Continuous Kernel Learning

John Moeller, Vivek Srikumar, Sarathkrishna Swaminathan, Suresh Venkatasubramanian, and Dustin Webb

School of Computing, University of Utah, Salt Lake City, UT 84112, USA

Abstract. Kernel learning is the problem of determining the best kernel (either from a dictionary of fixed kernels, or from a smooth space of kernel representations) for a given task. In this paper, we describe a new approach to kernel learning that establishes connections between the Fourier-analytic representation of kernels arising out of Bochner's theorem and a specific kind of feed-forward network using cosine activations. We analyze the complexity of this space of hypotheses and demonstrate empirically that our approach provides scalable kernel learning superior in quality to prior approaches.

1 Introduction

Kernel methods have been a powerful tool in machine learning for decades and *kernel learning* is the problem of learning the "right" or "best" kernel for a given task. Broadly speaking, we can divide kernel learning methods into two categories. Multiple kernel learning (MKL) methods largely assume that the desired kernel can be represented as a combination of a dictionary of *fixed* kernels, and seeks to learn their mixing weights. The other approach is based on a Fourier-analytic representation of shift-invariant kernels via Bochner's theorem: roughly speaking, a kernel can be represented (in dual form) as a probability distribution, and so the search for a kernel becomes a search over distributions.

In both approaches, training the model is challenging with many thousands of training points and hundreds of dimensions. Standard training approaches either employ some form of convex or alternating optimization (for MKL) or parameterize the space of distributions in terms of known distributions and try to optimize their parameters.

In this paper, we describe *continuous kernel learning (CKL)*, a new way of tackling this problem by establishing and exploiting a connection to feed-forward networks. Working within the Fourier-analytic framework for kernel learning, we propose to search directly over the space of shift-invariant kernels instead of optimizing the parameters of a known family of distributions. In doing so, though we lose the ability to isolate parameters of a single learned kernel, we gain representability in terms of a nonlinear basis of cosines that can be naturally interpreted as activations for a feed-forward network. This interpretation allows us to deploy the power of backpropagation on this network to learn the desired kernel representation. In addition, the generalization power of the cosine representation can be established formally using machinery from learning theory: this also helps guide the regularization that we use to learn the resulting kernel. We support these arguments with a suite of experiments on relatively large data sets

(tens of thousands of points, hundreds of dimensions) that demonstrate that our learned kernels are more accurate than the state-of-the-art MKL methods.

In summary, our main contributions are:

- We develop the *continuous kernel learning* (CKL) framework, a kernel learning method that learns an implicit representation of a kernel. We show that we can interpret the learning task as a feed-forward network. This allows us to utilize recent advances in optimization technology from deep learning to train a classifier.
- We prove VC-dimension and generalization bounds for a single Fourier embedding, which yields natural regularization techniques for CKL.
- We show via experiments that CKL outperforms existing scalable MKL methods.

1.1 Technical Overview

The starting point for our work is the representation of any shift-invariant kernel[1] as an infinite linear combination of cosine basis elements via Bochner's theorem [9], as first demonstrated by Rahimi and Recht [38]. This representation is typically used to generate a *random* low-dimensional embedding of the associated Hilbert space.

If we move away from a random low-dimensional embedding and embrace the entire distribution that we sample from, we reach infinite-width embeddings. Dealing with infinite-width embeddings simply means that we consider the expectation of the embedding over the distribution. Neal [33] linked infinite-width networks to Gaussian processes when the distribution is Gaussian. Much later, Cho and Saul [11] applied the technique to infinite-width *rectified linear units* (ReLUs), and showed a correspondence to a kernel they called the *arc-cosine kernel*. Hazan and Jaakkola [20] extended this result further, and analyzed the kernel corresponding to two infinite layers stacked in series.In all of this, a *specific distribution* is chosen in order to obtain a kernel.

In our work, we return to the infinite representation provided by Bochner's theorem. Rather than picking a specific distribution over weights, we *learn* a distribution based on our training data. This effectively means we learn a *representation* of a kernel. While we cannot learn an infinite-width embedding directly, since the space of functions is itself infinite, we are able to construct approximate representations from a finite number of Fourier embeddings. Since the learned kernel representations are a form of kernel learning, we dub our technique *continuous kernel learning* (CKL).

2 Prior Kernel Learning Work

2.1 Multiple Kernel Learning (MKL)

Multiple Kernel Learning, or MKL, is an extension to kernelized support vector machines (SVMs) that employs a combination of kernels to extend the space of possible kernel functions. MKL algorithms learn not only the parameters of the SVM, but also the parameters of the kernel combination. In this sense, MKL algorithms seek to find

[1] A kernel $\kappa(x,y)$ expressible as $\kappa(x,y) = k(x-y)$.

the correct kernel function for the training data instead of relying on a predefined kernel function.

Lanckriet et al. [25] describe several convex optimization problems that learn the coefficients of a linear combination of kernel functions $\kappa_\gamma(\cdot,\cdot) = \sum_i \gamma_i \kappa_i(\cdot,\cdot)$. There are several algorithms to solve the MKL problem, including [1, 3, 16, 17, 39]. In addition to solving the MKL problem, MWUMKL [31] and SPG-GMKL [21] also work at scale.

2.2 Approaches Utilizing Bochner's Theorem

The key mathematical tool that drives much of kernel learning work is Bochner's theorem:

Theorem 1 (Bochner [9]). *A continuous function $k : \mathbb{R}^d \to \mathbb{R}$ is positive-definite iff $k(\cdot)$ is the Fourier transform of a non-negative measure.*

Several papers have been published that explore the connection between Bochner's theorem and learning a kernel. A Bayesian view produces an interpretation of this optimization as learning the kernel of a Gaussian process (GP). Wilson and Adams [43] equate stationary (shift-invariant) kernels to the spectral density function of a GP. They observe that linear combinations of squared-exponential kernels are dense in the space of stationary kernels. The resulting kernel has few parameters and is relatively easy to interpret.

Yang et al. [48] extend the ideas in [43] and combine it with the principles from Fastfood [26]. The authors also discuss variants of their algorithms such as computing a piecewise linear kernel. Similarly, the BaNK method by Oliva et al. [34] learns a kernel using the GP technique and trains the kernel using MCMC. Finally in the GP vein, Wilson et al. [44] integrate a deep network as input to the GP, treating the GP as an "infinite-dimensional" layer of the network, and optimize the parameters of the GP simultaneously with the parameters of the network using backpropagation.

Băzăvan et al. [10], in contrast, optimize Fourier embeddings, but decompose each ω_i into a parameter σ_i multiplied by a nonlinear function of a uniform random variable to represent the sample. The uniform variable is resampled during optimization as the parameter is learned.

3 Continuous Kernel Learning

3.1 Bochner's Theorem

A couple observations must be made in order for Theorem 1 to be relevant to our setting. First, we observe that (for the purposes of this paper) a positive definite *function* $k(\cdot)$ is a positive definite *kernel* $\kappa(\cdot,\cdot)$ when $\kappa(\mathbf{x},\mathbf{x}') = k(\mathbf{x}-\mathbf{x}')$. A kernel of this type is a *shift-invariant* kernel. Examples include the Gaussian or RBF kernel ($e^{-\|\mathbf{x}-\mathbf{x}'\|^2/\sigma^2}$) and the Laplacian kernel ($e^{-\lambda\|\mathbf{x}-\mathbf{x}'\|}$).

Next, any non-negative measure $\mu : \mathbb{R}^d \to \mathbb{R}^+$ can be converted to a probability distribution if we normalize by $Z = \int_{\mathbb{R}^d} d\mu$. Since Fourier transforms are linear, we can normalize the kernel by the same factor Z and maintain the equivalence. So without

loss of generality, we can assume that the measure μ is a probability measure. This equivalence between shift-invariant kernel and distribution is important in the rest of this paper.

3.2 Fourier Embeddings

Rahimi and Recht [38] built on Bochner's theorem by observing that the Fourier transform of μ is also an expectation:

$$k(\mathbf{x}-\mathbf{x}') = \int_{\mathbb{R}^d} e^{i\omega^\top(\mathbf{x}-\mathbf{x}')} f_\mu(\omega)\, d\omega \tag{1}$$

$$= E_\omega[\zeta_\omega(\mathbf{x})\overline{\zeta_\omega(\mathbf{x}')}], \tag{2}$$

if $\zeta_\omega(\mathbf{x}) = e^{i\omega^\top\mathbf{x}}$ and $\omega \sim \mathcal{D}_\mu$, where $\mathcal{D}_\mu$ is the probability distribution over Borel sets on $\mathbb{R}^d$ with measure μ. This shows that $\zeta_\omega(\mathbf{x})\overline{\zeta_\omega(\mathbf{x}')}$ is an unbiased estimate of $k(\mathbf{x}-\mathbf{x}')$. Because $k(\mathbf{x}-\mathbf{x}')$ is real, we know that $E_\omega[\zeta_\omega(\mathbf{x})\overline{\zeta_\omega(\mathbf{x}')}]$ has no imaginary component. A straightforward Chernoff-type argument [see 32, Ch. 4] shows that averaging $\zeta_\omega(\mathbf{x})\overline{\zeta_\omega(\mathbf{x}')}$ over D samples of ω produces a bound on the error of the estimate that diminishes exponentially in D. The lifting map then becomes $\Phi(\mathbf{x}) = \sqrt{1/D}(\zeta_{\omega_1}(\mathbf{x}),\ldots,\zeta_{\omega_D}(\mathbf{x}))$. The inner product $\langle\Phi(\mathbf{x}),\overline{\Phi(\mathbf{x}')}\rangle$ is obviously the desired average.

We can avoid complex numbers by using $z_{\omega,b}(\mathbf{x}) = \sqrt{2}\cos(\omega^\top\mathbf{x}+b)$ with $\omega\sim\mathcal{D}_\mu$ and $b\sim U[0,2\pi)$, which offers the same unbiased estimate (see [38]). The lifting map in this case is $\Phi(\mathbf{x}) = \sqrt{2/D}(z_{\omega_1,b_1}(\mathbf{x}),\ldots,z_{\omega_D,b_D}(\mathbf{x}))$. In this work we will refer to these maps (of the real or complex type) as *Fourier embeddings*. In [38] these embeddings are called *random Fourier features*, because they are selected at random from the distribution that is Fourier-dual to the approximated kernel. We will demonstrate that Fourier embeddings of this type need not be selected at random, and can in fact be optimized.

Our approach. Our approach is most similar to that in Băzăvan et al. [10]. Like the authors of [10], we recognize that we can optimize the parameters $\{\omega_i\}$ of a Fourier embedding. Băzăvan et al. decompose ω_i as follows:

$$\omega_i = \sigma_i \odot h(\mathbf{u}_i), \tag{3}$$

where σ_i is the parameter of a shift-invariant kernel, h is a nonlinear function (essentially an inverse CDF), and $\mathbf{u}_i$ is a sample drawn from a multivariate uniform distribution (cube). The procedure is to optimize σ_i and periodically resample $\mathbf{u}_i$. This has the advantage of being able to represent the kernel with its parameter σ_i, which adds to clarity, but the kernel must be one of a particular class of shift-invariant kernels that decomposes into this form. A Gaussian kernel, however, does decompose this way.

In contrast, we sample the vectors ω_i from the distribution $\mathcal{D}_\mu$, and then optimize them directly. The weights $\{\omega_i\}$ become different vectors $\{\omega_i'\}\subset\mathbb{R}^d$ – and are now very unlikely to be drawn i.i.d. from the distribution $\mathcal{D}_\mu$ anymore. As in prior approaches, by learning the embeddings, we learn the kernel, because the Bochner equivalence between distributions and kernels guarantees this. We use backpropagation to

learn the weights, avoiding the need to resample at every step, and allowing us to take advantage of recent neural network technology to perform scalable optimization. While other approaches focus on decomposing the representation of the kernels into individual kernel components and learn their parameters, we avoid this and focus only on producing the final weights ω_i'. We lose the clarity and sparsity of individual kernel parameters but gain the flexibility of learning a representation of a shift-invariant kernel free of individual base kernels, and recent technology allows us to do this training quickly.

For brevity, we refer to the $d \times D$ matrices $\mathbf{W}$ (for the $\{\omega_i\}$) and $\mathbf{W}'$ (for the $\{\omega_i'\}$), since there are D samples from $\mathbb{R}^d$.

3.3 Generalization Bounds in Fourier Embeddings

We now examine the capacity of this class of kernels by analyzing its VC-dimension. Note that the cosine function complicates this analysis since it has nontrivial gradient almost everywhere.

Fortunately we can exploit an observation already well-known in kernel learning that a narrow kernel function, for example, a Gaussian kernel with a small variance, is more likely to overfit (and therefore have higher capacity). This is because a narrow kernel function only allows the model to examine a very small range around each point, so a new point is unlikely to be affected by the model at all. Because the kernel is the Fourier transform of a distribution, a narrow kernel function corresponds to a distribution with high variance – using the same example, a Gaussian kernel with variance parameter σ^2 is the Fourier transform of a Gaussian distribution with variance $1/\sigma^2$. So a small variance in the kernel corresponds to a high variance in the distribution, and vice-versa.

In fact, we can demonstrate that if the norm of the embedding parameter ω is high, then this translates to higher capacity. Let $z(x) = e^{2\pi ix}$, $\mathsf{Re}(z)$ and $\mathsf{Im}(z)$ be the real and imaginary components of z, respectively, and let $\mathbf{1}_P(x)$ be the indicator (or characteristic) function of $P : \mathbb{R} \to \{0,1\}$.

Definition 1. *An* (ω, β, d)-range *is the set* $\{\mathbf{x} \in \mathbb{R}^d \mid \mathsf{Im}(z(\omega \cdot \mathbf{x} + \beta)) \geq 0, \|\mathbf{x}\| < 1\}$ *where* $d \geq 1$ *is an integer,* $\omega \in \mathbb{R}^d$*, and* $\beta \in [0,1)$.

Definition 2. *Let* $\mathcal{G}_d(R)$ *be the set of all* (ω, β, d)*-ranges such that* $\|\omega\|_2 \leq R$.

Clearly, every (ω, β, d)-range corresponds to a binary classifier and the range space $(\mathbb{R}^d, \mathcal{G}_d(R))$ is the hypothesis space of interest. We denote the unbounded range space $\cup_R \mathcal{G}_d(R)$ by $\mathcal{G}_d(\infty)$.

VC-dimension of (ω, β, d)-ranges.

Theorem 2. *The VC-dimension of the range space* $(\mathbb{R}^d, \mathcal{G}_d(R))$ *is* $\Theta(\max\{d \log R, d + 1\})$.

We split the proof of this theorem into two lemmas:

Lemma 1. *The VC-dimension of* $(\mathbb{R}^d, \mathcal{G}_d(R))$ *is at least* $d \max\{\lfloor \log_2 R \rfloor, 1\} + 1$.

Lemma 2. *The VC-dimension of* $(\mathbb{R}^d, \mathcal{G}_d(R))$ *is* $O(d \log R)$.

Proof of Lemma 1. We first prove Lemma 1 by using the following lemma (proved in Appendix B):

Lemma 3. *The decision function* $\mathbf{1}_{\mathsf{lm}(z(wx+\beta))\geq 0}$ *induces a unique binary labeling for the set* $x \in \{1/2^i\}_{i=1}^n$ *for every integer value of* $w \in [1..2^n]$*, and any* $\beta \in (0, 2^{-(n+1)})$.

Proof of Lemma 1. Let $n = \lfloor \log_2 R \rfloor$, for $R \geq 2$. We now construct a set of dn points. Along each axis of $\mathbb{R}^d$, place n points with corresponding coordinate from the set $\{1/2^i\}_{i=1}^n$. From Lemma 3, we know that we can induce a binary labeling on every axis-restricted set, using integers $1..2^n$. Given $\omega \in [1..2^n]^d$, each $\omega_j \in 1..2^n$ will give a unique labeling to the points on axis $j \in 1..d$, independent of any other axis j. Therefore we can uniquely label the whole set of dn points, for all possible labelings.

To add one more point to the set, we select a point $\mathbf{c}$, the d-dimensional vector with all coordinates equal to a constant c, and make sure that we can find values β_+ and β_- so that $\langle \mathbf{c}, \omega \rangle + \beta_+ \geq 0$ and $\langle \mathbf{c}, \omega \rangle + \beta_- < 0$, independently of ω. Observe that $\langle \mathbf{c}, \omega \rangle = c \sum_j \omega_j$, and that $d \leq \sum_j \omega_j \leq d2^n$. For $\langle \mathbf{c}, \omega \rangle + \beta_- < 0$ we need that $\beta_+ < -\langle \mathbf{c}, \omega \rangle$ for all ω, since the choice of β must be independent of ω. This means that first, $c < 0$ since $\beta_- > 0$ and $\sum_j \omega_j > 0$. Then $-cd \leq -\langle \mathbf{c}, \omega \rangle \leq -cd2^n$, so we need to pick $\beta_+ < -cd$. Similarly, we require $\beta_+ \geq -cd2^n$, and since $\beta_+ < 2^{-(n+1)}$, we need $-c < 1/d2^{-(2n+1)}$. Set $c = -1/d2^{2n+2}$, $\beta_+ = 2^{-(n+2)}$, and $\beta_- = 2^{-(2n+3)}$. We can now uniquely label $dn+1$ points for all possible labelings, when $R > 2$.

Regardless of the value of R, there is always a unique labeling of $d+1$ points induced by the range space, since we can restrict to a ball small enough that $\mathsf{lm}(z(\omega x + \beta)) = \sin(2\pi(\omega x + \beta))$ is monotonic for appropriate values of β. Within the ball, the range space is effectively the range of half-spaces, which has VC-dimension $d+1$. □

Lemma 1 yields a VC-dimension bound for the set of *all* ranges:

Corollary 1. *The VC-dimension of the range space* $(\mathbb{R}^d, \mathcal{G}_d(\infty))$ *is unbounded.*

Proof of Lemma 2. To prove Lemma 2, we require another lemma and proceed via the *shatter function* of $(\mathbb{R}^d, \mathcal{G}_d(R))$ [19]. For a positive integer n, the shatter function of a range space is the maximum highest number of subsets induced by the range space on any set of n points X_n. That is, any range $\mathcal{R}$ induces a subset of X_n simply by the intersection $\mathcal{R} \cap X_n$, and the shatter function counts all unique subsets of this type. The shatter function gives an idea of the power of a range space (and therefore the power of its associated hypothesis classes). The following lemma (proved in Appendix B) gives the shatter function:

Lemma 4. *The shatter function of* $(\mathbb{R}^d, \mathcal{G}_d(R))$ *is* $O(R^d n^{d+1})$.

The proof of Lemma 2 follows directly from the relation between the shatter function and VC-dimension [19] but we provide a proof for completeness:

Proof of Lemma 2. We begin by observing that if the VC-dimension is δ, then $2^\delta \leq cR^d\delta^{d+1}$, by the definitions of shatter function and VC-dimension. This inequality shows that δ cannot be polynomial in R, and cannot have any more than one log factor. Suppose that δ is $O(g(R,d)\log_2 R^d)$, for some $g(R,d) = o(\log R^d)$. Then $2^\delta = R^{d \cdot g(R,d)}$, so we know further that $g(R,d) \leq 1$. Therefore δ is $O(d \log R)$. □

With Lemmas 1 and 2, we have proven Theorem 2. The VC dimension also gives us a generalization bound, due to Bartlett and Mendelson [4]:

Theorem 3. *Let F be a class of ± 1-valued functions defined on a set $\mathcal{X}$. Let P be a probability distribution on $\mathcal{X} \times \{\pm 1\}$, and suppose that $(X_1, Y_1), \ldots, (X_n, Y_n)$ and (X, Y) are chosen independently according to P. Then for any positive integer n, w.p. $(1-\delta)$ over samples of length n, every $f \in F$ satisfies*

$$P(Y \neq f(X)) \leq \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{Y_i \neq f(X_i)} + O\left(\sqrt{\frac{\max\{d \log R, d+1\}}{n}} + \sqrt{\frac{\ln 1/\delta}{n}}\right) \tag{4}$$

Regularization. Theorems 2 and 3 immediately suggest three regularization techniques: First, we limit the norm of the Fourier weights with weight decay (a.k.a. L_2 regularization). Alternatively, we simply cap the norm of each Fourier weight vector to some constant at each round of the training. We can further control the initial capacity by setting the variance of the initializing distribution.

4 From an embedding to a feed-forward network

We now return to the single Fourier embedding

$$z_{\omega,b} = \sqrt{2}\cos(\omega^\top \mathbf{x} + b)$$

If we fix an input $\mathbf{x}$, then we can view the mapping $z_{\omega,b}$ as a neuron with a cosine activation function and biases of the form $b \in [0, 2\pi)$. We call this type of neuron a *cosine neuron*: such a neuron (with a cutoff to ensure zero support outside an interval) was introduced in [15].

Consider a layer of such cosine neurons, each with associated weight vector ω_j. Each of these weights can be viewed as a sample from *some* distribution, and therefore the entire ensemble is a (dual) representation of some shift-invariant kernel by Bochner's theorem. We can then write the associated classifier for such a combination. Denoting the bias vector by $\mathbf{b}$ and the collection of all the weight vectors ω_j by W, the resulting classifier (with a `softmax` layer to combine the individual activations and logarithmic loss), can be written as

$$\ell_{\log}(\text{softmax}(\cos(\mathbf{W} \cdot \mathbf{x_i} + \beta)), y_i), \tag{5}$$

where $\text{softmax}(\mathbf{r})_j = e^{r_j} / \sum_k e^{r_k}$, and $\ell_{\log}$ is the log loss.

What we now have is a standard (shallow) 2-layer network that we can train using backpropagation and stochastic gradient descent.

5 Experiments

We have designed our experiments to answer the following questions: (1) Does allowing the learning algorithm to pick an arbitrary kernel improve performance over

standard MKL techniques that are only allowed to select from a fixed library of kernels? (2) How does the learning algorithm for CKL adapt to large datasets and higher dimensions?

5.1 MKL vs. CKL on Small Datasets

Since CKL is proposed as an alternative to MKL, we compare CKL to two scalable MKL algorithms, namely SPG-GMKL [21] and MWUMKL [31].

Data Sets. All of the datasets used for the experiments are taken from the `libsvm` repository[2]. See Table 1 for details of the datasets.

Dataset	Features	Examples
Liver	6	345
Diabetes	8	768
Cod-RNA	8	59535
Breast Cancer	10	683
German-Numeric	24	1000
Mushroom	112	8192
Adult	123	32561
Gisette	5000	6000

Table 1. Datasets for comparison of MKL and CKL

Experimental Procedure. The data for Adult and Mushroom datasets consist of binary features (one-hot representations of categorical features), so no scaling was applied. Features were scaled to the range [-1, 1] for other datasets.

For MKL experiments, we used the *Scikit-Learn* Python package [37] for much of the testing infrastructure. For testing with MKL methods, the training data is split randomly into 75% training and 25% validation data. The random splits were repeated 100 times for all sets except Mushroom, Gisette, and Adult, which received 20 splits for considerations of time. The C parameter was selected through cross validation and for MWUMKL, the ε parameter was chosen to be 0.005, to achieve high accuracy while allowing all of the experiments to complete (the number of iterations of the algorithm in [31] is proportional to $1/\varepsilon$). We use two kernels: a linear kernel and a Gaussian kernel. For the Gaussian kernel, a wide range of γ are tried and the the best accuracy observed is used in the results.

For CKL experiments, the same test/train split was applied, and additionally, the training portion was split further into 75% training and 25% validation. We apply early stopping and momentum, and random searches for: the width (h_0) of the hidden layer, a parameter (σ) used for initializing the weights of the hidden layer, and the learning rate

[2] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

(ℓ) hyperparameters. Training was stopped if the validation objective did not decrease within 100 epochs and was otherwise permitted to run for up to $10,000$ epochs. Momentum was applied from the first epoch with a value of 0.5 that was increased to 0.99 over the course of 10 epochs.

Values for h_0 were selected from $\{2^i\}$ with i sampled uniformly from $[0..9]$, except for Gisette, where i was sampled uniformly from $[0..14]$. The weights of the hidden layer were sampled from a Gaussian distribution with variance σ selected from $\{2^i\}$ where i was sampled uniformly from $[-6..0]$. The weights of the softmax layer were selected from $U[-0.1,0.1]$. Finally ℓ was sampled from $LU[10^{-5},0.2)$[3]. 100 models with random hyperparameters were trained, and then the one with the highest performance was chosen and validated with 100 random splits (as described in the previous paragraph).

Results. The results are shown in Table 2. CKL is not different in any significant capacity from either GMKL or MWUMKL on very small datasets. Letting the learning algorithm pick an arbitrary kernel improves performance over standard MKL techniques that only choose a mixture of kernels. Additionally, we see that CKL adapts to large datasets and higher dimensions better than MKL.

Small Datasets	GMKL	MWUMKL	CKL
Liver	67.78% (4.78%)	59.34% (6.04%)	66.45% (6.19%)
Diabetes	77.06% (2.66%)	75.59% (2.92%)	76.08% (2.95%)
Cod-RNA	**87.31%** (0.13%)	72.42% (7.30%)	85.7% (1.14%)
Breast Cancer	97.14% (1.20%)	91.89% (2.22%)	96.87% (1.22%)
German-Numeric	73.05% (3.25%)	74.40% (3.01%)	76.14% (2.57%)
Mushroom	99.80% (0.08%)	99.93% (0.04%)	**100%** (0.0042%)
Adult Income	83.94% (0.28%)	76.90% (0.82%)	**84.80%** (0.35%)
Gisette	95.15% (0.53%)	93.50% (0.72%)	**96.90%** (0.52%)
Million Song Dataset	GMKL	MWUMKL	CKL
Genre 1	77.62% (0.36%)	68.14% (1.06%)	**81.68%** (0.39%)
Genre 2	69.12% (0.33%)	53.02% (0.55%)	**74.16%** (0.36%)
Year Pred.	75.38% (0.1%)	57.72% (1.64%)	**77.57%** (0.11%)

Table 2. Mean accuracies (standard deviations) for various datasets on MKL and CKL. If a mean, minus the standard deviation, is greater than all other means plus standard deviations in the row, then the mean is bold.

5.2 MKL vs. CKL on Million Song Datasets

In this section, we compare MKL methods with CKL on the Million Song Dataset [6]. The Million Song Dataset consists of audio features and metadata of one million contemporary popular music tracks. For the experiments, we utilized three different subsets

[3] A random variable X is drawn from $LU[a,b]$ if $X = e^Y$, where $Y \sim U[\ln(a),\ln(b))$.

of the Million Song Dataset, all binary. The features are the average and covariance of the pitch and timbre vectors for each track:

- **Genre 1:** The two most common genres in Million Song Dataset - "*classic pop and rock*" and "*folk*." The tracks which have both genres as tags are removed to avoid confusion.
- **Genre 2:** The ten most common genres in the Million Song Dataset. Since the "*classic pop and rock*" genre has significantly more tracks than any other genre, "*classic pop and rock*" is considered as one class and everything else together as another class.
- **Year Prediction:** Taken from the UCI Machine Learning Repository. All tracks prior to the year 2000 are considered as one class and all tracks after and including the year 2000 are considered as the other class.

The dimensions of the dataset are summarized in Table 3.

Name	Dimensions	Examples	Notes
Genre 1	182	$37,037$	"*Classic pop and rock*" vs. "*folk*"
Genre 2	182	$59,485$	"*Classic pop and rock*" vs. everything else
Year pred.	90	$515,345$	< 2000 vs. ≥ 2000

Table 3. Summary of Million Song Datasets

Results. The results are shown in Table 2. CKL is clearly superior to the scalable MKL methods that we tested against, adding to the evidence that higher-dimensional and larger datasets can benefit from our technique.

5.3 MKL vs. CKL on Images

We compare MKL and CKL on CIFAR10. CIFAR10 [24] is a labeled image dataset containing 60,000 1,024-dimensional (32×32) images and 10 classes used extensively for testing image classification algorithms. While image classification is an important benchmark for neural networks, we wish to point out that our objective is *not* to classify the CIFAR10 dataset better than all other previous techniques. Instead, we wish to provide comparisons between the methods described in this paper on a large and very challenging task using a simple convolutional neural architecture.

Preprocessing. We first centered the CIFAR10 training set by mean, and then used Pylearn2 [18] to apply two transformations: global contrast normalization [12] and ZCA whitening [5][4]. We applied the same transformations computed for the training set to the testing set.

[4] PCA whitening attempts to decorrelate features and normalize singular values ("whitening") of the original data by rotating the data by singular vectors, and then normalizing singular

Feature extraction. For MKL, we used a convolutional neural network (CNN) [27] to learn a representation from the data. In total, we trained 100 models and we extracted the features from the model with the best performance. All of the models had the form $\text{conv}_{\text{ReLU}} \rightarrow \text{pool}_{max} \rightarrow \text{fc}_{\text{ReLU}} \rightarrow \text{softmax}$ where $\text{conv}_{\text{ReLU}}$ is a convolutional layer using ReLU non-linearities, pool_{max} is a max-pool layer, fc_{ReLU} was a fully-connected layer using ReLU non-linearities, and $softmax$ was a *softmax* layer.

We trained the models with (1) momentum, initialized to 0.5 and increased to 0.99 over the first 100 epochs, and (2) early stopping: we set aside the last $10,000$ samples of the training set as a validation set for early stopping, and trained the models for at most $5,000$ epochs. We initialized the weights of all layers by selecting values uniformly at random from the range $[-0.01, 0.01]$.

The parameters of best performing model were as follows: (1) the convolutional layer (with ReLU activations): a 5×5 kernel with 1×1 stride, 32 channels, a max kernel norm of 1.8, and cross channel normalization with $\alpha = 3.2 \times 10^{-4}$ and $\beta = 0.75$, (2) the max pooling layer: a 3×3 kernel with 2×2 stride, (3) the fully connected layer: $1,000$ rectified linear units, and (4) the softmax layer: one output for each CIFAR10 class.

Each sample of CIFAR10 was passed through the CNN and the activations of the fully connected layer were recorded as the new representation.

CIFAR10 with MKL For MKL experiments, the testing infrastructure and the experimental procedures are similar to the experimental procedure of Section 5.1 except for the following details: (1) One-vs-one multiclass strategy is used for the classification task, (2) Random 75% of the training data is used for training and tested on the standard test data. The runs were repeated 20 times, and (3) We used two Gaussian kernels, one with $\gamma = 1$ and the other with a range of γ from 2^{-7} to 2^{7}. The best accuracy observed is used in Table 4.

CIFAR10 with CKL For comparison with MKL, we trained a network of the form $\text{conv}_{\text{ReLU}} \rightarrow \text{pool}_{max} \rightarrow \text{fc}_{\text{ReLU}} \rightarrow \text{fc}_{\cos} \rightarrow \text{softmax}$. A CKL model of this form uses the same structure as the CNN used for the MKL/CKL experiments (defined in the paragraph "Feature Extraction"), up to and including the fully connected layer of rectified linear units. Instead of a softmax layer, the units of the fully connected layer were connected to a CKL model with $1,000$ hidden units (untuned).

The primary difference between this model and MKL trained on features extracted from a CNN (see Section 5.3) is that this model is trained all at once, while in the MKL experiments the CNN used for feature learning and the MKL model were trained separately. This end-to-end learning allows the features of each layer to adapt to the features that appear later in the network. It is also important to note that the MKL experiments were trained on a one-vs.-one basis, while the CKL model uses multinomial (softmax) regression with log loss.

---

values. ZCA whitening, in contrast, attempts to do the same, but make the resulting data as close to the original as possible, in a least-squares sense. The ZCA transformation is simply to multiply by the inverse square root of the covariance matrix of the data.

Experimental procedure. The models in these experiments were trained using stochastic gradient descent for a maximum of $1,000$ epochs with early stopping and momentum. The initial momentum rate was 0.5 and was adjusted from the first epoch to 0.99 over the first 500 epochs of the training.

Results. The CKL model outstrips the MKL methods by a wide margin. We conjecture that this is due to two effects: (1) the end-to-end training allows for better adaptation in the training process and (2) the search space of kernels is much larger. The first effect demonstrates that CKL is more adaptable than MKL in these settings. It is also important to note that training is a crucial component for CKL models when operating on large datasets. In the case of CIFAR10, evaluating any random model upon initialization yielded an accuracy of only 10.1% with standard deviation of 0.235%. In contrast, evaluating random models on smaller datasets frequently yields accuracies that are better than chance.

GMKL	MWUMKL	CKL+CNN
44.43% (0.57%)	48.2% (0.41%)	**67.77%** (0.61%)

Table 4. Accuracy for CIFAR10 on MKL and CKL with CNN.

CIFAR10 with Two Layer Convnets A natural question to ask is whether stacking two cosine layers has any beneficial effect. Stacking two cosine layers is approximately the same as composing two lifting maps. If the composed lifting map is defined, then it corresponds to a kernel.

Zhuang et al. [50] construct an algorithm specifically for the composition of two kernels – essentially layering the kernels. Lu et al. [28] discuss extensions to [38] that cover products, sums, and compositions of kernels. Since these are based on the sampling methodology of [38], there is a direct analogy to composing two cosine layers (fixed, in this case).

We used various combinations of cosine (cos) and rectified linear (ReLU) activation functions to see whether those involving cosine do in fact yield better accuracy on average. In fact, we did not observe significant improvement in accuracy when we employed combinations of two cosine layers. One possible explanation is that since the composition of a kernel is itself a kernel, it can be argued that optimizing a network that contains two consecutive cosine layers accomplishes no more than doing so with one individual cosine layer.

6 Related Work

Multiple kernel learning. The general area of kernel learning was initiated by Lanckriet et al. [25] who proposed to simultaneously train an SVM as well as learn a convex combination of kernel functions. The key contribution was to frame the learning problem as

an optimization over positive semi-definite kernel matrices which in turn reduces to a QCQP. Soon after, Bach et al. [3] proposed a block-norm regularization method based on *second order cone programming* (SOCP).

For efficiency, researchers started using optimization methods that alternate between updating the classifier parameters and the kernel weights. Many authors then explored the MKL landscape, including Rakotomamonjy et al. [39], Sonnenburg et al. [40], Xu et al. [45, 46]. However, as pointed out by Cortes [13], most of these methods do not compare favorably (both in accuracy as well as speed) even with the simple *uniform* heuristic. More recently, Moeller et al. [31] developed a multiplicative-weight-update based approach that has a much smaller memory footprint and scales far more effectively. Other kernel learning methods include [14, 30, 35, 36, 41] and notably methods using the ℓ_p-norm [22, 23, 42].

Infinite-width networks. Infinite networks have been analyzed in the literature at least as far back as Neal [33] – in this work, the authors tie infinite networks to Gaussian processes, assuming that the distribution is Gaussian. Cho and Saul [11] analyzed the case where the network is either a step network (the output is 1 if the input is positive, 0 otherwise) or a rectified linear unit (ReLU), a type of network used frequently in deep networks (the input z is passed through the function $\max\{0, z\}$). They showed that if the distribution is Gaussian in these settings, the function $\phi_{\mathbf{x}}$ output by the network is a lifting map corresponding to a kernel they dub the *arc-cosine kernel*.

Hazan and Jaakkola [20] extended this result further, and analyzed the kernel corresponding to two infinite layers stacked in series. They showed that such a network, when the distribution of the first layer is Gaussian, and the second layer is treated as a Gaussian process, (a process is a distribution of distributions), corresponds to a kernel that can be computed explicitly.

Layered kernels. Zhuang et al. [50] develop a multiple kernel learning technique where they use a layered kernel to combine the output of several other kernels. Their algorithm alternates the use of standard SVM and stochastic gradient descent. Lu et al. [28] scale up [38] by making some interesting mathematical observations about kernels and distributions. Their work relies heavily on the correspondence between distributions and kernels, a theme that we explore as well. Yu et al. [49] also seek to optimize a kernel, using alternating optimization and also based on Bochner's theorem.

Neural networks as kernels. Yang et al. [47] draw on the idea of a correspondence between ReLUs and arc-cosine kernels, and replace the weights of a ReLU with a fastfood transform [26] to reduce the complexity of the network. Aslan et al. [2] seek to make the optimization of neural networks convex through kernels and matrix techniques. Mairal et al. [29] extend hierarchical kernel descriptors [7, 8] to act as convolutional layers. Very recently, Wilson et al. [44] combine neural networks with Gaussian processes, drawing on the infinite-width network setting, to produce "deep" kernels.

Bibliography

[1] F. Aiolli and M. Donini. EasyMKL: a scalable multiple kernel learning algorithm. *Neurocomputing*, 169:215–224, 2015.

[2] O. Aslan, X. Zhang, and D. Schuurmans. Convex Deep Learning via Normalized Kernels. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *NIPS*, pages 3275–3283. Curran Associates, Inc., 2014.

[3] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML*, Banff, Canada, 2004.

[4] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. *JMLR*, 3:463–482, Mar. 2003.

[5] A. J. Bell and T. J. Sejnowski. Edges are the 'Independent Components' of Natural Scenes. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *NIPS*, pages 831–837. MIT Press, 1997.

[6] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. The Million Song Dataset. In *ISMIR*, 2011.

[7] L. Bo, X. Ren, and D. Fox. Kernel Descriptors for Visual Recognition. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *NIPS*, pages 244–252. Curran Associates, Inc., 2010.

[8] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *CVPR*, pages 1729–1736, June 2011.

[9] S. Bochner. *Lectures on Fourier integrals*. Number 42 in Annals of Mathematics Studies. Princeton University Press, 1959.

[10] E. G. Băzăvan, F. Li, and C. Sminchisescu. Fourier Kernel Learning. In *ECCV*, 2012.

[11] Y. Cho and L. K. Saul. Kernel Methods for Deep Learning. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *NIPS*, pages 342–350. Curran Associates, Inc., 2009.

[12] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *AIStats*, pages 215–223, 2011.

[13] C. Cortes. Invited talk: Can learning kernels help performance? In *ICML*, Montreal, Canada, 2009.

[14] C. Cortes, M. Mohri, and A. Rostamizadeh. Learning Non-Linear Combinations of Kernels. In *NIPS*, Vancouver, Canada, 2009.

[15] A. Gallant and H. White. There exists a neural network that does not make avoidable mistakes. In *, IEEE International Conference on Neural Networks, 1988*, pages 657–664 vol.1, July 1988. doi: 10.1109/ICNN.1988.23903.

[16] M. Gönen and E. Alpaydın. Localized multiple kernel learning. In *ICML*, Helsinki, Finland, 2008.

[17] M. Gönen and E. Alpaydın. Localized algorithms for multiple kernel learning. *Pattern Recognition*, 46(3):795–807, 2013.

[18] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research

library. *arXiv:1308.4214 [cs, stat]*, Aug. 2013. URL `http://arxiv.org/abs/1308.4214`. arXiv: 1308.4214.

[19] S. Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, Boston, MA, USA, 2011.

[20] T. Hazan and T. Jaakkola. Steps Toward Deep Kernel Methods from Infinite Neural Networks. *arXiv:1508.05133 [cs]*, Aug. 2015. URL `http://arxiv.org/abs/1508.05133`. arXiv: 1508.05133.

[21] A. Jain, S. V. N. Vishwanathan, and M. Varma. SPG-GMKL: generalized multiple kernel learning with a million kernels. In *KDD*, pages 750–758, 2012.

[22] M. Kloft, U. Brefeld, S. Sonnenburg, P. Laskov, K.-R. Müller, and A. Zien. Efficient and Accurate Lp-Norm Multiple Kernel Learning. In *NIPS*, Vancouver, Canada, 2009.

[23] M. Kloft, U. Brefeld, S. Sonnenburg, and A. Zien. Lp-norm multiple kernel learning. *JMLR*, 12:953–997, 2011.

[24] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Citeseer, 2009.

[25] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the Kernel Matrix with Semidefinite Programming. *JMLR*, 5:27–72, Dec. 2004.

[26] Q. Le, T. Sarlos, and A. Smola. Fastfood - Computing Hilbert Space Expansions in loglinear time. In *ICML*, pages 244–252, 2013.

[27] Y. LeCun. Generalization and Network Design Strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier. an extended version was published as a technical report of the University of Toronto.

[28] Z. Lu, A. May, K. Liu, A. B. Garakani, D. Guo, A. Bellet, L. Fan, M. Collins, B. Kingsbury, M. Picheny, and F. Sha. How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets. *arXiv:1411.4000 [cs, stat]*, Nov. 2014. arXiv: 1411.4000.

[29] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional Kernel Networks. In *NIPS*, pages 2627–2635, 2014.

[30] C. A. Micchelli and M. Pontil. Learning the Kernel Function via Regularization. *JMLR*, 6:1099–1125, Dec. 2005.

[31] J. Moeller, P. Raman, S. Venkatasubramanian, and A. Saha. A Geometric Algorithm for Scalable Multiple Kernel Learning. In *AIStats*, pages 633–642, 2014.

[32] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[33] R. M. Neal. Priors for Infinite Networks. In *Bayesian Learning for Neural Networks*, number 118 in Lecture Notes in Statistics, pages 29–53. Springer New York, 1996.

[34] J. Oliva, A. Dubey, B. Poczos, J. Schneider, and E. P. Xing. Bayesian Nonparametric Kernel-Learning. *arXiv:1506.08776 [stat]*, June 2015. URL `http://arxiv.org/abs/1506.08776`. arXiv: 1506.08776.

[35] C. S. Ong, A. J. Smola, and R. C. Williamson. Learning the Kernel with Hyperkernels. *JMLR*, 6:1043–1071, 2005.

[36] F. Orabona and J. Luo. Ultra-Fast Optimization Algorithm for Sparse Multi Kernel Learning. In *ICML*, Bellevue, USA, 2011.

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

[38] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, pages 1177–1184, 2007.

[39] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *ICML*, Corvalis, USA, 2007.

[40] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large Scale Multiple Kernel Learning. *JMLR*, 7:1531–1565, Dec. 2006.

[41] M. Varma and B. R. Babu. More generality in efficient multiple kernel learning. In *ICML*, Montreal, Canada, 2009.

[42] S. V. N. Vishwanathan, Z. Sun, N. Ampornpunt, and M. Varma. Multiple Kernel Learning and the SMO Algorithm. In *NIPS*, Vancouver, Canada, 2010.

[43] A. Wilson and R. Adams. Gaussian Process Kernels for Pattern Discovery and Extrapolation. In *ICML*, pages 1067–1075, 2013.

[44] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep Kernel Learning. *arXiv:1511.02222 [cs, stat]*, Nov. 2015. URL `http://arxiv.org/abs/1511.02222`. arXiv: 1511.02222.

[45] Z. Xu, R. Jin, I. King, and M. R. Lyu. An Extended Level Method for Efficient Multiple Kernel Learning. In *NIPS*, Vancouver, Canada, 2008.

[46] Z. Xu, R. Jin, H. Yang, I. King, and M. R. Lyu. Simple and Efficient Multiple Kernel Learning by Group Lasso. In *ICML*, Haifa, Israel, 2010.

[47] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang. Deep Fried Convnets. *arXiv:1412.7149 [cs, stat]*, Dec. 2014. URL `http://arxiv.org/abs/1412.7149`. arXiv: 1412.7149.

[48] Z. Yang, A. Wilson, A. Smola, and L. Song. À la Carte – Learning Fast Kernels. In *AIStats*, pages 1098–1106, 2015.

[49] F. X. Yu, S. Kumar, H. Rowley, and S.-F. Chang. Compact Nonlinear Maps and Circulant Extensions. *arXiv:1503.03893 [cs, stat]*, Mar. 2015. URL `http://arxiv.org/abs/1503.03893`. arXiv: 1503.03893.

[50] J. Zhuang, I. W. Tsang, and S. Hoi. Two-layer multiple kernel learning. In *AIStats*, pages 909–917, 2011.