Large margin classification in infinite neural networks

Youngmin Cho and **Lawrence K. Saul**

Department of Computer Science and Engineering

University of California, San Diego

9500 Gilman Drive

La Jolla, CA 92093-0404

Abstract

We introduce a new family of positive-definite kernels for large margin classification in support vector machines (SVMs). These kernels mimic the computation in large neural networks with one layer of hidden units. We also show how to derive new kernels, by recursive composition, that may be viewed as mapping their inputs through a series of nonlinear feature spaces. These recursively derived kernels mimic the computation in deep networks with multiple hidden layers. We evaluate SVMs with these kernels on problems designed to illustrate the advantages of deep architectures. Comparing to previous benchmarks, we find that on some problems, these SVMs yield state-of-the-art results, beating not only other SVMs, but also deep belief nets.

Keywords: Kernel methods, neural networks, large margin classification.

1 Introduction

Kernel methods provide a powerful framework for pattern analysis and classification (Boser et al., 1992; Cortes & Vapnik, 1995; Schölkopf & Smola, 2001). Intuitively, the so-called "kernel trick" works by mapping inputs into a nonlinear, potentially infinite-dimensional feature space, then applying classical linear methods in this space. The mapping is induced by a kernel function that operates on pairs of inputs and computes a generalized inner product. Typically, the kernel function measures some highly non-linear or domain-specific notion of similarity.

This general approach has been particularly successful for problems in classification, where kernel methods are most often used in conjunction with support vector machines (SVMs) (Boser et al., 1992; Cortes & Vapnik, 1995; Cristianini & Shawe-Taylor, 2000). SVMs have many elegant properties. Working in nonlinear feature space, SVMs compute the hyperplane decision boundary that separates positively and negatively labeled examples by the largest possible margin. The required optimization can be formulated as a convex problem in quadratic programming. Theoretical analyses of SVMs have also succeeded in relating the margin of classification to their expected generalization error. These computational and statistical properties of SVMs account in large part for their many empirical successes.

Notwithstanding these successes, however, recent work in machine learning has highlighted various circumstances that appear to favor deep architectures, such as multilayer neural networks and deep belief networks, over shallow architectures such as SVMs (Bengio & LeCun, 2007). Deep architectures learn complex mappings by transforming their inputs through multiple layers of nonlinear processing (Hinton et al., 2006). Researchers have advanced a number of different motivations for deep architectures: the wide range of functions that can be parameterized by composing weakly nonlinear transformations, the appeal of hierarchical distributed representations, and the potential for combining methods in unsupervised and supervised learning. Experiments have also shown the benefits of deep learning in several interesting applications (Hinton & Salakhutdinov, 2006; Ranzato et al., 2007; Collobert & Weston, 2008).

Many issues surround the ongoing debate over deep versus shallow architectures (Bengio & LeCun, 2007; Bengio, 2009). Deep architectures are generally more difficult to train than shallow ones. They involve highly nonlinear optimizations and many heuristics for gradient-based learning. These challenges of deep learning explain the early and continued appeal of SVMs. Unlike deep architectures, SVMs are trained by solving a problem in convex optimization. On the other hand, SVMs are seemingly unequipped to discover the rich internal representations of multilayer neural networks.

In this paper, we develop and explore a new connection between these two different approaches to statistical learning. Specifically, we introduce a new family of positive-definite kernels that mimic the computation in large neural networks with one or more layers of hidden units. They mimic this computation in the following sense: given a large neural network with Gaussian-distributed weights, and the multidimensional nonlinear outputs of such a neural network from inputs $\mathbf{x}$ and $\mathbf{y}$, we derive a kernel function $k(\mathbf{x}, \mathbf{y})$ that approximates the inner product computed directly between the outputs of the neural network. Put another way, we show that the nonlinear feature spaces induced by these kernels encode internal representations similar to those of single-layer or multilayer neural networks. Having introduced these kernels briefly in earlier work (Cho & Saul, 2009), in this paper we provide a more complete description of their properties and a fuller evaluation of their use in SVMs.

The organization of this paper is as follows. In section 2, we derive this new family of kernels and contrast their properties with those of other popular kernels for SVMs. In section 3, we evaluate SVMs with these kernels on several problems in binary and multiway classification. Finally, in section 4, we conclude by summarizing our most important contributions, reviewing related work, and suggesting directions for future research.

2 Arc-cosine kernels

In this section, we develop a new family of kernel functions for computing the similarity of vector inputs $\mathbf{x}, \mathbf{y} \in \Re^d$. As shorthand, let $\Theta(z) = \frac{1}{2}(1 + \mathrm{sign}(z))$ denote the Heaviside step function. We define the nth order arc-cosine kernel function via the integral representation:

$$k_n(\mathbf{x}, \mathbf{y}) \;=\; 2\int d\mathbf{w}\, \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\, \Theta(\mathbf{w}\cdot\mathbf{x})\, \Theta(\mathbf{w}\cdot\mathbf{y})\, (\mathbf{w}\cdot\mathbf{x})^n\, (\mathbf{w}\cdot\mathbf{y})^n. \qquad (1)$$

The kernel function in eq. (1) has interesting connections to neural computation (Williams, 1998) that we explore further in sections 2.2–2.3. However, we begin by elucidating its basic properties.

2.1 Basic properties

We focus primarily on kernel functions in this family with non-negative integer values of $n \in \{0, 1, 2, \ldots\}$. For such values, we show how to evaluate the integral in eq. (1) analytically in the appendix[1]. The final result is most easily expressed in terms of the angle θ between the inputs:

$$\theta \;=\; \cos^{-1}\left(\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right). \tag{2}$$

The integral in eq. (1) has a simple, trivial dependence on the magnitudes of the inputs $\mathbf{x}$ and $\mathbf{y}$, but a complex, interesting dependence on the angle between them. In particular, we can write:

$$k_n(\mathbf{x},\mathbf{y}) \;=\; \frac{1}{\pi}\,\|\mathbf{x}\|^n\|\mathbf{y}\|^n J_n(\theta) \tag{3}$$

where all the angular dependence is captured by the family of functions $J_n(\theta)$. Evaluating the integral in the appendix, we show that this angular dependence is given by:

$$J_n(\theta) \;=\; (-1)^n(\sin\theta)^{2n+1}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi-\theta}{\sin\theta}\right) \quad \text{for} \quad \forall n \in \{0,1,2,\ldots\}. \tag{4}$$

For $n=0$, this expression reduces to the supplement of the angle between the inputs. However, for $n>0$, the angular dependence is more complicated. The first few expressions are:

$$J_0(\theta) \;=\; \pi-\theta \tag{5}$$

$$J_1(\theta) \;=\; \sin\theta + (\pi-\theta)\cos\theta \tag{6}$$

$$J_2(\theta) \;=\; 3\sin\theta\cos\theta + (\pi-\theta)(1+2\cos^2\theta). \tag{7}$$

Higher-order expressions can be computed from eq. (4). We describe eq. (3) as an arc-cosine kernel because for $n=0$, it takes the simple form $k_0(\mathbf{x},\mathbf{y}) = 1-\frac{1}{\pi}\cos^{-1}\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$.

We briefly consider the kernel function in eq. (1) for non-integer values of n. The general form in eq. (3) still holds, but the function $J_n(\theta)$ does not have a simple analytical form. However, it remains possible to evaluate the integral in eq. (1) for the special

[1]Interestingly, this computation was also carried out earlier in different context (Price, 1958).

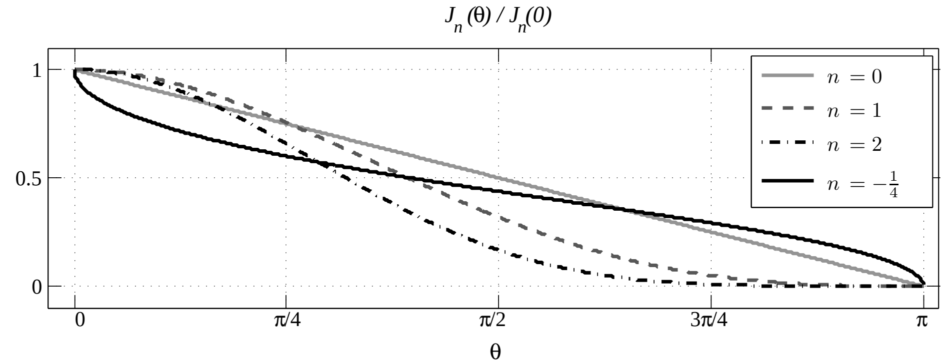

Figure 1: Form of $J_n(\theta)$ in eq. (3) for arc-cosine kernels with different values of n. The function $J_n(\theta)$ takes its maximum value at $\theta = 0$ and decays monotonically to zero at $\theta = \pi$ for all values of n. However, note the different behaviors for small θ.

case where $\mathbf{x} = \mathbf{y}$. In the appendix, we show that:

$$k_n(\mathbf{x}, \mathbf{x}) \;=\; \frac{1}{\pi} \|\mathbf{x}\|^{2n} J_n(0) \quad \text{where} \quad J_n(0) \;=\; \sqrt{\pi}\, 2^n\, \Gamma\left(n + \frac{1}{2}\right). \tag{8}$$

Note that this expression diverges as $J_n(0) \sim (n + \frac{1}{2})^{-1}$ due to a non-integrable singularity. Thus the family of arc-cosine kernels is only defined for $n > -\frac{1}{2}$; for smaller values, the integral in eq. (1) is not defined. The magnitude of $k_n(\mathbf{x}, \mathbf{x})$ also diverges for fixed non-zero $\mathbf{x}$ as $n \to \infty$. Thus as n takes on increasingly positive or negative values within its allowed range, the kernel function in eq. (1) maps inputs to larger and larger vectors in feature space.

The arc-cosine kernel exhibits qualitatively different behavior for negative values of n. For example, when $n < 0$, the kernel function performs an inversion, mapping the origin in input space to infinity in feature space, with $k_n(\mathbf{x}, \mathbf{x}) \sim \|\mathbf{x}\|^{2n}$. We are not aware of other kernel functions with this property. Though $J_n(\theta)$ does not have a simple analytical form for $n < 0$, it can be computed numerically; more details are given in the appendix. Figure 1 compares the form of $J_n(\theta)$ for different settings of n. For $n < 0$, note that $J_n(\theta)$ decays quickly away from its maximum value at $\theta = 0$. This decay serves to magnify small differences in angle between nearby inputs.

Finally, we verify that the arc-cosine kernels $k_n(\mathbf{x}, \mathbf{y})$ are positive-definite. This property follows immediately from the integral representation in eq. (1) by observing that it defines a covariance function (i.e., the expected product of functions evaluated at $\mathbf{x}$ and $\mathbf{y}$).

2.2 Computation in single-layer threshold networks

Consider the single-layer network shown in Figure 2 (left) whose weights W_{ij} connect the jth input unit to the ith output unit (i.e., $\mathbf{W}$ is of size m-by-d). The network maps inputs $\mathbf{x}$ to outputs $\mathbf{f}(\mathbf{x})$ by applying an elementwise nonlinearity to the matrix-vector product of the inputs and the weight matrix: $\mathbf{f}(\mathbf{x}) = g(\mathbf{W}\mathbf{x})$. The nonlinearity is described by the network's so-called activation function. Here we consider the family of piecewise-smooth activation functions:

$$g_n(z) \ = \ \Theta(z)z^n \tag{9}$$

illustrated in the right panel of Figure 2. For $n = 0$, the activation function is a step function, and the network is an array of perceptrons. For $n=1$, the activation function is a ramp function (or rectification nonlinearity (Hahnloser et al., 2003)), and the mapping $\mathbf{f}(\mathbf{x})$ is piecewise linear. More generally, the nonlinear behavior of these networks is induced by thresholding on weighted sums. We refer to networks with these activation functions as single-layer threshold networks of degree n.

Computation in these networks is closely connected to computation with the arc-cosine kernel function in eq. (1). To see the connection, consider how inner products are transformed by the mapping in single-layer threshold networks. As notation, let the vector $\mathbf{w}_i$ denote ith row of the weight matrix $\mathbf{W}$. Then we can express the inner product between different outputs of the network as:

$$\mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{y}) \ = \ \sum_{i=1}^{m} \Theta(\mathbf{w}_i \cdot \mathbf{x})\Theta(\mathbf{w}_i \cdot \mathbf{y})(\mathbf{w}_i \cdot \mathbf{x})^n(\mathbf{w}_i \cdot \mathbf{y})^n, \tag{10}$$

where m is the number of output units. The connection with the arc-cosine kernel function emerges in the limit of very large networks (Neal, 1996; Williams, 1998). Imagine that the network has an infinite number of output units, and that the weights W_{ij} are Gaussian distributed with zero mean and unit variance. In this limit, we see that eq. (10) reduces to eq. (1) up to a trivial multiplicative factor:

$$\lim_{m\to\infty} \left[\frac{2}{m}\, \mathbf{f}(\mathbf{x})\cdot\mathbf{f}(\mathbf{y})\right] \ = \ k_n(\mathbf{x}, \mathbf{y}). \tag{11}$$

Thus the arc-cosine kernel function in eq. (1) can be viewed as the inner product between feature vectors derived from the mapping of an infinite single-layer threshold network.

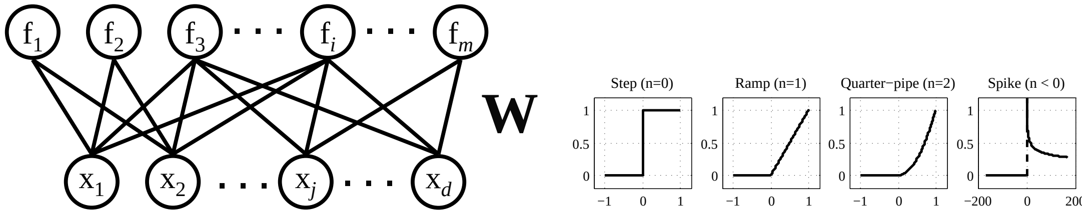

Figure 2: A single-layer threshold network (*left*) and nonlinear activation functions (*right*) for different values of n in eq. (9): step ($n = 0$), ramp ($n = 1$), quarter-pipe ($n = 2$), and spike ($n < 0$).

Many researchers have noted the general connection between kernel machines and one layer neural networks (Bengio & LeCun, 2007). Interestingly, the $n = 0$ arc-cosine kernel in eq. (1) can also be derived from an earlier result obtained in the context of Gaussian processes. Specifically, Williams (1998) derived a covariance function for Gaussian processes that mimic the computation in infinite neural networks with sigmoidal activation functions. To derive this covariance function, he evaluated a similar integral as eq. (1) for $n = 0$, but with two differences: first, the weights $\mathbf{w}$ were integrated over a Gaussian distribution with a general covariance matrix; second, the activation function was an error function with range $[-1, 1]$ as opposed to a step function with range $[0, 1]$. Specializing to a covariance matrix that is a multiple of the identity matrix, and taking the limit of very large variances, the result in Williams (1998) reduces to a kernel function that mimics the computation in infinite neural networks with activation functions $\Theta(\mathbf{x}) - \Theta(-\mathbf{x})$. The kernel function in this limit is expressed in terms of arc-sine functions of normalized dot products between inputs. Using elementary identities, our result for the $n = 0$ arc-cosine kernel follows as a simple corollary.

More generally, however, we are unaware of any previous theoretical or empirical work on the general family of these kernels for degrees $n > -\frac{1}{2}$. In this paper, we focus on the use of these kernels for large margin classification. Viewing these kernels as covariance functions for Gaussian processes, it also follows that ridge-regression with arc-cosine kernels is equivalent to MAP estimation in neural networks with a Gaussian prior. Thus our results also expand the family of neural networks whose computations can be mimicked (or perhaps more tractably implemented) by Gaussian processes (Neal, 1996; Williams, 1998).

2.3 Computation in multilayer threshold networks

A kernel function can be viewed as inducing a nonlinear mapping from inputs $\mathbf{x}$ to feature vectors $\mathbf{\Phi}(\mathbf{x})$. The kernel computes the inner product in the induced feature space:

$$k(\mathbf{x},\mathbf{y}) \;=\; \mathbf{\Phi}(\mathbf{x})\cdot\mathbf{\Phi}(\mathbf{y}). \tag{12}$$

In this section, we consider how to compose the nonlinear mappings induced by kernel functions, an idea suggested in Schölkopf et al. (1996). Specifically, we show how to derive new kernel functions

$$k^{(\ell)}(\mathbf{x},\mathbf{y}) \;=\; \underbrace{\mathbf{\Phi}(\mathbf{\Phi}(...\mathbf{\Phi}}_{\ell\text{ times}}(\mathbf{x})))\cdot\underbrace{\mathbf{\Phi}(\mathbf{\Phi}(...\mathbf{\Phi}}_{\ell\text{ times}}(\mathbf{y}))) \tag{13}$$

which compute the inner product after ℓ successive applications of the nonlinear mapping $\mathbf{\Phi}(\cdot)$. Our motivation is the following: intuitively, if the base kernel function $k(\mathbf{x},\mathbf{y}) = \mathbf{\Phi}(\mathbf{x})\cdot\mathbf{\Phi}(\mathbf{y})$ models the computation in a single-layer network, then the iterated mapping in eq. (13) should model the computation in a multilayer network.

We first examine the results of this procedure for widely used kernels. Here we find that the iterated mapping in eq. (13) does not yield particularly interesting results. For instance, consider the two-fold composition that maps $\mathbf{x}$ to $\mathbf{\Phi}(\mathbf{\Phi}(\mathbf{x}))$. For homogeneous polynomial kernels $k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^d$, the composition yields:

$$\mathbf{\Phi}(\mathbf{\Phi}(\mathbf{x}))\cdot\mathbf{\Phi}(\mathbf{\Phi}(\mathbf{y})) = (\mathbf{\Phi}(\mathbf{x})\cdot\mathbf{\Phi}(\mathbf{y}))^d = (\mathbf{x}\cdot\mathbf{y})^{d^2}. \tag{14}$$

The above result is not especially interesting: the kernel implied by this composition is also polynomial, just of higher degree. Likewise, for RBF kernels $k(\mathbf{x},\mathbf{y}) = e^{-\lambda\|\mathbf{x}-\mathbf{y}\|^2}$, the composition yields $e^{-2\lambda(1-k(\mathbf{x},\mathbf{y}))}$. Though non-trivial, this does not represent a particularly interesting computation. Recall that RBF kernels mimic the computation of soft vector quantizers, yielding $k(\mathbf{x},\mathbf{y}) \ll 1$ when $\|\mathbf{x}-\mathbf{y}\|$ is large compared to the kernel width. It is hard to see how the iterated mapping $\mathbf{\Phi}(\mathbf{\Phi}(\mathbf{x}))$ would generate a qualitatively different representation than the original mapping $\mathbf{\Phi}(\mathbf{x})$.

Next we consider the ℓ-fold composition in eq. (13) for arc-cosine kernel functions. We work out a simple example before stating the general formula. Consider the $n=0$ arc-cosine kernel, for which $k_0(\mathbf{x},\mathbf{y}) = 1-\frac{\theta}{\pi}$, where θ is the angle between $\mathbf{x}$ and $\mathbf{y}$.

For this kernel, it follows that:

$$\Phi(\Phi(\mathbf{x}))\cdot\Phi(\Phi(\mathbf{y})) \;=\; 1-\frac{1}{\pi}\cos^{-1}\left(\frac{\Phi(\mathbf{x})\cdot\Phi(\mathbf{y})}{\|\Phi(\mathbf{x})\|\|\Phi(\mathbf{y}))\|}\right) \tag{15}$$

$$= \; 1-\frac{1}{\pi}\cos^{-1}\left(\frac{k_0(\mathbf{x},\mathbf{y})}{\sqrt{k_0(\mathbf{x},\mathbf{x})\,k_0(\mathbf{y},\mathbf{y})}}\right) \tag{16}$$

$$= \; 1-\frac{1}{\pi}\cos^{-1}\left(1-\frac{\theta}{\pi}\right). \tag{17}$$

More generally, we can work out a recursive formula for the ℓ-fold composition in eq. (13). The base case is given by eq. (3) for kernels of depth $\ell = 1$ and degree n. Substituting into eq. (3), we obtain the construction for kernels of greater depth:

$$k_n^{(\ell+1)}(\mathbf{x},\mathbf{y}) \;=\; \frac{1}{\pi}\left[\,k_n^{(\ell)}(\mathbf{x},\mathbf{x})\,k_n^{(\ell)}(\mathbf{y},\mathbf{y})\,\right]^{n/2}\,J_n\big(\theta_n^{(\ell)}\big)\,, \tag{18}$$

where $\theta_n^{(\ell)}$ is the angle between the images of $\mathbf{x}$ and $\mathbf{y}$ in the feature space induced by the ℓ-fold composition. In particular, we can write:

$$\theta_n^{(\ell)} \;=\; \cos^{-1}\left(\frac{k_n^{(\ell)}(\mathbf{x},\mathbf{y})}{\sqrt{k_n^{(\ell)}(\mathbf{x},\mathbf{x})\,k_n^{(\ell)}(\mathbf{y},\mathbf{y})}}\right). \tag{19}$$

The recursion in eq. (18) is simple to compute in practice. The resulting kernels mimic the computations in large multilayer threshold networks where the weights are Gaussian distributed with zero mean and unit variance. Above, for simplicity, we have assumed that the arc-cosine kernels have the same degree n at every level (or *layer*) ℓ of the recursion. We can also use kernels of different degrees at different layers. In the next section, we experiment with SVMs whose kernel functions are constructed in these ways.

3 Experiments

We evaluated SVMs with arc-cosine kernels on several medium-sized data sets. Our experiments had three goals: first, to study arc-cosine kernels of different order and depth; second, to understand their differences with more traditional RBF kernels; third, to compare large margin classification with arc-cosine kernels to deep learning in multilayer neural nets. We followed the same experimental methodology as previous authors (Larochelle et al., 2007) in an empirical study of SVMs, multilayer autoencoders, and deep belief nets. SVMs were trained using libSVM (version 2.88) (Chang & Lin, 2001), a publicly available software package. For each SVM, we held out part of the training data to choose the margin penalty parameter; after tuning this parameter on the held-out data, we then retrained each SVM on all the training data. We experimented on several different tasks in visual pattern recognition, which we present in their order of difficulty. To illustrate certain interesting trends (or the absence of such trends), our results compare test error rates obtained from a large number of different kernels. It should be emphasized, however, that one does not know a priori which kernel should be chosen for any given task. Therefore, for each data set, we also indicate which kernel was selected by its performance on held-out training examples (before retraining on all the training examples). The results from these properly selected kernels permit meaningful comparisons to previous benchmarks.

3.1 Handwritten digit recognition

We first evaluated SVMs with arc-cosine kernels on the MNIST data set of 28×28 grayscale handwritten digits (LeCun & Cortes, 1998). The MNIST data set has 60000 training examples and 10000 test examples of the digits [0-9]; examples are shown in Figure 3 (left). We used libSVM to train 45 SVMs, one on each pair of different digit classes, and labeled test examples by summing the votes from all 45 classifiers. For each SVM, we held out the last (roughly) 1000 training examples of each digit class to choose the margin-penalty parameter. The SVMs were trained and tested on deskewed 28x28 grayscale images.

We experimented with arc-cosine kernels of degree $n = 0, 1$ and 2, corresponding to threshold networks with "step", "ramp", and "quarter-pipe" activation functions. We also experimented with the multilayer kernels described in section 2.3, composed from one to six levels of recursion. Figure 3 shows the test set error rates from arc-cosine kernels of varying degrees (n) and numbers of layers (ℓ). We experimented first with multilayer kernels that used the same base kernel in each layer. In this case, we observed that the performance generally improved with increasing number of layers (up to $\ell = 6$) for the ramp ($n = 1$) kernel, but generally worsened[2] for the step ($n = 0$)

[2]Also, though not shown in the figure, the quarter-pipe kernel gave essentially random results, with roughly 90% error rates, when composed with itself three or more times.

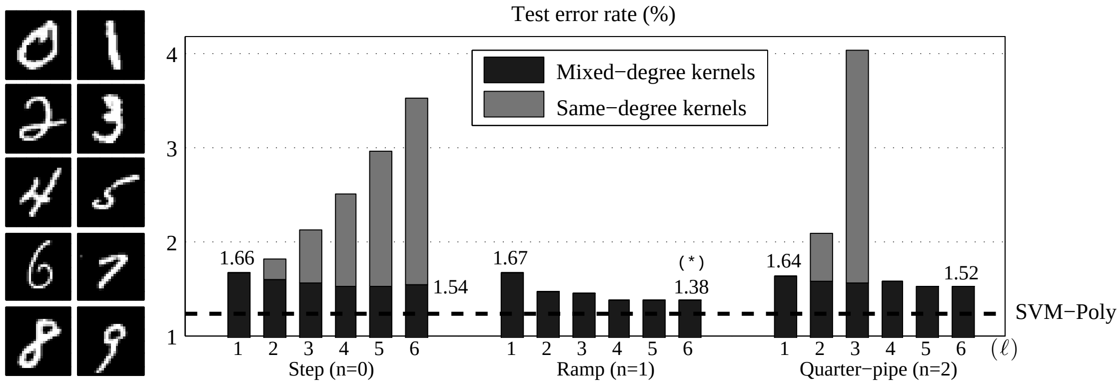

Figure 3: *Left*: examples from the MNIST data set. *Right*: classification error rates on the test set from SVMs with multilayer arc-cosine kernels. The figure shows results for kernels of varying degrees (n) and numbers of layers (ℓ). In one set of experiments, the multi-layer kernels were constructed by composing arc-cosine kernels of the same degree. In another set of experiments, only arc-cosine kernels of degree $n = 1$ were used at higher layers; the figure indicates the degree used at the first layer. The asterisk indicates the configuration that performed best on held-out training examples; precise error rates for some mixed-degree kernels are displayed for better comparison. The best comparable result is 1.22% from SVMs using polynomial kernels of degree 9 (Decoste & Schölkopf, 2002). See text for details.

and quarter-pipe ($n = 2$) kernels. These results led us to hypothesize that only $n = 1$ arc-cosine kernels preserve sufficient information about the magnitude of their inputs to work effectively in composition with other kernels. Recall from section 2.1 that only the $n = 1$ arc-cosine kernel preserves the norm of its inputs: the $n = 0$ kernel maps inputs to the unit hypersphere in feature space, while higher-order ($n > 1$) kernels distort input magnitudes in the same way as polynomial kernels.

We tested this hypothesis by experimenting with "mixed-degree" multilayer kernels which used arc-cosine kernels of degree $n = 1$ at all higher levels ($\ell > 1$) of the recursion in eqs. (18–19). Figure 3 shows these sets of results in a darker shade of gray. In these experiments, we used arc-cosine kernels of different degrees at the first layer of nonlinearity, but only the ramp ($n = 1$) kernel at successive layers. The best of these kernels yielded a test error rate of 1.38%, comparable to many other results from SVMs (LeCun & Cortes, 1998) on deslanted MNIST digits, though not matching the best previous result of 1.22% (Decoste & Schölkopf, 2002). These results also reveal

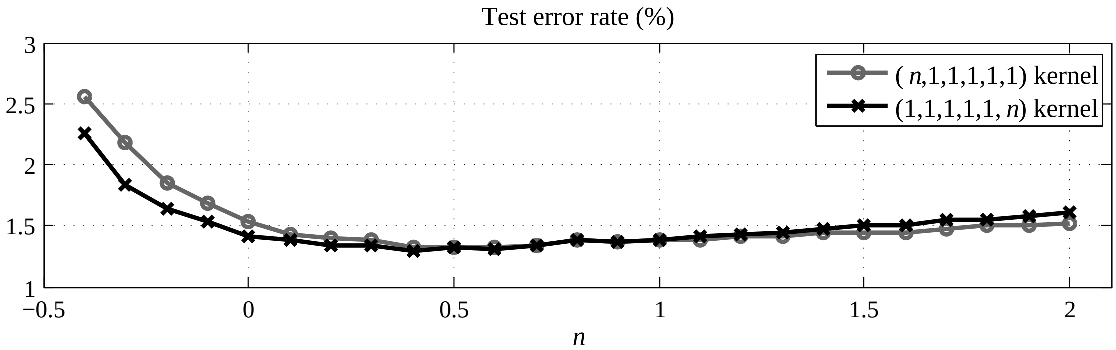

Figure 4: Classification error rates on the MNIST data set using six-layer arc-cosine kernels with fractional degrees. The SVMs in these experiments used mixed-degree kernels in which the kernel in the first or final layer had the continuously varying degree n shown on the x-axis. The other layers used arc-cosine kernels of degree $n = 1$.

that multilayer kernels yield different results than their single layer counterparts. (With increasing depth, there is also a slight but suggestive trend toward improved results.) Though SVMs are inherently shallow architectures, this variability is reminiscent of experience with multilayer neural nets. We explore the effects of multilayer kernels further in sections 3.2–3.3, experimenting on problems that were specifically designed to illustrate the advantages of deep architectures.

We also experimented briefly with fractional and negative values of the degree n in multilayer arc-cosine kernels. The kernel functions in these experiments had to be computed numerically, as described in the appendix. Figure 4 shows the test error rates from SVMs in which we continuously varied the degree of the arc-cosine kernel used in the first or final layer. (The other layers used arc-cosine kernels of degree $n = 1$.) The best results in these experiments were obtained by using kernels with fractional degrees; however, the improvements were relatively modest. In subsequent sections, we only report results using arc-cosine kernels of degree $n \in \{0, 1, 2\}$.

3.2 Shape classification

We experimented next on two data sets for binary classification that were designed to illustrate the advantages of deep architectures (Larochelle et al., 2007). The examples in these data sets are also 28×28 grayscale pixel images. In the first data set, known as *rectangles-image*, each image contains an occluding rectangle, and the task is to

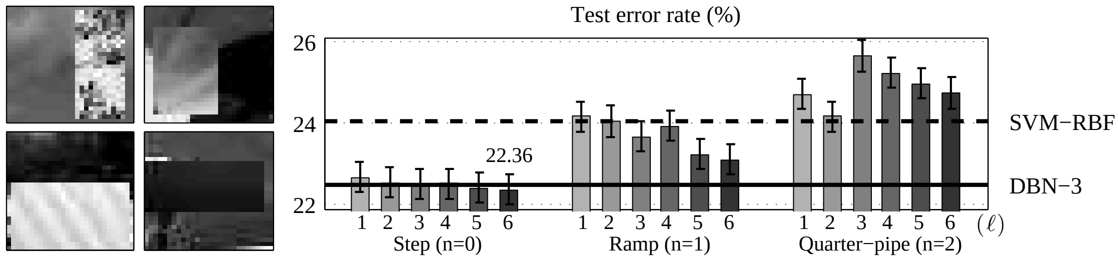

Figure 5: *Left*: examples from the *rectangles-image* data set. *Right*: classification error rates on the test set. SVMs with arc-cosine kernels have error rates from 22.36–25.64%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 24.04% for SVMs with RBF kernels and 22.50% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed; the error bars correspond to 95% confidence intervals. See text for details.

determine whether the width of the rectangle exceeds its height; examples are shown in Figure 5 (left). In the second data set, known as *convex*, each image contains a white region, and the task is to determine whether the white region is convex; examples are shown in Figure 6 (left). The *rectangles-image* data set has 12000 training examples, while the *convex* data set has 8000 training examples; both data sets have 50000 test examples. We held out the last 2000 training examples of each data set to select the margin-penalty parameters of SVMs.

The classes in these tasks involve abstract, shape-based features that cannot be computed directly from raw pixel inputs, but rather seem to require many layers of processing. The positive and negative examples also exhibit tremendous variability, making these problems difficult for template-based approaches (e.g., SVMs with RBF kernels). In previous benchmarks on binary classification (Larochelle et al., 2007), these problems exhibited the biggest performance gap between deep architectures (e.g., deep belief nets) and traditional SVMs. We experimented to see whether this gap could be reduced or even reversed by the use of arc-cosine kernels.

Figures 5 and 6 show the test set error rates from SVMs with mixed-degree, multilayer arc-cosine kernels. For these experiments, we used arc-cosine kernels of degree $n = 1$ in all but the first layer. The figures also show the best previous results from SVMs with RBF kernels and three-layer deep belief nets (Larochelle et al., 2007).

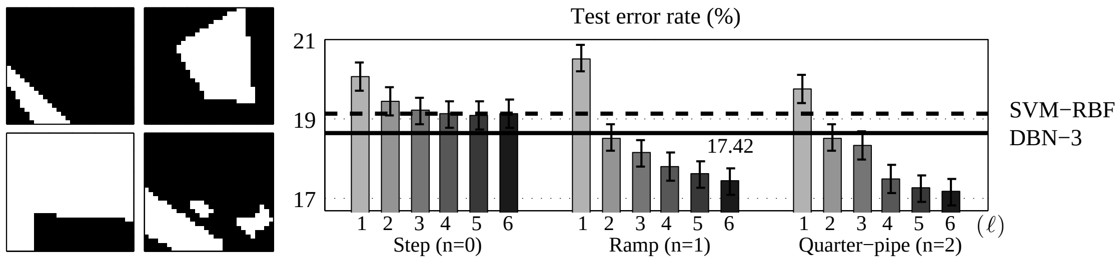

Figure 6: *Left*: examples from the *convex* data set. *Right*: classification error rates on the test set. SVMs with arc-cosine kernels have error rates from 17.15–20.51%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 19.13% for SVMs with RBF kernels and 18.63% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed; the error bars correspond to 95% confidence intervals. See text for details.

Overall, the figures show that many SVMs with arc-cosine kernels outperform SVMs with RBF kernels, and a certain number also outperform deep belief nets. The results on the *rectangles-image* data set show that the $n = 0$ kernel seems particularly well suited to this task. The results on the *convex* data set show a pronounced trend in which increasing numbers of layers leads to lower error rates.

Beyond these improvements in performance, we also note that SVMs with arc-cosine kernels are quite straightforward to train. Unlike SVMs with RBF kernels, they do not require tuning a kernel width parameter, and unlike deep belief nets, they do not require solving a difficult nonlinear optimization or searching over many possible architectures. Thus, as a purely practical matter, SVMs with arc-cosine kernels are very well suited for medium-sized problems in binary classification.

We were curious if SVMs with arc-cosine kernels were discovering sparse solutions to the above problems. To investigate this possibility, we compared the numbers of support vectors used by SVMs with arc-cosine and RBF kernels. The best SVMs with arc-cosine kernels used 9600 support vectors for the *rectangles-image* data set and 5094 support vectors for the *convex* data set. These numbers are slightly lower (by several hundred) than the numbers for RBF kernels; however, they still represent a sizable fraction of the training examples for these data sets.

3.3 Noisy handwritten digit recognition

We experimented last on two challenging data sets in multiway classification. Like the data sets in the previous section, these data sets were also designed to illustrate the advantages of deep architectures (Larochelle et al., 2007). They were created by adding different types of background noise to the MNIST images of handwritten digits. Specifically, the *mnist-back-rand* data set was generated by filling image backgrounds with white noise, while the *mnist-back-image* data set was generated by filling image backgrounds with random image patches; examples are shown in Figures 7 and 8. Each data set contains 12000 training examples and 50000 test examples. We initially held out the last 2000 training examples of each data set to select the margin-penalty parameters of SVMs.

The right panels of Figures 7 and 8 show the test set error rates of SVMs with multilayer arc-cosine kernels. For these experiments, we again used arc-cosine kernels of degree $n = 1$ kernels in all but the first layer. Though we observed that arc-cosine kernels with more layers often led to better performance, the trend was not as pronounced as in previous sections. Moreover, on these data sets, SVMs with arc-cosine kernels performed slightly worse than the best SVMs with RBF kernels and significantly worse than the best deep belief nets.

On this problem, it is not too surprising that SVMs fare much worse than deep belief nets. Both arc-cosine and RBF kernels are rotationally invariant, and such kernels are not well-equipped to deal with large numbers of noisy and/or irrelevant features (Ng, 2004). Presumably, deep belief nets perform better because they can learn to directly suppress noisy pixels. Though SVMs perform poorly on these tasks, more recently we have obtained positive results using arc-cosine kernels in a different kernel-based architecture (Cho & Saul, 2009). These positive results were obtained by combining ideas from kernel PCA (Schölkopf et al., 1998), discriminative feature selection, and large margin nearest neighbor classification (Weinberger & Saul, 2009). The rotational invariance of arc-cosine kernel could also be explicitly broken by integrating in eq. (1) over a multivariate Gaussian distribution with a general (non-identity) covariance matrix. By choosing the covariance matrix appropriately (or perhaps by learning it), we could potentially tune arc-cosine kernels to suppress irrelevant features in the inputs.

It is less clear why SVMs with arc-cosine kernels perform worse on these data sets

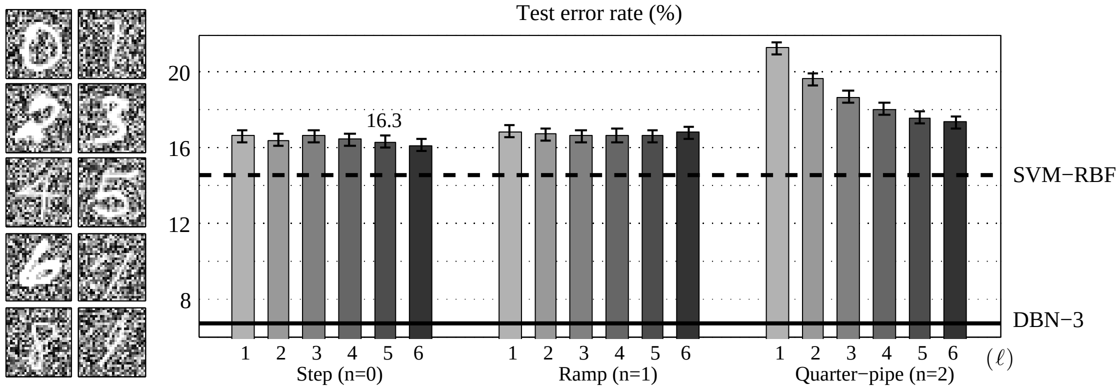

Figure 7: *Left*: examples from the *mnist-back-rand* data set. *Right*: classification error rates on the test set. SVMs with arc-cosine kernels have error rates from 16.14–21.27%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 14.58% for SVMs with RBF kernels and 6.73% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed; the error bars correspond to 95% confidence intervals. See text for details.

than SVMs with RBF kernels. Our results in multiway classification were obtained by a naive combination of binary SVM classifiers, which may be a confounding effect. We also examined the SVM error rates on the 45 sub-tasks of one-against-one classification. Here we observed that on average, the SVMs with arc-cosine kernels performed slightly worse (from 0.06–0.67%) than the SVMs with RBF kernels, but not as much as the differences in Figures 7 and 8 might suggest. In future work, we may explore arc-cosine kernels in a more principled framework for multiclass SVMs (Crammer & Singer, 2001).

4 Conclusion

In this paper, we have explored a new family of positive-definite kernels for large margin classification in SVMs. The feature spaces induced by these kernels mimic the internal representations stored by the hidden layers of large neural networks. Evaluating these kernels in SVMs, we found that on certain problems they led to state-of-the-art results. Interestingly, they also exhibited trends that seemed to reflect the benefits of learning in deep architectures.

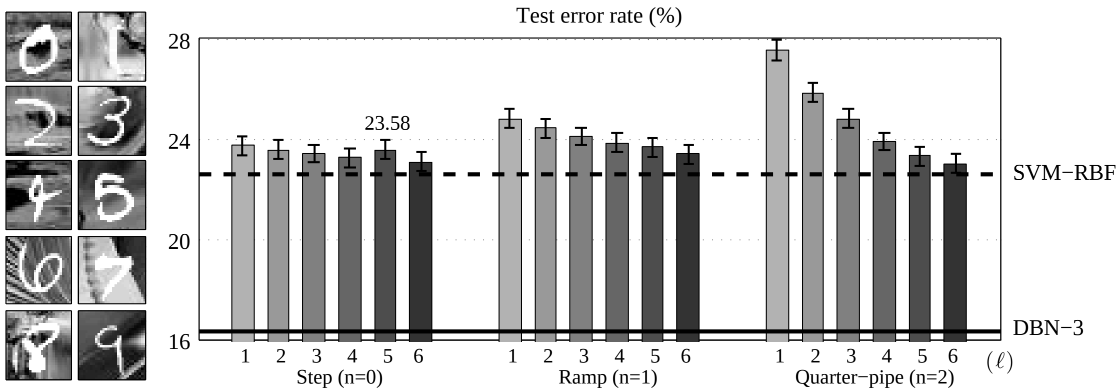

Figure 8: *Left*: examples from the *mnist-back-image* data set. *Right*: classification error rates on the test set. SVMs with arc-cosine kernels have error rates from 23.03–27.58%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 22.61% for SVMs with RBF kernels and 16.31% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed; the error bars correspond to 95% confidence intervals. See text for details.

Our approach in this paper builds on several lines of related work by previous authors. Computation in infinite neural networks was first studied in the context of Gaussian processes (Neal, 1996). In later work, Williams (1998) derived analytical forms for kernel functions that mimicked the computation in large networks. In networks with sigmoidal activation functions, these earlier results reduce as a special case to eq. (1) for the arc-cosine kernel of degree $n = 0$ as shown in section 2.2. Our contribution to this line of work has been to enlarge the family of kernels which can be computed analytically. In particular, we have derived kernels to mimic the computation in large networks with ramp ($n\!=\!1$), quarter-pipe ($n\!=\!2$), and higher-order activation functions. Exploring the use of these kernels for large margin classification, we often found that the best results were obtained by arc-cosine kernels with $n\!>\!0$.

More recently, other researchers have explored connections between kernel machines and neural computation. Bengio et al. (2006) showed how to formulate the training of multilayer neural networks as a problem in convex optimization; though the problem involves an infinite number of variables, corresponding to all possible hidden units, finite solutions are obtained by regularizing the weights at the output layer and incrementally inserting one hidden unit at a time. Related ideas were also previously considered by Lee et al. (1996) and Zhang (2003). Rahimi & Recht (2009) studied networks that pass inputs through a large bank of arbitrary randomized nonlinearities, then compute weighted sums of the resulting features; these networks can be designed to mimic the computation in kernel machines, while scaling much better to large data sets. For the most part, these previous studies have exploited the connection between kernel machines and neural networks with one layer of hidden units. Our contribution to this line of work has been to develop a more direct connection between kernel machines and multilayer neural networks. This connection was explored through the recursive construction of multilayer kernels in section 2.3.

Our work was also motivated by the ongoing debate over deep versus shallow architectures (Bengio & LeCun, 2007). Motivated by similar issues, Weston et al. (2008) suggested that kernel methods could play a useful role in deep learning; specifically, these authors showed that shallow architectures for nonlinear embedding could be used to define auxiliary tasks for the hidden layers of deep architectures. Our results suggest another way that kernel methods may play a useful role in deep learning—by mimicking directly the computation in large, multilayer neural networks.

It remains to be understood why certain problems are better suited for arc-cosine kernels than others. We do not have definitive answers to this question. Perhaps the $n = 0$ arc-cosine kernel works well on the *rectangles-image* data set (see Figure 5) because it normalizes for brightness, only considering the angle between two images. Likewise, perhaps the improvement from multilayer kernels on the *convex* data set (see Figure 6) shows that this problem indeed requires a hierarchical solution, just as its creators intended. We hope to improve our understanding of these effects as we obtain more experience with arc-cosine kernels on a greater variety of data sets.

Encouraged especially by the results in section 3.2, we are pursuing several directions for future research. We are currently exploring the use of arc-cosine kernels in other types of kernel machines besides SVMs, including architectures for unsupervised and semi-supervised learning (Cho & Saul, 2009). We also believe that methods for multiple kernel learning (Lanckriet et al., 2004; Rakotomamonjy et al., 2008; Bach, 2009) may be useful for combining arc-cosine kernels of different degrees and depth. We hope that researchers in Gaussian processes will explore arc-cosine kernels and use them to further develop the connections to computation in multilayer neural networks (Neal, 1996). Finally, an open question is how to incorporate prior knowledge, such as the invariances modeled by convolutional neural networks (LeCun et al., 1989; Ranzato et al., 2007), into the types of kernels studied in this paper. These issues and others are left for future work.

Acknowledgments

This work was supported by award number 0957560 from the National Science Foundation. The authors are tremendously grateful to the reviewers, whose many knowledgeable comments and thoughtful suggestions helped to improve all parts of the paper. The authors also benefited from many personal discussions at a Gatsby Unit Workshop on Deep Learning.

A Derivation of kernel function

In this appendix, we show how to evaluate the multidimensional integral in eq. (1) for the arc-cosine kernel. We begin by reducing it to a one-dimensional integral. Let θ denote the angle between the inputs $\mathbf{x}$ and $\mathbf{y}$. Without loss of generality, we can take the w_1-axis to align with the input $\mathbf{x}$ and the w_1w_2-plane to contain the input $\mathbf{y}$. Integrating out the orthogonal coordinates of the weight vector $\mathbf{w}$, we obtain the result in eq. (3) where $J_n(\theta)$ is the remaining integral:

$$J_n(\theta) \;=\; \int\! dw_1\, dw_2\, e^{-\frac{1}{2}(w_1^2+w_2^2)}\, \Theta(w_1)\, \Theta(w_1\cos\theta\!+\!w_2\sin\theta)\, w_1^n\, (w_1\cos\theta\!+\!w_2\sin\theta)^n. \tag{20}$$

Changing variables to $u=w_1$ and $v=w_1\cos\theta\!+\!w_2\sin\theta$, we simplify the domain of integration to the first quadrant of the uv-plane:

$$J_n(\theta) \;=\; \frac{1}{\sin\theta}\int_0^\infty\! du \int_0^\infty\! dv\; e^{-(u^2+v^2-2uv\cos\theta)/(2\sin^2\theta)}\; u^n v^n. \tag{21}$$

The prefactor of $(\sin\theta)^{-1}$ in eq. (21) is due to the Jacobian. We reduce the two dimensional integral in eq. (21) to a one dimensional integral by adopting polar coordinates: $u\!=\!r\cos\phi$ and $v\!=\!r\sin\phi$. The integral over the radius coordinate r is straightforward, yielding:

$$J_n(\theta) = n!\,(\sin\theta)^{2n+1}\int_0^{\frac{\pi}{2}}\! d\phi\; \frac{\sin^n 2\phi}{(1-\cos\theta\sin 2\phi)^{n+1}}. \tag{22}$$

Finally, to convert this integral into a more standard form, we make the simple change of variables $\psi = 2\phi - \frac{\pi}{2}$. The resulting integral is given by:

$$J_n(\theta) = n!\,(\sin\theta)^{2n+1}\int_0^{\frac{\pi}{2}} d\psi\,\frac{\cos^n\psi}{(1-\cos\theta\cos\psi)^{n+1}}. \tag{23}$$

Note the dependence of this final one-dimensional integral on the degree n of the arc-cosine kernel function. As we show next, this integral can be evaluated analytically for all $n \in \{0, 1, 2, \ldots\}$.

We first evaluate eq. (23) for the special case $n = 0$. The following result can be derived by contour integration in the complex plane (Carrier et al., 2005):

$$\int_0^{\xi} \frac{d\psi}{1-\cos\theta\cos\psi} \;=\; \frac{1}{\sin\theta}\,\tan^{-1}\!\left(\frac{\sin\theta\,\sin\xi}{\cos\xi-\cos\theta}\right), \tag{24}$$

The integral in eq. (24) can also be verified directly by differentiating the right hand side. Evaluating the above result at $\xi = \frac{\pi}{2}$ gives:

$$\int_0^{\pi/2} \frac{d\psi}{1-\cos\theta\cos\psi} \;=\; \frac{\pi-\theta}{\sin\theta}. \tag{25}$$

Substituting eq. (25) into our expression for the angular part of the kernel function in eq. (23), we recover our earlier claim that $J_0(\theta) = \pi - \theta$. Related integrals for the special case $n = 0$ can also be found in earlier work (Williams, 1998; Watkin et al., 1993).

Next we show how to evaluate the integrals in eq. (23) for higher order kernel functions. For integer $n > 0$, the required integrals can be obtained by the method of differentiating under the integral sign. In particular, we note that:

$$\int_0^{\frac{\pi}{2}} d\psi\,\frac{\cos^n\psi}{(1-\cos\theta\cos\psi)^{n+1}} \;=\; \frac{1}{n!}\,\frac{\partial^n}{\partial(\cos\theta)^n}\int_0^{\pi/2}\frac{d\psi}{1-\cos\theta\cos\psi}. \tag{26}$$

Substituting eq. (26) into eq. (23), then appealing to the previous result in eq. (25), we recover the expression for $J_n(\theta)$ as stated in eq. (4).

Finally, we consider the general case where the degree n of the arc-cosine kernel is real-valued. For real-valued n, the required integral in eq. (23) does not have a simple analytical form. However, we can evaluate the original representation in eq. (1) for the special case of equal inputs $\mathbf{x} = \mathbf{y}$. In this case, without loss of generality, we can again take the w_1-axis to align with the input $\mathbf{x}$. Integrating out the orthogonal coordinates of the weight vector $\mathbf{w}$, we obtain:

$$k_n(\mathbf{x},\mathbf{x}) \;=\; \sqrt{\frac{2}{\pi}}\,\|\mathbf{x}\|^{2n}\int_0^\infty dw_1\, e^{-\frac{1}{2}w_1^2}\, w_1^{2n} \;=\; \frac{2^n}{\sqrt{\pi}}\,\Gamma\left(n+\frac{1}{2}\right)\|\mathbf{x}\|^{2n}. \tag{27}$$

The gamma function on the right hand side of eq. (27) diverges as its argument approaches zero; thus $k_n(\mathbf{x},\mathbf{x})$ diverges as $n\rightarrow -\frac{1}{2}$ for all inputs $\mathbf{x}$. This divergence shows that the integral representation of the arc-cosine kernel in eq. (1) is only defined for $n > -\frac{1}{2}$.

Though the arc-cosine kernel does not have a simple form for real-valued n, the intermediate results in eqs. (3) and (23) remain generally valid. Thus, for non-integer n, the integral representation in eq. (1) can still be reduced to a one-dimensional integral for $J_n(\theta)$, where θ is the angle between the inputs $\mathbf{x}$ and $\mathbf{y}$. For $\theta > 0$, it is straightforward to evaluate the integral for $J_n(\theta)$ in eq. (23) by numerical methods. This was done for the experiments in section 3.1.

References

Bach, F. (2009). Exploring large feature spaces with hierarchical multiple kernel learning. In Koller, D., Schuurmans, D., Bengio, Y., & Bottou, L. (Eds.), *Advances in Neural Information Processing Systems 21*, (pp. 105–112)., Cambridge, MA. MIT Press.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *to appear*.

Bengio, Y. & LeCun, Y. (2007). *Scaling learning algorithms towards AI.* MIT Press.

Bengio, Y., Roux, N. L., Vincent, P., Delalleau, O., & Marcotte, P. (2006). Convex neural networks. In Weiss, Y., Schölkopf, B., & Platt, J. (Eds.), *Advances in Neural Information Processing Systems 18*, (pp. 123–130)., Cambridge, MA. MIT Press.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, (pp. 144–152). ACM Press.

Carrier, G. F., Krook, M., & Pearson, C. E. (2005). *Functions of a Complex Variable: Theory and Technique*. Society for Industrial and Applied Mathematics.

Chang, C.-C. & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Cho, Y. & Saul, L. K. (2009). Kernel methods for deep learning. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., & Culotta, A. (Eds.), *Advances in Neural Information Processing Systems 22*, (pp. 342–350)., Cambridge, MA. MIT Press.

Collobert, R. & Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, (pp. 160–167).

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Crammer, K. & Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, *2*, 265–292.

Cristianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Decoste, D. & Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, *46*(1-3), 161–190.

Hahnloser, R. H. R., Seung, H. S., & Slotine, J. J. (2003). Permitted and forbidden sets in symmetric threshold-linear networks. *Neural Computation*, *15*(3), 621–638.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.

Hinton, G. E. & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L. E., & Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, *5*, 27–72.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning (ICML-07)*, (pp. 473–480).

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

LeCun, Y. & Cortes, C. (1998). The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

Lee, W. S., Bartlett, P., & Williamson, R. (1996). Efficient agnostic learning of neural networks with bounded fan-in. *IEEE Transactions on Information Theory*, *42*(6), 2118–2132.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc.

Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the 21st International Conference on Machine Learning (ICML-04)*, (pp. 78–85).

Price, R. (1958). A useful theorem for nonlinear devices having Gaussian inputs. *IRE Transactions on Information Theory*, *4*(2), 69–72.

Rahimi, A. & Recht, B. (2009). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In Koller, D., Schuurmans, D., Bengio, Y., & Bottou, L. (Eds.), *Advances in Neural Information Processing Systems 21*, (pp. 1313–1320)., Cambridge, MA. MIT Press.

Rakotomamonjy, A., Bach, F. R., Canu, S., & Grandvalet, Y. (2008). SimpleMKL. *Journal of Machine Learning Research*, *9*, 2491–2521.

Ranzato, M. A., Huang, F. J., Boureau, Y. L., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Pro-*

ceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-07), (pp. 1–8).

Schölkopf, B., Smola, A., & Müller, K. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*(5), 1299–1319.

Schölkopf, B. & Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. The MIT Press.

Schölkopf, B., Smola, A. J., & Müller, K.-R. (1996). Nonlinear component analysis as a kernel eigenvalue problem. Technical Report 44, Max-Planck-Institut für biologische Kybernetik.

Watkin, T. H. L., Rau, A., & Biehl, M. (1993). The statistical mechanics of learning a rule. *Reviews of Modern Physics*, *65*(2), 499–556.

Weinberger, K. Q. & Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, *10*, 207–244.

Weston, J., Ratle, F., & Collobert, R. (2008). Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, (pp. 1168–1175).

Williams, C. K. I. (1998). Computation with infinite neural networks. *Neural Computation*, *10*(5), 1203–1216.

Zhang, T. (2003). Sequential greedy approximation for certain convex optimization problems. *IEEE Transactions on Information Theory*, *49*(3), 682–691.