Delta}) = \quad & \prod_{i=1}^{L} \int_{\boldsymbol{\omega}_i} p_i(\boldsymbol{\omega}_i) e^{j\boldsymbol{\omega}_i^{\mathrm{T}} \mathbf{\Delta}} \mathrm{d}\boldsymbol{\omega}_i \\
= \quad & \int_{\boldsymbol{\omega}_1 .. \boldsymbol{\omega}_L} p_1(\boldsymbol{\omega}_1) \ldots p_L(\boldsymbol{\omega}_L) e^{j(\sum_i^L \boldsymbol{\omega}_i^{\mathrm{T}}) \mathbf{\Delta}} \mathrm{d}\boldsymbol{\omega}_1 \ldots \mathrm{d}\boldsymbol{\omega}_L \\
\overset{\tilde{\omega} = \sum_i^L \boldsymbol{\omega}_i}{=} \quad & \int_{\tilde{\boldsymbol{\omega}}} \left[\int_{\boldsymbol{\omega}_1 \ldots \boldsymbol{\omega}_{L-1}} p_1(\boldsymbol{\omega}_1) \ldots p_{L-1}(\boldsymbol{\omega}_{L-1}) \right. \\
& \left. p_L(\tilde{\boldsymbol{\omega}} - \sum_i^{L-1} \boldsymbol{\omega}_i) \mathrm{d}\boldsymbol{\omega}_1 \ldots \mathrm{d}\boldsymbol{\omega}_{L-1} \right] e^{j\tilde{\boldsymbol{\omega}}^{\mathrm{T}} \mathbf{\Delta}} \mathrm{d}\tilde{\boldsymbol{\omega}} \\
= \quad & \int p_{\tilde{\omega}}(\tilde{\omega}) e^{j\tilde{\boldsymbol{\omega}}^{\mathrm{T}} \mathbf{\Delta}} \mathrm{d}\tilde{\boldsymbol{\omega}} \\
= \quad & \int p_{\tilde{\omega}}(\tilde{\omega}) e^{j\tilde{\boldsymbol{\omega}}^{\mathrm{T}} (\boldsymbol{x}-\boldsymbol{z})} \mathrm{d}\tilde{\omega} \\
= \quad & \mathbb{E}_{\tilde{\boldsymbol{\omega}}} \left[\phi_{\tilde{\omega}}(\boldsymbol{x}) \phi_{\tilde{\omega}}(\boldsymbol{z})^* \right]
\end{aligned}
$$

We have used the fact (due to convolution theorem) that

$$
\begin{aligned}
& \int_{\boldsymbol{\omega}_1 \ldots \boldsymbol{\omega}_{L-1}} p_1(\boldsymbol{\omega}_1) \ldots p_{L-1}(\boldsymbol{\omega}_{L-1}) \\
& \quad p_L(\tilde{\boldsymbol{\omega}} - \sum_i^{L-1} \boldsymbol{\omega}_i) \mathrm{d}\boldsymbol{\omega}_1 \ldots \mathrm{d}\boldsymbol{\omega}_{L-1} \\
= \quad & p_1(\boldsymbol{\omega}_1) * p_2(\boldsymbol{\omega}_2) * \cdots * p_L(\boldsymbol{\omega}_L) = p_{\tilde{\omega}}(\tilde{\boldsymbol{\omega}})
\end{aligned}
$$

It means we have found a new distribution $p_{\tilde{\omega}}(\tilde{\boldsymbol{\omega}})$ as the random projection generating distribution for the new kernel

$$k(\cdot, \cdot) = \prod_{i=1}^{L} k_i(\cdot, \cdot).$$

From the definition of $\tilde{\omega}$, in order to sample from $p_{\tilde{\omega}}(\tilde{\boldsymbol{\omega}})$, we can simply use the sum of independent samples from $p_i(\boldsymbol{\omega}_i)$. $\qquad \square$

B   Detailed experimental Results

B.1   Image recognition

We first provide details on our empirical studies on challenging problems in image recognition.

B.1.1 Handwritten digit recognition

Dataset and Preprocessing   Dataset is described in section 5.2 of the main text. We scale the input between $[0, 1)$ by dividing 256.

Kernel   We use Gaussian RBF and Laplacian kernels with kernel bandwidth selected from $\{1, 1.5, 2, 2.5, 3\} \times$ median of the pairwise distance in data. We select the learning rate from $\{5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}\}$. The random feature dimension we have used is 150,000. Performance with different dimension is shown in Table 5.

DNN   We trained DNNs with $1, 2, 3,$ or 4 hidden layers, with 1000, 2000, 2000 and 2000 hidden units respectively. We firstly pre-trained 1 Gaussian-Bernoulli and 3 consecutive Bernoulli restricted Boltzmann machines (RBMs), all using Stochastic Gradient Descent (SGD) with Contrastive Divergence (CD-1) algorithm [20].

We select learning rate from $\{10^{-1}, 1.5 \times 10^{-1}\}$, momentum from $\{0.5, 0.9\}$ and set L2 regularization to 2×10^{-4} for 2 epochs of pretraining. In finetuning, we tune SGD with learning rate from $\{5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}, 5 \times 10^{-1}\}$, momentum from $\{0.7, 0.9\}$. We decrease the learning rate by a factor of 0.99 for every epoch and set mini-batch size to 100, L2 regularization to 0. We use early-stopping to control overfitting. When trained with data augmentation, we use smaller learning rate and run for more epochs.

Data Augmentation   We use mask-out noise with ratio $\{0.1, 0.2, 0.3\}$ for both kernel methods and DNN.

Results   Table 6 compares the performance of kernel methods to deep neural nets of different architectures. The best result of DNN is a 4-hidden-layer neural network. Deep nets generally have slightly smaller test errors. Kernel models benefit more from data augmentation and achieve similar error rates.

Table 5: Kernel Methods on MNIST-6.7M (error rates %)

Kernel type	Data aug.	10K	50K	100K	150K
Gaussian	No	1.45/1.42	1.03/1.25	0.98/1.12	0.97/1.09
Laplacian	No	1.93/1.93	1.21/1.34	1.16/1.17	1.10/1.13
Gaussian	Yes	-	0.83/1.03	0.79/0.92	0.77/0.85

Table 6: DNN on MNIST-6.7M (error rates %)

Model	Original		Augmented	
	Validation	Test	Validation	Test
kernel	0.97	1.09	0.77	0.85
4 hidden	0.71	**0.69**	0.64	0.80
3 hidden	0.78	0.73	0.74	0.77
2 hidden	0.76	0.71	0.64	0.79
1 hidden	0.84	0.95	0.79	**0.76**

PCA Embedding   In addition to t-SNE visualization, we also project each model's pre-softmax activation onto the first two principle components given by PCA. Fig. 2 contrasts the PCA embeddings for 1000 samples from MNIST-6.7M's test set. Neural network seems to give more spread out embeddings compared to kernel machine. The most noticeable class is digit 3 (light blue in lower left corner) and digit 6 (green on the right).

B.1.2 Object Recognition

Dataset and Preprocessing   Dataset is described in section 5.2 of the main text. We scale the input between $[0, 1)$ by dividing 256. No other preprocessing is used because we would like to relate to previously reported results on DNNs on this data where preprocessing is not applied.
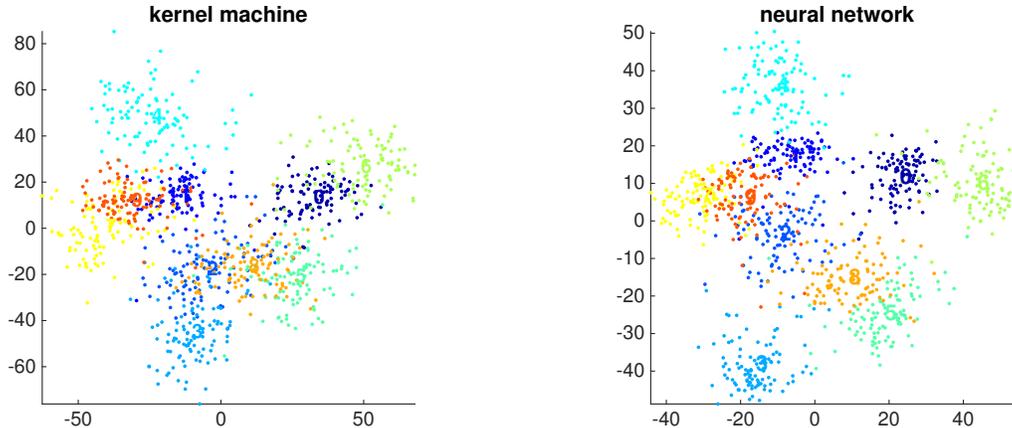
Figure 2: Embeddings of data by kernel (Left) and by DNN (Rights).

Kernel Gaussian kernel is used. We achieve the best performance by using 4,000,000 random features. This is done by training 200K single models in parallel and then combine. Table 7 shows the performance of kernel with respect to different number of random features. Similarly, we select bandwidth from $\{0.5, 1, 1.5, 2, 3\} \times$ median distance and learning rate from $\{5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}\}$.

DNN We trained DNNs with 1 to 4 hidden layers, with 2000 hidden units per layer. In pretraining, we use a Gaussian RBM for the input layer and three Bernoulli RBMs for intermediate hidden layers using CD-1 algorithm. (We adopt the parameterization of GRBM in [10], which shows better performance.)

For Gaussian RBM, we tune learning rate from $\{5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}\}$, momentum from $\{0.2, 0.5, 0.9\}^3$, and L2 regularization from $\{2 \times 10^{-5}, 2 \times 10^{-4}\}$. For Bernoulli RBM, we tune learning rate from $\{10^{-2}, 2.5 \times 10^{-2}, 5 \times 10^{-2}\}$, momentum from $\{0.2, 0.5, 0.9\}$, and L2 regularization is $\{2 \times 10^{-5}\}$. In finetuning, we tune SGD with learning rate from $\{4 \times 10^{-2}, 8 \times 10^{-2}, 1.2 \times 10^{-1}\}$, momentum from $\{0.2, 0.5, 0.9\}$, decrease the learning rate by a factor of 0.9 for every 20 or 50 epochs and set mini-batch size to 50, L2 regularization to 0. We use early-stopping to control overfitting. When trained with data augmentation, we use smaller learning rate and run for more epochs.

We used constant learning rate throughout 30 epochs and update momentum after 5 epochs. In finetuning stage, we used stochastic gradient descent with 0.9 momentum, fixed learning rate schedule, decreasing the learning rate by 10 after 50 epochs. The optimal model is selected according to the classification accuracy on validation dataset. We trained DNNs with 1, 2, 3 and 4 hidden layers with 2000 hiddenunits per layer. Overfitting was observed after we increased model from 3 hidden layers to 4 hidden layers.

Data Augmentation We apply additive Gaussian noise with standard deviation $\{0.1, 0.2, 0.3\}$ on raw pixel for both kernel methods and DNN.

Results Table 8 contrasts results of kernel models and DNN. We observe overfitting for 4 hidden layer neural network, and achieve best results in a 3 hidden layer architecture. Deeper models start to overfit and give worse validation and test performance. Kernel models achieve the best error when data augmentation is used.

B.2 Automatic speech recognition

In what follows, we provide comprehensive details on our empirical studies on challenging problems in automatic speech recognition.

B.2.1 Tasks, datasets and evaluation metrics

Task We have selected the task of acoustic modeling, a crucial component in automatic speech recognition. In its most basic form, acoustic modeling is analogous to the conventional multi-class classification, that is, to learn

---

[3]Momentum can be increased to another choice from the three after 5 epochs.

Table 7: Kernel Methods on CIFAR-10 (error rate%)

Gaussian r.f.	Original		Augmented	
	Validation	Test	Validation	Test
200K	43.74	44.48	42.15	43.13
1M	43.43	44.08	41.62	42.38
2M	43.26	44.04	41.47	42.26
4M	43.22	**43.93**	41.36	**42.23**

Table 8: DNN on CIFAR-10 (error rate %)

Model	Original		Augmented	
	Validation	Test	Validation	Test
kernel	43.22	43.93	41.36	**42.23**
4 hidden	43.21	43.74	43.00	43.38
3 hidden	42.89	**43.29**	42.93	43.35
2 hidden	43.30	43.76	43.80	44.81
1 hidden	48.40	48.94	47.28	47.79

a predictive model to assign phoneme context-dependent state labels to short segments of speech, called frames. While speech signals are highly non-stationary and context-sensitive, acoustic modeling addresses this issue by using acoustic features extracted from context windows (i.e., neighboring frames in temporal proximity) to capture the transient characteristics of the signals.

Data characteristics To this end, we use two datasets: the IARPA Babel Program Cantonese (IARPA-babel101-v0.4c) and Bengali (IARPA-babel103b-v0.4b) limited language packs. Each pack contains a 20-hour training, and a 20-hour test sets. We designate about 10% of the training data as a held-out set to be used for model selection and tuning (i.e., tuning hyperparameters etc). The training, held-out, and test sets contain different speakers. The acoustic data is very challenging as 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. Compared to the more familiar TIMIT corpus, which contains about 4 hours of training data, the Babel data is substantially more challenging because the TIMIT data is read speech recorded in a well-controlled, quiet studio environment.

As is standard on previous work using DNNs for speech recognition, the data is preprocessed using Gaussian mixture models to give alignments between phoneme state labels and 10-millisecond-frames of speech [26]. The acoustic features are 360-dimensional real-valued dense vectors. There are 1000 (non-overlapping) phoneme context-dependent state labels for each language pack. For Cantonese, there are about 7.5 million data points for training, 0.9 million for held-out, and 7.2 million for test, and on Bengali, 7.7 million for training, 1.0 million for held-out and 7.1 million for test.

Evaluation metrics We will be reporting 3 evaluation metrics, typically found in mainstream speech recognition research.

Perplexity Given a set of examples, $\{(\boldsymbol{x}_i, y_i), i = 1 \ldots m\}$, the perplexity is defined as

$$\mathsf{perp} = \exp\left\{ -\frac{1}{m} \sum_{i=1}^{m} \log p(y_i | \boldsymbol{x}_i) \right\}$$

The perplexity measure is lower bound by 1 when all predictions are perfect: $p(y_i | \boldsymbol{x}_i) = 1$ for all samples. With random guessing $p(y_i | \boldsymbol{x}_i) = 1/C$, where C is the number of classes, the perplexity attains C.

We use the perplexity measure on the held-out for model selection and tuning. This is because the perplexity is often found to be correlated with the next two performance measures.

Accuracy The classification accuracy is defined as

$$\mathsf{acc} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1} \left[y_i = \arg\max_{y \in 1, 2, \ldots, C} p(y|\boldsymbol{x}_i) \right]$$

Token Error Rate (TER) Speech recognition is inherently a sequence recognition problem. Thus, perp and acc provide only proxy (and intermediate goals) to the sequence recognition error. To measure the latter, a full automatic speech recognition pipeline is necessary where the posterior probabilities of the phoneme labels $p(y|\boldsymbol{x})$ are combined with the probabilities of the language models (of the interested linguistic units such as words) to yield the most probable sequence of those units. A best alignment with the ground-truth sequence is computed, yielding token error rates. For Bengali, the token error rate is the word-error-rate (WER) and for Cantonese, it is character-error-rate (CER).

Because it entails performing speech recognition, obtaining TER is computationally costly thus it is rarely used for model selection and tuning. Note also that the token error rates obtained on the Babel tasks are much higher than those are reported for other conversational speech tasks such as Switchboard or Broadcast News. This is because we have much less training data for Babel than for the other tasks. This *low-resource* setting is an important one in the speech processing area, given that there are a large number of languages in the world for which speech and language models do not currently exist.

B.2.2 Deep neural nets acoustic models

There are many variants of DNNs techniques. We have decided to choose two flavors that are very different in learning from data, in order to have a broader comparison. In either case, our model tuning is extensive.

IBM's DNN We have used IBM's proprietary system Attila for the conventional speech recognition that is adapted for the above-mentioned Babel task. A detailed description appears in [26]. Attila contains a state-of-the-art acoustic model provided by IBM. It also powers our full ASR pipeline in order to compute token error rate (TER). We have also used it to convert raw speech signals into acoustic features. Concretely, the features at a frame is a 40-dimensional speaker-adapted representation that has previously been shown to work well with DNN acoustic models [26]. Features at 8 neighboring contextual frames are concatenated, yield 360-dimensional features. We have used the same features for our kernel methods.

IBM's DNN acoustic model contains five hidden-layers, each of which contains 1,024 units with logistic nonlinearities. The output is a softmax nonlinearity with 1,000 targets that correspond to quinphone context-dependent HMM states clustered using decision trees. All layers in the DNN are fully connected. The training of the DNN occurs in two stages. First, a greedy layer-wise discriminative pretraining [45] to set the weights for each layer in a reasonable range. Then, the cross-entropy criterion is minimized with respect to all parameters in the network, using stochastic gradient descent with a mini-batch size of 250 samples, without momentum, and with annealing the learning rate based on the reduction in cross-entropy loss on a held-out set.

RBM-DNN We have designed another version of DNN, following the original Restricted Boltzman Machine (RBM)-based training procedure for learning DNNs[22]. Specifically, the pre-training is unsupervised. We have trained DNNs with 1, 2, 3 and 4 hidden layers, and 500, 1000, and 2000 hidden units per layer (thus, totally 12 architectures per language).

The first hidden layer is a Gaussian RBM and the upper layers are Binary-Bernoulli RBM. In pre-training, we use 5 epochs of SGD with Contrastive Divergence (CD-1) algorithm on all training data. We tuned 3 hyper-parameters, which are learning rate, momentum, and the strength for an ℓ_2 regularizer. For fine-tuning, we used error back-propagation. We tuned the initial learning rate, learning rate decay, momentum and the strength for another ℓ_2 regularizer. The fine-tuning usually converges in 10 epochs.

B.2.3 Kernel acoustic models

The development of kernel acoustic models does not require combinatory searching over many factors. We experimented only two types of kernels: Gaussian RBF and Laplacian kernels. The only hyper-parameter there to tune is the kernel bandwidth, which ranges from 0.3 - 5 median of the pairwise distances in the data. (Typically, the median works well.)

Table 9: Best perplexity and accuracy by different models (see texts for description of different models)

Model	Bengali		Cantonese	
	perp	acc (%)	perp	acc (%)
ibm	3.4/3.5	71.5/71.2	6.8/6.16	56.8/58.5
rbm	3.3/3.4	72.1/71.6	6.2/5.7	58.3/59.3
1-k	3.7/3.8	70.1/69.7	6.8/6.2	57.0/58.3
a-2-k	3.6/3.8	70.3/70.0	6.7/6.0	57.1/58.5
m-2-k	3.7/3.8	70.3/69.9	6.7/6.1	57.1/58.4
c-2-k	3.5/3.6	71.0/70.4	6.5/5.7	57.3/58.8

Table 10: Performance of rbm acoustic models

(h, L)	Bengali		Cantonese	
	perp	acc (%)	perp	acc (%)
(1, 500)	3.9/3.9	69.2/69.3	7.1/6.4	55.8/57.4
(2, 500)	3.5/3.6	70.9/70.7	6.6/6.1	57.3/58.4
(3, 500)	3.5/3.5	71.2/70.9	6.4/5.9	57.7/58.6
(4, 500)	3.4/3.5	71.2/70.8	6.4/5.9	57.5/58.7
(1, 1000)	3.7/3.7	70.1/70.1	6.8/6.2	56.4/58.0
(2, 1000)	3.4/3.4	71.6/71.4	6.3/5.8	58.2/59.0
(3, 1000)	3.4/3.5	71.7/71.3	6.3/5.7	58.0/59.2
(4, 1000)	3.3/3.5	71.8/71.4	6.6/5.8	57.1/58.6
(1, 2000)	3.6/3.7	70.5/70.3	6.7/6.1	56.9/58.1
(2, 2000)	3.4/3.4	71.8/71.4	6.2/5.7	58.3/59.3
(3, 2000)	3.4/3.5	71.5/71.2	6.2/5.6	57.8/59.1
(4, 2000)	3.3/3.4	72.1/71.6	6.4/5.8	57.8/59.1

The random feature dimensions we have used ranging from 2,000 to 400,000, though a stable performance is often observed at 25,000 or above. For training with very large number of features, we used the parallel training procedure, described in section 4.1 of the main text.

All kernel acoustic models are multi-nomial logistic regression, thus optimized by convex optimization. As mentioned in section 4.1 of the main text, we use Stochastic Average Gradient (SAG), which efficiently leverages the convexity property. We do tune the step size, selected from a loose range of 4 values $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$.

For additive and multiplicative kernel combinations, we combine only two, one Gaussian and the other Laplacian. For additive combinations, we first train two models, one for each kernel. The combining coefficient α is selected from $0.1, 0.2, \ldots, 0.9$. For composite kernels, we compose Gaussian with Laplacian. We perform a supervised dimensionality reduction, as described in section 4.2 of the main text. The reduced dimensionality is chosen from 50, 100, or 360. The first kernel's bandwidth is greedily selected to be optimal as a single-kernel acoustic model. The other kernel's bandwidth is selected after composing the features.

B.2.4 Results on Perplexity and Accuracy

Table 9 concisely contrasts the best perplexity and accuracy attained by various systems: ibm (IBM's DNN), rbm (RBM-trained DNN), 1-k (single kernel based model), a-2-k (additive combination of two kernels), m-2-k (multiplicative combination of two kernels) and c-2-k (composite of two kernels). We report the metrics on both the held-out and the test datasets (the numbers are separated by a /). In general, the metrics are consistent across both datasets and perp correlates with ACC reasonably well.

On Bengali, across all systems, the rbm attains the best perplexity (red colored numbers in the table), outperforming ibm and suggesting that unsupervised pre-training is advantageous. The best performing kernel model is c-2-k, trailing slightly behind rbm and ibm.

Similarly, on Cantonese, rbm performs the best, followed by c-2-k, both outperforming ibm. As an illustrate example, we show in Table 10 the performance of rbm on Bengali, under different types of architectures (h is the

Table 11: Performance of single Laplacian kernel

Dim	Bengali		Cantonese	
	perp	acc (%)	perp	acc (%)
2k	4.4/4.4	66.5/66.8	8.5/7.4	52.7/54.8
5k	4.1/4.2	67.8/67.8	7.8/7.0	53.9/56.0
10k	4.0/4.1	68.4/68.3	7.5/6.7	54.9/56.6
25k	3.8/3.9	69.2/69.0	7.1/6.4	55.9/57.3
50k	3.8/3.9	69.7/69.4	6.9/6.2	56.5/57.9
100k	3.7/3.8	70.0/69.6	6.8/6.2	56.8/58.2
200k	3.7/3.8	70.1/69.7	6.8/6.2	57.0/58.3

Table 12: Best Token Error Rates on Test Set (%)

Model	Bengali	Cantonese
ibm	70.4	67.3
rbm	69.5	66.3
1-k	70.0	65.7
a-2-k	73	68.8
m-2-k	72.8	69.1
c-2-k	71.2	68.1

number of hidden layers and L the number of hidden units). Meanwhile, in Table 11, we show the performance of single Laplacian kernel acoustic model with different number of random features.

Contrasting these two tables, it is interesting to observe that kernel models use far more parameters than DNNs to achieve similar perplexity and accuracy. For instance, for a rbm with $(h = 1, L = 500)$ with a perplexity of 3.9, the number of parameters is $360 \times 500 + 500 \times 1000 = 0.68$ million. This is a fraction of a comparable kernel model with Dim=10k$\times$1000 $= 10$ million parameters. In some way, this ratio provides an intuitive measure of the price being convenient, i.e., using random features in kernel models instead of adapting features to the data as in DNNs.

B.2.5 Results on Token Error Rates

Table 12 reports the performance of various models measured in TER, another important and more relevant metric to speech recognition errors.

Note that the RBM-trained DNN (rbm) performs the best on Bengali, but our best kernel model performs the best on Cantonese. Both perform better than IBM's DNN. On Cantonese, the improvement of our kernel model over ibm is noticeably large (1.6% reduction in absolute).

Table 13 highlights several interesting comparison between rbm and kernel models. Concretely, it seems that DNNs need to be big enough in order to reach the proximity of its best TER. On the other end, the kernel models' performance plateaus rather quickly. This is the opposite to what we have observed when we compare two methods using perplexity and accuracy.

One possible explanation is that for different models, the relationship between perplexity and TER are different. This is certainly plausible, given TER is highly complex to compute and two different models might explore parameter spaces very differently.

Another possible explanation is that these two different models learn different representations that bias either toward perplexity or toward TER. Table 14 suggests that this might indeed be true: as we combine two different models, we see handsome gains in performance over each individual one's.

B.2.6 DNN and kernels learn complementary representations

Inspired by what we have observed in the previous section, we set out to analyze in what way the representations learnt by two different models might be complementary. We have obtained preliminary results.

We took a learned DNN (we used the best perform one in terms of TER) and computed its pre-activation to the output layer, which is a linear transformation of the last hidden layer's outputs. For the best performing single-

Table 13: Detailed Comparison on TER for Bengali

Model	Arch.	TER (%)
rbm	$h = 2, L = 1000$	73.1
rbm	$h = 3, L = 1000$	72.7
rbm	$h = 2, L = 2000$	72.4
rbm	$h = 3, L = 2000$	72.2
rbm	$h = 4, L = 1000$	69.8
rbm	$h = 4, L = 2000$	69.5
1-k	Dim = 25k	73.1
1-k	Dim = 50k	70.2
1-k	Dim = 100k	70.0
1-k	Dim = 200k	70.0

Table 14: Token Error Rates (%) for Combined Models

Model	Bengali	Cantonese
BEST SINGLE SYSTEM	69.5	65.7
rbm $(h = 3, L = 2000)$ + 1-k	69.7	65.3
rbm $(h = 4, L = 1000)$ + 1-k	69.2	64.9
rbm $(h = 4, L = 2000)$ + 1-k	69.1	64.9

kernel model, we computed the pre-activation similarly. Note that since they both predict the same set of labels, the pre-activations from either model have the same dimensionality.

We perform PCA on them independently and then visualize in 2D. Fig. 3 displays the two scatter plots where each has 1000 points, representing the means of the learned representations for data points in each class. To visualize easily, we color each point not by its phoneme state labels. Instead, we collapse them into phone labels (which are considerably few, generally around 40 - 60).

An initial examination seems to suggest that kernel models' representations tend to form clumps for data from the same class. In the figure, the most obvious observation is the cluster in the blue color. On the other end, those blue color scattered data points do not seem to form a large and tight cluster under the representations learned by the DNNs – they seem to be more spread out.

The clumps seem to be indicative of the Gaussian kernels we have used. However, how important they are and in what way, the more flourish patterns by DNNs' representations are more advantageous require more careful and detailed analysis. We hope our work has provided enough incentives and tools for that pursuit.
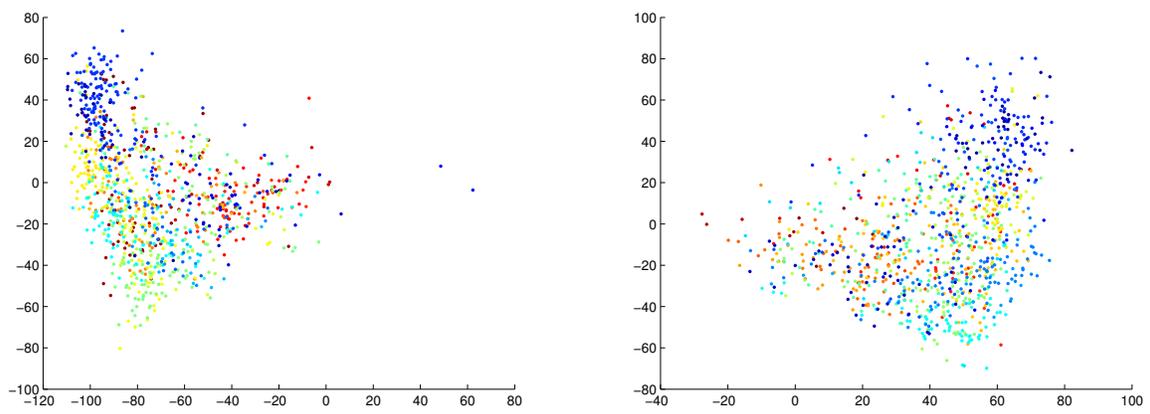
Figure 3: Contrast the learned representations by kernel models (left plot) and DNNs (right plot)