UNIVERSITY OF CALIFORNIA, SAN DIEGO

Kernel Methods for Deep Learning

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Youngmin Cho

Committee in charge:

Professor Lawrence Saul, Chair
Professor Garrison Cottrell
Professor Sanjoy Dasgupta
Professor Gert Lanckriet
Professor Nuno Vasconcelos

2012

The dissertation of Youngmin Cho is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2012

DEDICATION

To my loving wife, Nuree.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I would like to express my deep gratitude to my advisor Prof. Lawrence Saul. Without his amazing guidance and support, I could not come this far. I also appreciate the suggestions and comments from my committee members: Prof. Garrison Cottrell, Prof. Sanjoy Dasgupta, Prof. Gert Lanckriet, and Prof. Nuno Vasconcelos. They helped me to improve the quality of this thesis.

I am very grateful to Dr. Andrew Senior and Dr. Jason Weston for giving me the opportunity to work with them as a summer intern at Google. Special gratitude is also extended to the Samsung Scholarship Foundation for the financial support during my doctoral study. Many thanks are also due to my lab mates and friends for sharing ideas and concerns.

I am deeply indebted to my parents and my family for their continued support. Finally, I would like to offer my sincere thanks to my wife Nuree and my daughters Seren and Dana for giving me the strength to pursue my dream.

Parts of this thesis are reprints of the papers coauthored with Lawrence Saul. Chapters 1, 2, and 4 are based on "Large margin classification in infinite neural networks" by Cho, Y. and Saul, L. K. as it appears in Cho and Saul (2010). Chapters 1, 2, 4, and 5 is based on "Kernel methods for deep learning" by Cho, Y. and Saul, L. K. as it appears in Cho and Saul (2009). Chapters 1, 3, and 6 are based on "Analysis and extension of arc-cosine kernels for large margin classification" by Cho, Y. and Saul, L. K. as in appears in Cho and Saul (2012). The thesis author was the primary investigator and author of these work.

VITA

2001	B. S. in Computer Science and Engineering *summa cum laude*, Seoul National University, Republic of Korea
2012	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

Cho, Y. and Saul, L. K. (2012). Analysis and extension of arc-cosine kernels for large margin classification. Technical Report CS2012-0972, Department of Computer Science and Engineering, University of California, San Diego.

Senior, A., Cho, Y., and Weston, J. (2012). Learning improved linear transforms for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-12)*, to appear.

Cho, Y. and Saul, L. K. (2010). Large margin classification in infinite neural networks. *Neural Computation*, 22(10):2678–2697.

Hu, D. J., van der Maaten, L., Cho, Y., Saul, L. K., and Lerner, S. (2010). Latent variable models for predicting file dependencies in large-scale software development. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 865–873.

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

Cho, Y. and Saul, L. K. (2009). Learning dictionaries of stable autoregressive models for audio scene analysis. In *Proceedings of the Twenty Sixth International Conference on Machine Learning (ICML-09)*, pages 169–176.

Cho, Y. and Saul, L. K. (2009). Sparse decomposition of mixed audio signals by basis pursuit with autoregressive models. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-09)*, pages 1705–1708.

ABSTRACT OF THE DISSERTATION

Kernel Methods for Deep Learning

by

Youngmin Cho

Doctor of Philosophy in Computer Science

University of California, San Diego, 2012

Professor Lawrence Saul, Chair

We introduce a new family of positive-definite kernels that mimic the computation in large neural networks. We derive the different members of this family by considering neural networks with different activation functions. Using these kernels as building blocks, we also show how to construct other positive-definite kernels by operations such as composition, multiplication, and averaging. We explore the use of these kernels in standard models of supervised learning, such as support vector machines for large margin classification, as well as in new models of unsupervised learning based on deep architectures. On several problems, we obtain better results than previous, leading benchmarks from both support vector machines with Gaussian kernels as well as deep belief nets. Finally, we examine

the properties of these kernels by analyzing the geometry of surfaces that they induce in Hilbert space.

Chapter 1

Introduction

Machine learning is a field of study that aims to design and implement algorithms that enable machines to learn from examples. The resulting algorithms allow us to recognize patterns in the data, extract knowledge from them, and make predictions for new, unseen examples. Given the limited number of training examples and also restricted computational resources, the key issue in machine learning is how to cope with complex variabilities in the data efficiently.

Machine learning has been applied to various areas including, but never limited to: object recognition (identification of patterns that exclusively belong to a particular class of images), information extraction (extraction of structured knowledge from unstructured data such as documents on the Internet), recommendation systems (prediction of preference scores for unrated items for a particular user given the user profile and item properties), and so on.

Typically, machine learning models contain parameters to fit the data and thus they are often formulated as parameter optimization problems. This model fitting has two scenarios that should be avoided—overfitting and underfitting. Overfitting occurs when the training data are not enough but the learning models are unnecessarily complex. As a result, the models even fit to irrelevant details of training examples and fail to generalize beyond them. In contrast, underfitting occurs when the models do not have enough capacities to cover the variabilities in the examples and again, fail to generalize to test data. Hence, to strike a balance between these two extremes is an important challenge for machine learning.

1.1 Supervised Learning

Supervised learning is a machine learning scheme where statistical models are trained with input examples and their target labels. In supervised learning, we are typically provided with a training set of N examples $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ where $\mathbf{x}_n \in \mathbb{R}^d$ is the nth data point and y_n is its label which is either a real number (regression) or a categorical value (classification). The goal of supervised learning is to learn a statistical model from labeled examples so that it can generalize to unseen examples and predict their labels correctly.

Linear Models (and Their Limitations)

A simple yet effective way to achieve this goal is to use linear models. Consider the task of spam detection that can be formulated as a binary classification problem (spam versus regular). In linear classification models, we predict the label of an example $\mathbf{x} \in \mathbb{R}^d$ by first multiplying the weight w_i to each dimension x_i of the input, summing d such factors, and finally testing its sign:

$$t = \text{sign}\left(\sum_{i=1}^{d} w_i x_i\right), \tag{1.1}$$

where the positive sign means spam. The linear weights $\mathbf{w} = (w_1, w_2, \ldots, w_d)$ correspond to the parameters of the model that we learn from training data. For the case of classification, $\mathbf{w}$ defines a linear decision hyperplane.

Though linear models are efficient and also easy to analyze, they have serious limitations in practice (Bishop, 2006). Particularly, many real data sets reside in nonlinear spaces. For the example of binary classification, we cannot expect linear decision boundaries to separate two classes without much error in such spaces. It is possible to expand the input space by adding nonlinear features $\mathbf{\Phi}(\mathbf{x})$ explicitly and to learn the linear weights for the resulting representation; however, this often increases the input dimensionality too much, leading to a problematic situation known as the curse of dimensionality (Bellman, 1961).

More specifically, this phenomenon means we will need exponentially more

data points to cover the input space even when its dimensionality increases linearly. As a result, supervised learning algorithms have more risk of overfitting to (comparatively) small number of examples in high-dimensional space. For instance, there exist many linear decision boundaries for perfect classification of training examples when the input dimensionality exceeds the number of training examples; however, all of these boundaries do not necessarily generalize well to unseen examples. Another problem with such increased dimensionality is the cost of learning and prediction—it takes more computational resources to process the data.

There have been various efforts to overcome such limitations of linear models. A support vector machine (SVM) is a linear classifier that avoids overfitting in high-dimensional space by a *robust* decision hyperplane (Boser et al., 1992; Cortes and Vapnik, 1995; Schölkopf and Smola, 2001). It relies on the concept of "margin" between the binary classes and seeks a decision boundary that maximizes it (more details follow later). Interestingly, analyses from computational learning theory prove that margin maximization is an effective way to minimize the risk of overfitting (Vapnik, 1998). Besides such desirable property, SVMs are easily extended to nonlinear models by implicitly incorporating nonlinear features into data representation through kernels.

Neural networks (or deep learning) are another interesting learning approaches to address the problems of linear models. They transform inputs through multiple layers of nonlinear processing and most importantly, such nonlinearity is in parametric forms so that they can be adapted to each data sets through training. This is in contrast to other generalized linear models (including SVMs) that obtain nonlinear features through fixed operations such as basis expansion or kernels. Due to such adaptability, neural networks can focus only on regions of input space where important variations happen and thus provide compact nonlinear models without unnecessary feature expansion. (But this is achieved at the cost of more difficult training procedures.)

Finally, an orthogonal approach to these (non-)linear supervised methods is unsupervised learning. In unsupervised learning, we focus on capturing important patterns from data regardless of their labels. It is usually carried out in the form

of dimensionality reduction or feature extraction, and these methods are beneficial to supervised models in general. Particularly, unsupervised learning reduces the input dimensionality of data without losing crucial information and alleviates the curse of dimensionality. It also regularizes supervised models to concentrate their modeling capacities on relatively small regions of input space without overfitting to noisy details. Since it is much easier to acquire unlabeled examples compared with labeled ones, unsupervised learning will be even more useful in practice.

In this thesis, we combine the ideas from this discussion and propose a novel nonlinear supervised model. We first begin with reviewing kernel methods and deep architectures as follows.

Kernel Methods

Kernel methods provide a powerful framework for pattern analysis and classification (Boser et al., 1992; Cortes and Vapnik, 1995; Schölkopf and Smola, 2001). Intuitively, the so-called "kernel trick" works by mapping inputs $\mathbf{x}$ into a nonlinear, potentially infinite-dimensional feature space $\mathbf{\Phi}(\mathbf{x})$ (Aizerman et al., 1964), then applying classical linear methods in this space. The mapping is induced by a kernel function $k(\cdot, \cdot)$ that operates on pairs of inputs and computes a generalized inner product:

$$k(\mathbf{x}, \mathbf{y}) = \mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{y}). \tag{1.2}$$

Typically, the kernel function measures some highly nonlinear 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 and Vapnik, 1995; Cristianini and Shawe-Taylor, 2000). The SVM is a linear classifier that predicts binary labels of input examples based on projection of the examples into a decision hyperplane. In particular, we seek the decision boundary where the margin between the binary classes is maximized. Figure 1.1 illustrates the main idea.

More concretely, consider a binary classification data set of N examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where $\mathbf{x}_i \in \mathbb{R}^d$ is the ith data point and $y_i \in \{-1, 1\}$ is its label.

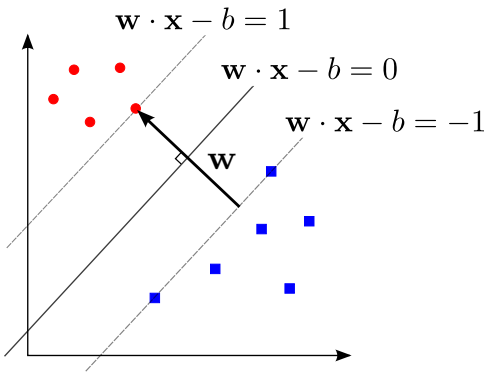


Figure 1.1: A decision hyperplane $\mathbf{w} \cdot \mathbf{x} - b = 0$ from a support vector machines for binary classification (red circles versus blue squares). We optimize $\mathbf{w}$ and b to maximize the distance $\frac{2}{\|\mathbf{w}\|}$ between the two dotted hyperplanes.

Then, the linear decision boundary for an input example $\mathbf{x}$ is described as

$$\mathbf{w} \cdot \mathbf{x} - b = 0, \tag{1.3}$$

where $\mathbf{w}$ is the weight vector and b is the offset. In SVMs, we require the decision hyperplane to separate the positively and negatively labeled examples by the largest possible margin. Particularly, we formulate this idea as new constraints on the hyperplane:

$$y_i \left(\mathbf{w} \cdot \mathbf{x}_i - b\right) \geq 1, \tag{1.4}$$

which means positive ($y_i = 1$) and negative ($y_i = -1$) examples should be separated further from the decision boundary[1]. Figure 1.1 explains this idea for examples in $\mathbb{R}^2$, where positive data points (red circles) are located above the hyperplane $\mathbf{w} \cdot \mathbf{x} - b = 1$ and negative data points (blue squares) are located below the hyperplane $\mathbf{w} \cdot \mathbf{x} - b = -1$.

For large margin classification, we attempt to maximize the distance $\frac{2}{\|\mathbf{w}\|}$ between these two hyperplanes, which is eventually formulated as the following

[1]In practice, there may not exist a linear hyperplane that perfectly separates the binary classes without errors. Soft margin support vector machines address this issue by introducing a slack variable $\xi_i \geq 0$ to relax each constraint: $y_i \left(\mathbf{w} \cdot \mathbf{x}_i - b\right) \geq 1 - \xi_i$ (Cortes and Vapnik, 1995). We use soft margin SVMs for experiments in this thesis.

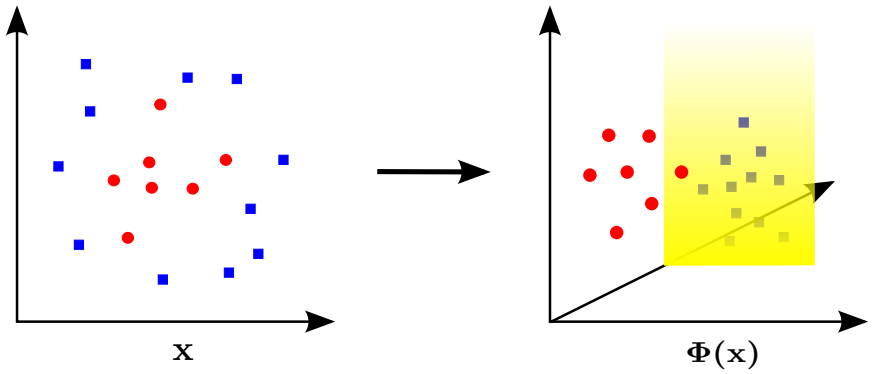


Figure 1.2: Classification in support vector machines with nonlinear kernels. *Left*: it is impossible to find a linear hyperplane separating input data space into two classes (red circles versus blue squares). *Right*: it is easy to find a decision boundary (the yellow hyperplane) in high dimensional feature space after the mapping $\mathbf{\Phi}(\cdot)$ induced by nonlinear kernels.

constrained minimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i\left(\mathbf{w}\cdot\mathbf{x}_i - b\right) \geq 1 \quad \forall i \in \{1,2,\ldots,N\} \end{aligned} \tag{1.5}$$

This convex optimization problem can be solved efficiently (Fan et al., 2008), but we are interested in the dual form of the problem as well:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \ \mathbf{x}_i\cdot\mathbf{x}_j \\ \text{subject to} \quad & \alpha_i \geq 0 \quad \forall i \in \{1,2,\ldots,N\}, \\ & \sum_{i=1}^{N}\alpha_i y_i = 0, \end{aligned} \tag{1.6}$$

where the dual variable $\boldsymbol{\alpha}$ is related to the solution of eq. (1.5) by $\mathbf{w} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i$. Most important point in this dual representation is we can replace the inner product $\mathbf{x}_i\cdot\mathbf{x}_j$ in the objective with a nonlinear kernel $k(\mathbf{x}_i,\mathbf{x}_j) = \mathbf{\Phi}(\mathbf{x}_i)\cdot\mathbf{\Phi}(\mathbf{x}_j)$. This makes it possible to use SVMs in nonlinear feature space where it is easier to find a linear decision boundary due to the high dimensionality of the space. Figure 1.2

illustrates this point. The required optimization in eq. (1.6) is a convex problem in quadratic programming, and we can obtain the global optimum very efficiently using specialized solvers (Platt, 1998). Finally, note that 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.

Deep Learning

Notwithstanding these successes of kernel methods and SVMs, 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 and LeCun, 2007).

Deep architectures learn complex mappings by transforming their inputs through multiple layers of nonlinear processing (Hinton et al., 2006). Figure 1.3 shows a multilayer neural network that transforms an input example $\mathbf{x} \in \mathbb{R}^d$ to the output $t \in [0, 1]$ for binary classification of the example. Each layer defines nonlinear transformation of inputs through the combination of linear mapping and nonlinear activation functions. For example, the original input vector $\mathbf{x}$ is first linearly mapped by a weight matrix $\mathbf{W}_1$; then, a bias term $\mathbf{b}_1$ is added; finally, sigmoid activation function[2] is applied elementwise:

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}. \tag{1.7}$$

The resulting vector $\mathbf{h}_1$ (called hidden units) is then fed to the next layer as inputs, and the same procedure is repeated multiple times. At the final layer, the output is a prediction score between 0 and 1 for the binary label of the input example $\mathbf{x}$.

We formulate this model as an optimization problem with respect to the parameters $\mathbf{W} = \{\mathbf{W}_i, \mathbf{b}_i\}_{i=1}^4$, which correspond to all the linear weights and the bias terms in the network. Since the task is binary classification, we use the cross-entropy loss to measure the performance of the model. More specifically, for a data

[2]There are other types of nonlinear activation functions such as $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.

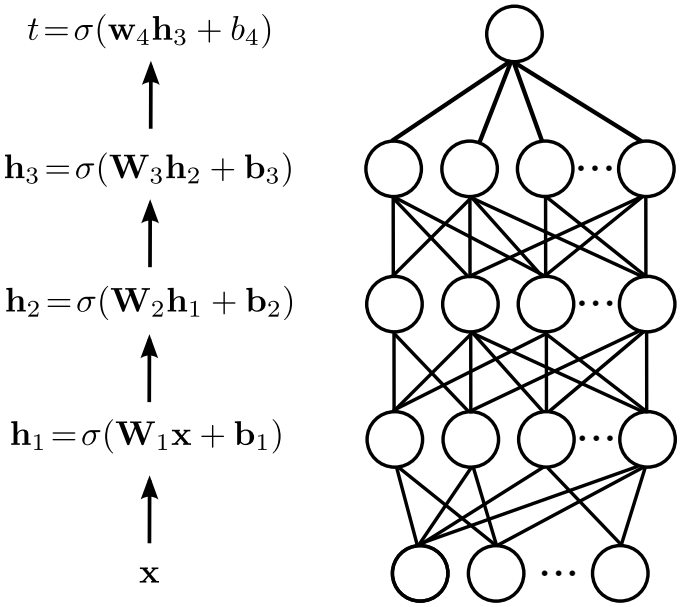


Figure 1.3: Multilayer neural networks. An input example $\mathbf{x} \in \mathbb{R}^d$ is transformed to the output $t \in [0, 1]$ through multiple layers of linear mappings $\mathbf{W}_i$ and an elementwise sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. Intermediate hidden representations are denoted as $\mathbf{h}_i$ and bias terms are $\mathbf{b}_i$.

set of N examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where $\mathbf{x}_i \in \mathbb{R}^d$ is the ith example and $y_i \in \{0, 1\}$ is its label, the optimization problem is defined as:

$$\min_{\mathbf{W}} \quad -\sum_{i=1}^{N} [\ y_i \log t_i + (1 - y_i) \log(1 - t_i)\ ], \tag{1.8}$$

where t_i is the prediction of the model for $\mathbf{x}_i$ as described in Figure 1.3.

The loss function in eq. (1.8) is not convex with respect to $\mathbf{W}$, and thus we are not guaranteed to find its global minimum (in contrast to the support vector machines in eq. (1.5)). As a result, gradient-based algorithms have been the main tools for the minimization of the objective, and especially (stochastic) gradient descent has been favored for its efficiency and simplicity. The error backpropagation algorithm (Rumelhart et al., 1986) is an efficient implementation of the gradient descent for multilayer neural networks.

Multilayer neural networks have also inspired development of other deep architectures. For example, deep belief nets (Hinton et al., 2006) are probabilistic

generative models that are constructed as layers of restricted Boltzmann machines. With unsupervised pre-training of the networks, they start from advantageous initial points for optimization of the weight parameters by the subsequent back-propagation algorithm. Stacked denoising autoencoders (Vincent et al., 2008) are another variant of multilayer neural networks where each layer is an autoencoder that extracts important features from inputs by minimizing the reconstruction error for the inputs. Similar to deep belief nets, each layer is trained in turn for pre-training of the weights. Finally, convolutional neural networks (LeCun et al., 1998; Ranzato et al., 2007) are special types of multilayer neural networks that exploit spatial correlation between features (e.g., image pixels). They repeat convolution and feature pooling stages multiple times to construct meaningful feature hierarchies.

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 including dimensionality reduction (Hinton and Salakhutdinov, 2006), computer vision (Ranzato et al., 2007), and natural language processing (Collobert and Weston, 2008).

1.2 Main Idea

Many issues surround the ongoing debate over deep versus shallow architectures (Bengio and 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, which do not guarantee global optimal solutions due to the non-convexity of the optimization problem. 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.

Like many, we are intrigued by the successes of deep architectures yet drawn to the elegance of kernel methods. In this thesis, we develop and explore a new connection between these two different approaches to statistical learning. Specifically, we begin with introducing a new family of positive-definite kernels called arc-cosine kernels that mimic the computation in large neural networks with a single layer 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 neural networks.

Our work then extends the arc-cosine kernels with different activation functions. The original arc-cosine kernels are derived by considering the mappings in neural networks with Heaviside step functions. We derive three new kernels with general one-sided polynomial activation functions, shifted Heaviside step functions, and sigmoidal activation functions respectively. These modifications are inspired by similar counterparts in conventional neural networks. They provide new hyperparameters—the polynomial growth rate, the amount of shift or smoothing—that can be tuned to improve the performance of the corresponding kernels.

We also show how to extend the family of arc-cosine kernels via combinations of individual members of the family. We use kernel composition, multiplication, and averaging to build more complex kernels. These operations yield more interesting results when they are applied to arc-cosine kernels than polynomial or Gaussian kernels. These new kernels can be interpreted as interesting forms of computations in multilayer neural networks. In particular, kernels from the composition of arc-cosine kernels mimic the computation in large multilayer neural networks and thus are called as multilayer kernels. We evaluate these kernels in support vector machines for large margin classification. In experiments on data

sets from a deep learning benchmark (Larochelle et al., 2007), we obtain competitive results compared to state-of-the-art methods. Interestingly, we often observe improved performance from combinations of more numbers of kernels (e.g., more layers in multilayer kernels), which is reminiscent of experience with multilayer neural networks.

Encouraged by such promising results in this direction of research, we propose another kernel-based hierarchical model called multilayer kernel machines (MKMs). Based on interesting parallels between our work (from original arc-cosine kernels to multilayer arc-cosine kernels) and the history of neural networks (from perceptrons to multilayer neural networks), our new model adopts recent advances in deep architectures—unsupervised learning. Specifically, we train MKMs by a combination of unsupervised and supervised learning methods. At the core of MKMs, we recursively iterate three processes: nonlinear transform by arc-cosine kernels, unsupervised dimensionality reduction by kernel principal component analysis (Schölkopf et al., 1998), and feature selection by mutual information (Guyon and Elisseeff, 2003). This cycle is repeated multiple times to construct the feature hierarchy of MKMs. MKMs are designed as a denoised version of the multilayer arc-cosine kernels, and this point is empirically verified using data sets of noisy image classification.

Finally, we carry out theoretical analysis of arc-cosine kernels in terms of the geometric properties. We aim to provide a richer understanding of the geometry of surfaces induced by arc-cosine kernels using tools from differential geometry (Amari and Wu, 1999; Burges, 1999). These surfaces are the images of the input space under the implicit nonlinear mapping performed by the kernels (and by association, the nonlinear transformations parameterized by large neural networks). Compared with other popular kernels such as Gaussian and polynomial kernels, we show that family of arc-cosine kernels exhibits a larger variety of behaviors—described by non-analytic, flat, or curved Riemannian manifold. We suggest how to utilize the geometric properties of these kernels in new research directions including nonlinear independent component analysis (Jutten and Karhunen, 2004).

1.3 Organization

The organization of this paper is as follows. In Chapter 2, we derive arc-cosine kernels from large single-layer neural networks and contrast their properties with those of other popular kernels for SVMs. We experiment with the arc-cosine kernels using SVMs and we also explain classification data sets that are used throughout this work. In Chapter 3, we show how to construct new arc-cosine kernels by considering neural networks with different activation functions. We also evaluate these kernels using SVMs for classification tasks. In Chapter 4, we review how to construct new kernels by composing, averaging, and taking products of existing ones and examine the form of such extended kernels when these operations are performed on arc-cosine kernels. We experiment with these new kernels and highlight interesting trends in performance that verify our intuition behind such construction. In Chapter 5, we describe multilayer kernel machines, kernel-based architecture with multiple layers of nonlinear transformation. We provide experimental results and discuss the strengths and weaknesses of our approach. In Chapter 6, we analyze the surfaces in Hilbert spaces induced by arc-cosine kernels and derive expressions for the metric, volume element, and scalar curvature when these surfaces can be described as Riemannian manifolds. Finally, in Chapter 7, we conclude by summarizing our most important contributions and suggesting directions for future research.

Chapter 1, in part, is a reprint of the material as it appears in Advances in Neural Information Processing Systems 22. Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 1, in part, is a reprint of the material as it appears in Neural Computation 22(10). Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 1, in part, is a reprint of the material as it appears in Technical Report CS2012-0972, Department of Computer Science and Engineering, University of California, San Diego. Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 2

Arc-cosine Kernels

2.1 Kernels From Neural Networks

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

$$k_n(\mathbf{x}, \mathbf{y}) \;=\; 2 \int \mathbf{dw} \; \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}} \; \Theta(\mathbf{w} \cdot \mathbf{x}) \; \Theta(\mathbf{w} \cdot \mathbf{y}) \; (\mathbf{w} \cdot \mathbf{x})^n \; (\mathbf{w} \cdot \mathbf{y})^n. \tag{2.1}$$

The kernel function in eq. (2.1) has interesting connections to neural computation (Williams, 1998) that we explore further. However, we begin by elucidating its basic properties.

2.1.1 Basic Properties

We focus primarily on kernel functions in this family with non-negative integer values of $n \in \{0, 1, 2, \ldots\}$. However, for non-zero inputs $\mathbf{x}$ and $\mathbf{y}$, the multidimensional integral in eq. (2.1) is well-defined and bounded for all real values $n > -\frac{1}{2}$, which we consider in Chapter 3. For non-negative integer values, we show how to evaluate the integral in eq. (2.1) analytically in Appendix A[1]. The final

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

result is most easily expressed in terms of the angle θ between the inputs:

$$\theta \;=\; \cos^{-1}\left(\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right). \tag{2.2}$$

The integral in eq. (2.1) 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:

$$k_n(\mathbf{x},\mathbf{y}) \;=\; \frac{1}{\pi}\,\|\mathbf{x}\|^n\|\mathbf{y}\|^n J_n(\theta) \tag{2.3}$$

where all the angular dependence is captured by the family of functions $J_n(\theta)$. Like homogeneous polynomial kernels, note how these kernels also behave in a simple way when the inputs are scaled: $k_n(\alpha\mathbf{x},\beta\mathbf{y}) = (\alpha\beta)^n k_n(\mathbf{x},\mathbf{y})$ for scaling factors $\alpha,\beta \geq 0$. Evaluating the integral in Appendix A, we show that this angular dependence is given by:

$$J_n(\theta) \;=\; (-1)^n(\sin\theta)^{2n+1}\left(\frac{1}{\sin\theta}\,\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi-\theta}{\sin\theta}\right) \quad \text{for} \quad \forall n \in \{0,1,2,\ldots\}. \tag{2.4}$$

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:

$$J_0(\theta) \;=\; \pi-\theta, \tag{2.5}$$

$$J_1(\theta) \;=\; \sin\theta + (\pi-\theta)\cos\theta, \tag{2.6}$$

$$J_2(\theta) \;=\; 3\sin\theta\cos\theta + (\pi-\theta)(1+2\cos^2\theta). \tag{2.7}$$

Higher-order expressions can be computed from eq. (2.4). Figure 2.1 compares the form of $J_n(\theta)$ for different settings of n. We describe eq. (2.3) as an arc-cosine kernel because for $n=0$, it takes the simple form[2]: $k_0(\mathbf{x},\mathbf{y}) = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right)$.

Arc-cosine kernels have other intriguing properties. From the magnitude dependence in eq. (2.3), we observe the following: (i) the $n=0$ arc-cosine kernel

[2]To constrain the maximum value of $k_0(\mathbf{x},\mathbf{y})$ as 1, we use the multiplicative factor 2 in eq. (2.1).

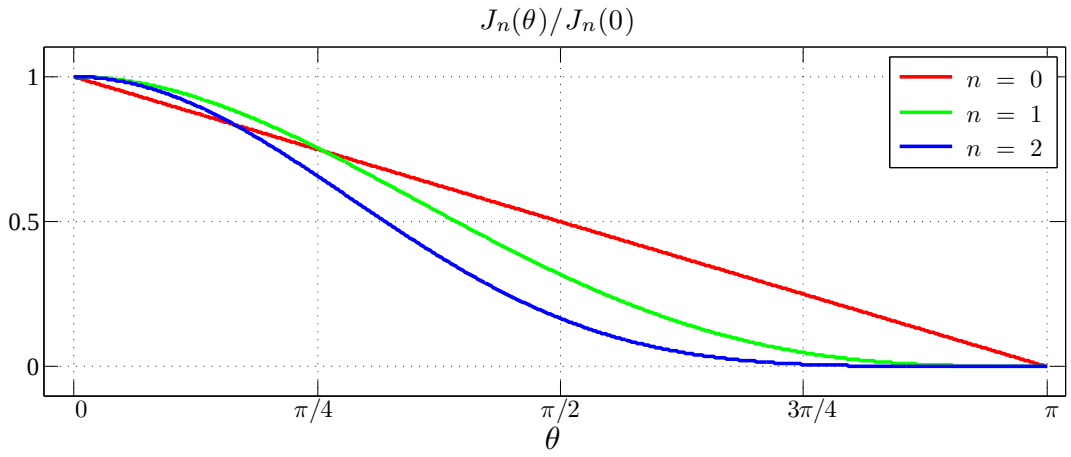


Figure 2.1: Form of $J_n(\theta)$ in eq. (2.4) for arc-cosine kernels with different values of n. The function $J_n(\theta)$ takes its maximum value at $\theta = 0$ and decays monotonically to zero at $\theta = \pi$ for all values of n. However, note the different behaviors for small θ.

maps inputs $\mathbf{x}$ to the unit hypersphere in feature space, with $k_0(\mathbf{x}, \mathbf{x}) = 1$; (ii) the $n = 1$ arc-cosine kernel preserves the norm of inputs, with $k_1(\mathbf{x}, \mathbf{x}) = \|\mathbf{x}\|^2$; (iii) higher order ($n > 1$) arc-cosine kernels expand the dynamic range of the inputs, with $k_n(\mathbf{x}, \mathbf{x}) \sim \|\mathbf{x}\|^{2n}$. Properties (i)–(iii) are shared respectively by radial basis function (RBF), linear, and polynomial kernels. Interestingly, though, the $n = 1$ arc-cosine kernel is highly nonlinear, also satisfying $k_1(\mathbf{x}, -\mathbf{x}) = 0$ for all inputs $\mathbf{x}$.

Finally, we verify that the arc-cosine kernels $k_n(\mathbf{x}, \mathbf{y})$ are positive-definite. In particular, consider any finite kernel matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$ whose elements $K_{ij} = k_n(\mathbf{x}_i, \mathbf{x}_j)$ store pairwise evaluations of the kernel function for inputs $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. Then for all $\mathbf{z} \in \mathbb{R}^N$, we have:

$$\begin{aligned}
\mathbf{z}^\top \mathbf{K}\, \mathbf{z} &= \sum_{i=1}^{N} \sum_{j=1}^{N} k_n(\mathbf{x}_i, \mathbf{x}_j)\; z_i\; z_j \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} 2 \int \mathbf{dw}\; \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}} \left[\Theta(\mathbf{w} \cdot \mathbf{x}_i)\; (\mathbf{w} \cdot \mathbf{x}_i)^n\, z_i\right] \left[\Theta(\mathbf{w} \cdot \mathbf{x}_j)\; (\mathbf{w} \cdot \mathbf{x}_j)^n\, z_j\right] \\
&= 2 \int \mathbf{dw}\; \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}} \left[\sum_{i=1}^{N} \Theta(\mathbf{w} \cdot \mathbf{x}_i)\; (\mathbf{w} \cdot \mathbf{x}_i)^n\, z_i\right]^2 \geq 0. \qquad (2.8)
\end{aligned}$$

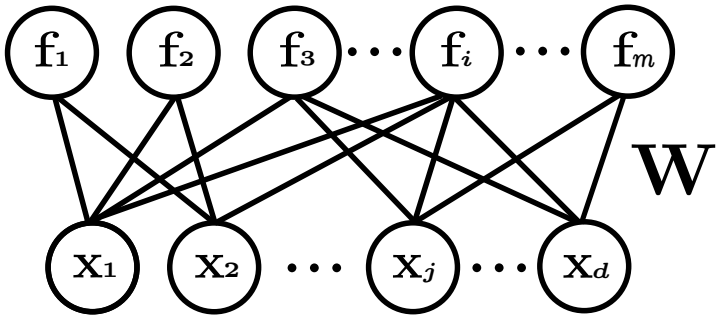


Figure 2.2: A single-layer threshold network where $\mathbf{x} \in \mathbb{R}^d$ is transformed to $\mathbf{f} \in \mathbb{R}^m$ by a nonlinear mapping.

Eq. (2.8) holds for all sets of $N \geq 1$ inputs $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$; thus, the kernel function in eq. (2.1) is positive-definite. In fact, this property also follows by observing that the integral representation in eq. (2.1) defines a covariance function (i.e., the expected product of functions evaluated at $\mathbf{x}$ and $\mathbf{y}$).

2.1.2 Computation in Single-layer Threshold Networks

Consider the single-layer network shown in Figure 2.2 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:

$$g_n(z) \;=\; \Theta(z) z^n, \tag{2.9}$$

which are displayed in Figure 2.3. 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

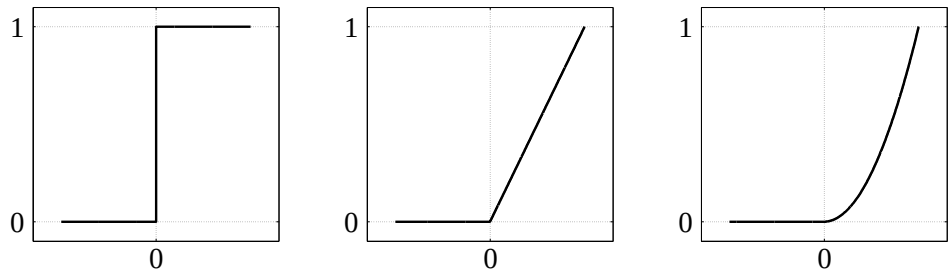


Figure 2.3: Nonlinear activation functions $g_n(z)$ in eq. (2.9) for different values of $n = 0, 1,$ and 2.

the arc-cosine kernel function in eq. (2.1). 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:

$$\mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{y}) \;=\; \sum_{i=1}^{m} \Theta(\mathbf{w}_i \cdot \mathbf{x})\Theta(\mathbf{w}_i \cdot \mathbf{y})(\mathbf{w}_i \cdot \mathbf{x})^n(\mathbf{w}_i \cdot \mathbf{y})^n, \tag{2.10}$$

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. (2.10) reduces to eq. (2.1) up to a trivial multiplicative factor:

$$\lim_{m\to\infty} \left[\frac{2}{m}\, \mathbf{f}(\mathbf{x})\cdot\mathbf{f}(\mathbf{y})\right] \tag{2.11}$$

$$= \lim_{m\to\infty} \left[\frac{2}{m} \sum_{i=1}^{m} \Theta(\mathbf{w}_i \cdot \mathbf{x})\Theta(\mathbf{w}_i \cdot \mathbf{y})(\mathbf{w}_i \cdot \mathbf{x})^n(\mathbf{w}_i \cdot \mathbf{y})^n\right] \tag{2.12}$$

$$= 2\, \mathbb{E}_{\mathbf{w}\sim\mathcal{N}(0,I_d)}\big[\, \Theta(\mathbf{w}\cdot\mathbf{x})\; \Theta(\mathbf{w}\cdot\mathbf{y})\; (\mathbf{w}\cdot\mathbf{x})^n\; (\mathbf{w}\cdot\mathbf{y})^n \,\big] \tag{2.13}$$

$$= 2 \int \mathbf{dw}\; \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\; \Theta(\mathbf{w}\cdot\mathbf{x})\; \Theta(\mathbf{w}\cdot\mathbf{y})\; (\mathbf{w}\cdot\mathbf{x})^n\; (\mathbf{w}\cdot\mathbf{y})^n \tag{2.14}$$

$$= k_n(\mathbf{x}, \mathbf{y}). \tag{2.15}$$

Thus the arc-cosine kernel function in eq. (2.1) can be viewed as the inner prod-

uct between feature vectors derived from the mapping of an infinite single-layer threshold network.

Many researchers have noted the general connection between kernel machines and one layer neural networks (Bengio and LeCun, 2007). Interestingly, the $n = 0$ arc-cosine kernel in eq. (2.1) 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. (2.1) for $n = 0$, but with two differences: first, the weights $\mathbf{w}$ were integrated over a Gaussian distribution with a general covariance matrix:

$$\mathbf{w} \sim \mathcal{N}(0, \Sigma); \tag{2.16}$$

second, the activation function was an error function with range $[-1, 1]$ as opposed to a step function with range $[0, 1]$:

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \; e^{-t^2} dt. \tag{2.17}$$

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 $2\Theta(z) - 1$. The kernel function $k_{\mathrm{GP}}(\mathbf{x}, \mathbf{y})$ in this limit is expressed in terms of arc-sine functions of normalized dot products between inputs[3]:

$$k_{\mathrm{GP}}(\mathbf{x}, \mathbf{y}) = \int \mathbf{dw} \; \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}} \; (2\Theta(\mathbf{w} \cdot \mathbf{x}) - 1) \; (2\Theta(\mathbf{w} \cdot \mathbf{y}) - 1) \tag{2.18}$$

$$= \frac{2}{\pi} \sin^{-1} \left(\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \right). \tag{2.19}$$

Using elementary identities, our result for the $n = 0$ arc-cosine kernel follows as a

[3]The original form in Williams (1998) is $k_{\mathrm{GP}}(\mathbf{x}, \mathbf{y}) = \frac{2}{\pi} \sin^{-1} \left(\frac{2\bar{\mathbf{x}}^\top \Sigma \bar{\mathbf{y}}}{\sqrt{(1+2\bar{\mathbf{x}}^\top \Sigma \bar{\mathbf{x}})(1+2\bar{\mathbf{y}}^\top \Sigma \bar{\mathbf{y}})}} \right)$ where $\bar{\mathbf{x}} = (1, \mathbf{x})$ and $\bar{\mathbf{y}} = (1, \mathbf{y})$.

simple corollary:

$$k_{\mathrm{GP}}(\mathbf{x},\mathbf{y}) = 2k_0(\mathbf{x},\mathbf{y}) - 2\int \mathbf{dw}\ \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\ \Theta(\mathbf{w}\cdot\mathbf{x}) \tag{2.20}$$

$$-2\int \mathbf{dw}\ \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\ \Theta(\mathbf{w}\cdot\mathbf{y}) + \int \mathbf{dw}\ \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\ 1. \tag{2.21}$$

$$k_0(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left[\ k_{\mathrm{GP}}(\mathbf{x},\mathbf{y}) + 2\cdot\frac{1}{2} + 2\cdot\frac{1}{2} - 1\ \right] \tag{2.22}$$

$$= \frac{1}{2}\left[\ \frac{2}{\pi}\sin^{-1}\left(\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right) + 1\ \right] \tag{2.23}$$

$$= \frac{1}{2}\left[\ \frac{2}{\pi}\left\{\frac{\pi}{2} - \cos^{-1}\left(\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right)\right\} + 1\ \right] \tag{2.24}$$

$$= 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right). \tag{2.25}$$

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 our work, 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).

Arc-cosine kernels differ from polynomial and RBF kernels in one especially interesting respect. As highlighted by the integral representation in eq. (2.1), arc-cosine kernels induce feature spaces that mimic the sparse, non-negative, distributed representations of single-layer threshold networks. Polynomial and RBF kernels do not encode their inputs in this way. In particular, the feature vector induced by polynomial kernels is neither sparse nor non-negative, while the feature vector induced by RBF kernels resembles the localized output of a soft vector quantizer. Chapter 4 explores further implications of this difference.

Finally, note that the computation in infinite neural networks with a Gaussian prior can also be explained from a different perspective—randomized weights

in statistical models. In particular, it has been found that instead of learning all the weight parameters of statistical models, randomizing (large parts of) the weights could lead to surprisingly good results with much less effort in training. This is mainly due to the random projection theorem (Johnson and Lindenstrauss, 1984), which provides theoretical guarantees that the distance between two data points is not distorted much after random projections. Based on that, Rahimi and Recht (2009) proposed a randomized learning model where inputs are passed through a large bank of randomized nonlinearities, and the resulting outputs are linearly combined to achieve state-of-the-art performance in classification. Furthermore, recent advances in deep architectures (Jarrett et al., 2009; Saxe et al., 2011) have also been made using networks with a large number of hidden units and random weights, suggesting it may be more important to efficiently maintain large number of hidden units and layers in neural networks than to train the weights carefully. The arc-cosine kernel in eq. (2.1) fits well into this scenario. It is defined from a network with infinitely many hidden units and whose weights are randomly drawn from a Gaussian distribution; then support vector machines are utilized to choose the best features from the large number of randomly generated features for large margin classification.

2.2 Experimental Results

We evaluated SVMs with arc-cosine kernels on several medium-sized data sets for classification. Our experiments in this section had two goals: first, to compare arc-cosine kernels with more traditional RBF kernels and linear kernels for various classification tasks; second, to provide baseline results that we would improve upon in the following chapters.

2.2.1 Data Sets

Table 2.1 lists the nine data sets used in our experiments and Figure 2.4 shows examples from the first six data sets, which can be visualized as 28×28 grayscale images. The first data set is *MNIST*, which is a well-known classification

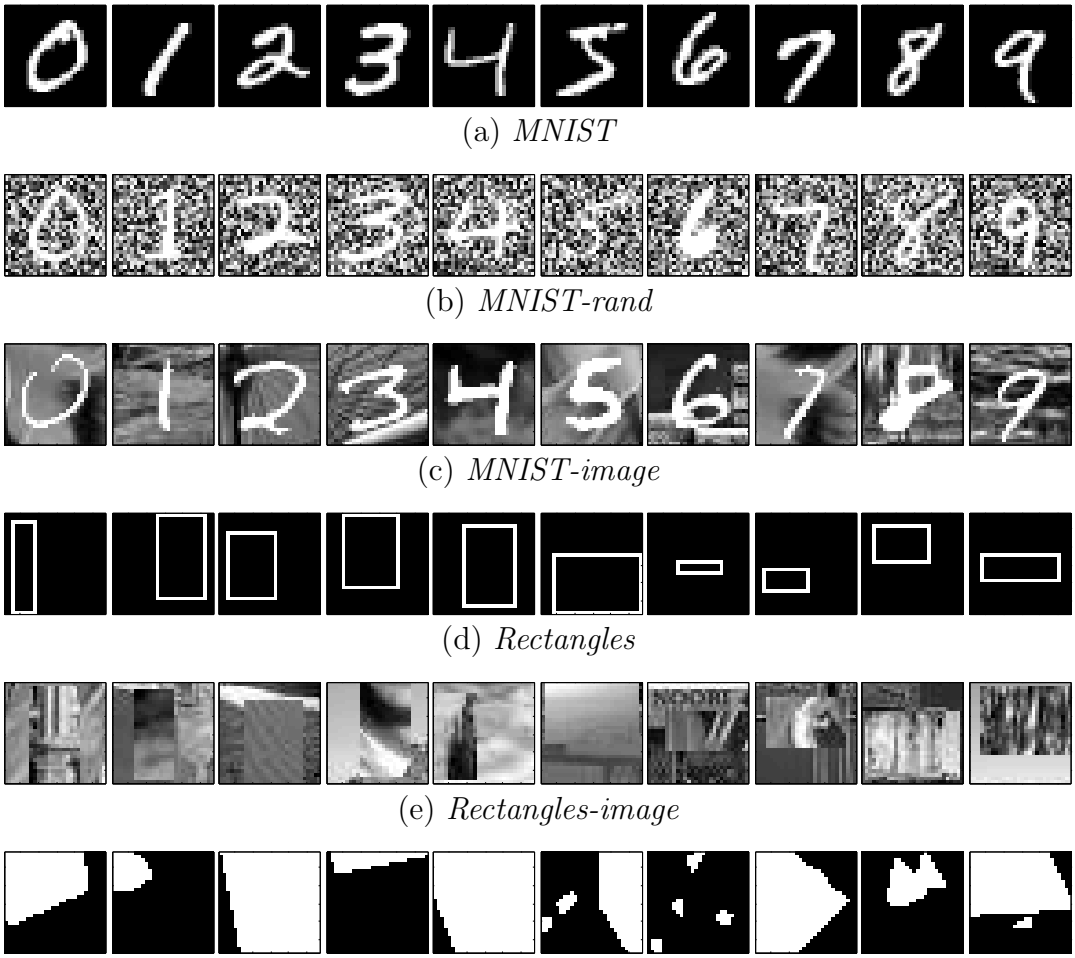
Table 2.1: Data set specifications: the number of training, validation, and test examples, input dimensionality, and the number of classes.

Data set	Training	Validation	Test	Dimension	Class
MNIST	50000	10000	10000	784	10
MNIST-rand	10000	2000	50000	784	10
MNIST-image	10000	2000	50000	784	10
Rectangles	1000	200	50000	784	2
Rectangles-image	10000	2000	50000	784	2
Convex	6000	2000	50000	784	2
20-Newsgroups	12748	3187	3993	62061	20
ISOLET	4990	1248	1559	617	26
Gisette	4800	1200	1000	5000	2

data set of grayscale handwritten digits (LeCun and Cortes, 1998). We held out the last (roughly) 1000 training examples of each digit class as validation examples.

The following five data sets in the table are image classification benchmarks from an empirical evaluation of deep learning (Larochelle et al., 2007). The first two of these are noisy variations of *MNIST*: the task in *MNIST-rand* is to recognize digits whose backgrounds have been corrupted by white noise, while the task in *MNIST-image* is to recognize digits whose backgrounds consist of other image patches. The other benchmarks are purely synthetic data sets. The task in *Rectangles* is to classify a single rectangle that appears in each image as tall or wide. *Rectangles-image* is a harder variation of this task in which the background of each rectangle consists of other image patches. Finally, the task in *Convex* is to classify a single white region that appears in each image as convex or non-convex. We partitioned these data sets into training, validation, and test examples as in previous benchmarks.

Note that the classes in *Rectangles*, *Rectangles-image*, and *Convex* 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 ex-



(a) *MNIST*

(b) *MNIST-rand*

(c) *MNIST-image*

(d) *Rectangles*

(e) *Rectangles-image*

(f) *Convex*

Figure 2.4: Examples from the first six data sets in Table 2.1 represented as 28 × 28 grayscale images (LeCun and Cortes, 1998; Larochelle et al., 2007).

hibited the biggest performance gap between deep architectures (e.g., deep belief nets) and traditional SVMs.

The bottom three data sets in Table 2.1 are from benchmark problems in text categorization, spoken letter recognition, and feature selection. The task in *20-Newsgroups* is to classify newsgroup postings (represented as bags of words) into one of twenty news categories (Lang, 1995). The task in *ISOLET* is to identify a spoken letter of the English alphabet (Frank and Asuncion, 2010). The task in *Gisette* is to distinguish the *MNIST* digits FOUR versus NINE, but the input representation has been padded with a large number of additional features—some

helpful, some spurious, and some sparse (Guyon et al., 2005). We randomly held out 20% of the training examples in these data sets for validation.

2.2.2 Methodology

For classification by SVMs, we compared five different kernels—the arc-cosine kernels of degree $n = 0, 1, 2$, the radial basis function (RBF) kernel, and the linear kernel. All SVMs were trained using LIBSVM (Chang and Lin, 2001), a publicly available software package. For multiclass problems, we adopted the so-called one-versus-one approach: SVMs were trained for each pair of different classes, and test examples were labeled by the majority vote of all the pairwise SVMs.

We followed a similar experimental methodology as in previous work to tune the margin-violation penalties in SVMs as well as the kernel width in RBF kernels (Larochelle et al., 2007). We used the held-out (validation) examples to determine these values, first searching over a coarse logarithmic grid, then performing a fine-grained search to improve their settings. Once these values were determined, however, we retrained each SVM on the combined set of training and validation examples. We used these retrained SVMs for the final performance evaluations on test examples.

2.2.3 Results

Table 2.2 displays the test error rates from the experiments where we can observe three interesting points as follows. First, arc-cosine kernels return lower error rates compared to the linear kernel in all the data sets, and such differences are generally bigger in the first six data sets of image classification; for high dimensional data sets like *20-Newsgroups*, however, the arc-cosine kernels (and the RBF kernel as well) are only slightly better than the linear kernel because it is not difficult to find linear decision boundaries in the high dimensional space.

Second, the arc-cosine kernels seem to be inferior to the RBF kernel in majority of cases. We suspect this is due to the lack of continuous tuning parameters

Table 2.2: Classification error rates (%) on test sets from SVMs with various kernels. The first three kernels are arc-cosine kernels of degree $n=0, 1,$ and 2. The best performing kernel for each data set is marked in bold. When different, the best performing arc-cosine kernel is marked in italics. See text for details.

Data set	Arc-cosine			RBF	Linear
	$n=0$	$n=1$	$n=2$		
MNIST	1.68	1.7	*1.63*	**1.31**	4.15
MNIST-rand	17.16	17.56	*16.87*	**14.80**	17.31
MNIST-image	*23.81*	24.76	24.36	**22.80**	25.07
Rectangles	13.08	4.62	*3.79*	**2.11**	30.30
Rectangles-image	**22.66**	24.6	24.94	23.42	49.69
Convex	20.05	19.9	*19.53*	**18.76**	45.67
20-Newsgroups	16.28	15.88	**15.63**	15.75	15.90
ISOLET	*3.40*	3.46	3.53	**3.01**	3.53
Gisette	**1.80**	2.2	2.1	2.10	2.20

in the arc-cosine kernels in contrast to RBF kernels where we can adapt the kernel width parameter to fit the particular data sets better. We attempt to address this point by introducing continuous tuning parameters to arc-cosine kernels in Chapter 3.

Third, it remains to be understood why some problems are better suited for certain 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 2.4) because it normalizes for brightness, only considering the angle between two images. Admittedly, choosing the best degree in arc-cosine kernels is dependent on data sets and thus cross-validation seems to be the only reasonable method for this issue, which is often the case for other kernels as well. However, we claim that this search for hyperparameters in kernels is an easier problem in need of less expertise compared to training complex deep architectures, given its easily parallelizable structure and efficient quadratic programming solvers for SVMs (Platt, 1998).

Finally, note that RBF kernels in the first six data sets of Table 2.2 are reported to return higher error rates than deep architectures, which were carefully

trained by experts (Larochelle et al., 2007)[4]. This will be reviewed more extensively in Chapter 4.

2.3 Discussion

In this chapter, we have explored a new family of positive-definite kernels called arc-cosine kernels. 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 satisfactory results (not state-of-the-art though).

Our approach 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. (2.1) for the arc-cosine kernel of degree $n=0$ as shown in Section 2.1.2. 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 any one-sided polynomial 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 degree $n>0$.

To improve the performance of arc-cosine kernels further, we explore various directions in the following chapters. First, in Chapter 3, we introduce continuous tuning parameters to the arc-cosine kernels by using different activation functions in the threshold networks and consequently in the definition of the kernels. Second, in Chapter 4, we combine arc-cosine kernels in hierarchical fashion to mimic the computations in multilayer threshold networks. Third, in Chapter 5, unsupervised learning methods are incorporated into arc-cosine kernels as in state-of-the-art deep architectures.

[4]Larochelle et al. (2007) reported slightly different results for RBF kernels compared to Table 2.2, but this is due to differences in the search ranges for hyperparameters and it does not affect the analysis in this section.

Chapter 2, in part, is a reprint of the material as it appears in Advances in Neural Information Processing Systems 22. Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 2, in part, is a reprint of the material as it appears in Neural Computation 22(10). Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 3

Variations in Activation Functions

3.1 Introduction

In this chapter, we explore three variations on the arc-cosine kernels that are derived from large neural networks with different activation functions. Specifically, we construct new kernels by adapting the activation functions as in Figure 3.1. We begin with using fractional degrees for the one-sided polynomial activation function in eq. (2.9). Then, we consider the effects of shifting the thresholds of Heaviside step functions as well as smoothing their nonlinearities. These modifications introduce continuous parameters—the polynomial growth rate, the amount of shift or smoothing—that can be tuned to improve the performance of the resulting kernels.

In particular, the first of these operations (fractional degrees) provides diverse options to model the input data (see Figure 3.2 for various forms of the activation function with respect to the change in the degree n); the second (biasing) induces more sparse representations of the data in feature space; the third (smoothing) refines ranges of feature representations by removing the discontinuities of the activation function.[1]. These effects are interesting to explore given the improvements they have yielded in conventional neural networks. We evaluate these variations of arc-cosine kernels in support vector machines for large margin classification. Our experiments show that these variations of arc-cosine kernels

[1]In Chapter 6, we show this is also related with non-analyticity of the arc-cosine kernel with degree $n=0$.

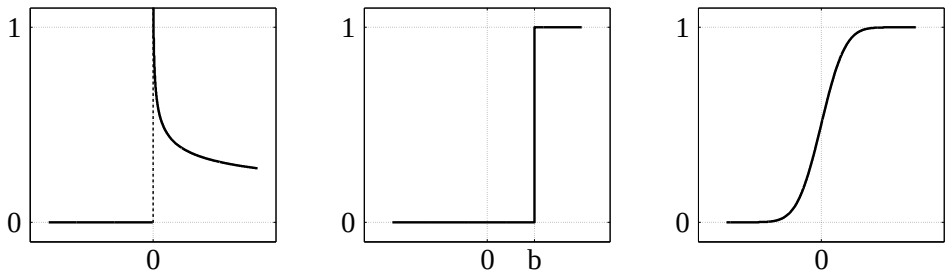


Figure 3.1: Nonlinear activation functions for new arc-cosine kernels. *Left*: fractional order polynomial function with degree $n=-0.25$ in eq. (2.9). *Middle*: biased threshold function $\Theta(z-b)$ with bias b. *Right*: smoothed threshold function $\Psi_\sigma(z)$ in eq. (3.14).

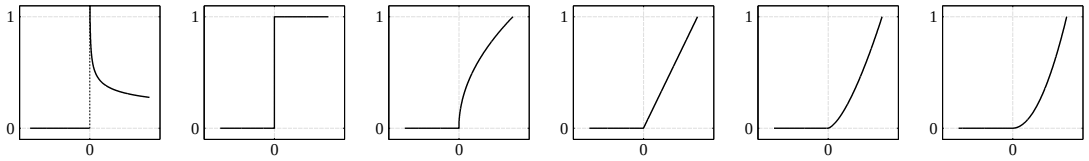


Figure 3.2: Various forms of the nonlinear activation functions $g_n(z)$ in eq. (2.9) for different values of $n = -0.25, 0, 0.5, 1, 1.5,$ and 2 in order from left to right.

often lead to better performance.

3.2 Activation Functions

3.2.1 Fractional Order Polynomial Threshold Functions

We consider the kernel function in eq. (2.1) for non-integer values of n. The general form in eq. (2.3) 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. (2.1) for the special case where $\mathbf{x} = \mathbf{y}$. In Appendix A, we show that:

$$k_n(\mathbf{x}, \mathbf{x}) \;=\; \frac{1}{\pi}\,\|\mathbf{x}\|^{2n}\,J_n(0) \quad \text{where} \quad J_n(0) \;=\; \sqrt{\pi}\,2^n\,\Gamma\!\left(n+\frac{1}{2}\right). \tag{3.1}$$

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. (2.1) is not defined. The magnitude of $k_n(\mathbf{x}, \mathbf{x})$

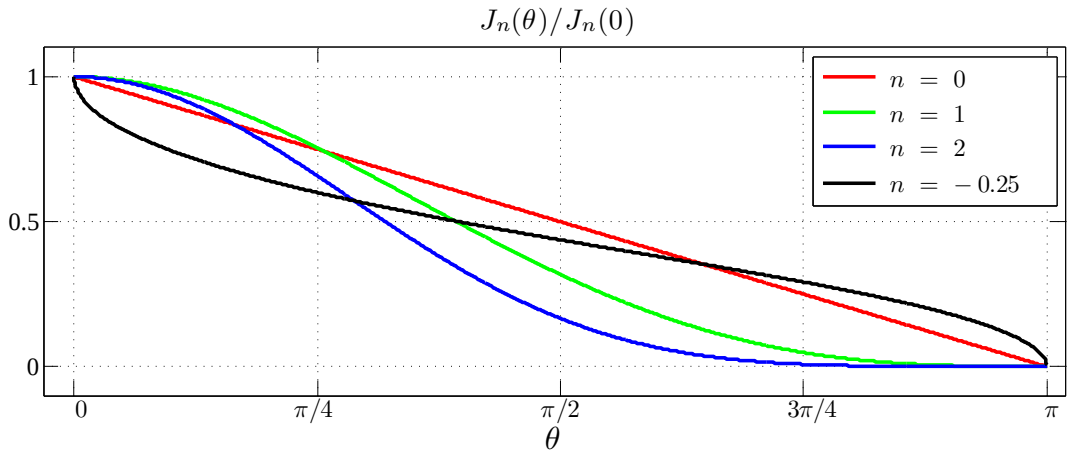


Figure 3.3: Behavior of the function $J_n(\theta)$ in eq. (2.4) including the case of $n = -0.25$.

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. (2.1) 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 Appendix A. Figure 3.3 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.

3.2.2 Biased Threshold Functions

Consider the arc-cosine kernel of degree $n = 0$ as defined by eq. (2.1). We obtain a new kernel by translating the Heaviside step functions in this definition

by a bias term $b \in \mathbb{R}$:

$$k^b(\mathbf{x}, \mathbf{y}) = 2 \int \mathbf{dw} \, \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}} \, \Theta(\mathbf{w} \cdot \mathbf{x} - b) \, \Theta(\mathbf{w} \cdot \mathbf{y} - b). \tag{3.2}$$

The motivation behind this construction is to regulate the sparsity of the infinite dimensional representation $\mathbf{\Phi}(\mathbf{x})$. Note that as the bias b is increased, a larger volume of the weight space $\mathbf{w} \in \mathbb{R}^d$ is associated with zero activation levels $\Theta(\mathbf{w} \cdot \mathbf{x} - b)$ from the input $\mathbf{x} \in \mathbb{R}^d$. Thus this construction is able to emulate a large neural network with especially sparse ($b > 0$) or dense ($b < 0$) hidden unit representations.

The integral in eq. (3.2) cannot be performed in closed form, but we can express it in terms of simple one-dimensional definite integrals. To this end, we use ξ, ψ, and θ to denote the internal angles of the triangle formed by the vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{x} - \mathbf{y}$; see Figure 3.4. Also, as shorthand, we define the two-parameter family of definite integrals:

$$I(r, \xi) \;=\; \frac{1}{\pi} \int_0^{\xi} d\phi \;\; \exp\left(- \frac{1}{2\, r^2 \sin^2 \phi} \right). \tag{3.3}$$

It is simple to compute this integral and store the results in a lookup table for discretized values of $\xi \in [0, \pi]$ and $r > 0$.

We begin by evaluating the right hand side of eq. (3.2) in the regime $b \geq 0$ of increased sparsity. Then, in terms of the above notation, we obtain the result:

$$k^b(\mathbf{x}, \mathbf{y}) \;=\; I\left(b^{-1}\|\mathbf{x}\|, \psi\right) + I(b^{-1}\|\mathbf{y}\|, \xi) \quad \text{for} \quad b \geq 0. \tag{3.4}$$

The derivation of this result is given in Appendix B.

The result in the opposite regime $b \leq 0$ is obtained by a simple transformation. In this regime, we can evaluate the integral in eq. (3.2) by noting that $\Theta(z) = 1 - \Theta(-z)$ and exploiting the symmetry of the integral in weight space. It follows that:

$$k^b(\mathbf{x}, \mathbf{y}) \;=\; k^{-b}(\mathbf{x}, \mathbf{y}) + \operatorname{erf}\left(\frac{-b}{\sqrt{2}\|\mathbf{x}\|} \right) + \operatorname{erf}\left(\frac{-b}{\sqrt{2}\|\mathbf{y}\|} \right) \quad \text{for} \quad b \leq 0, \tag{3.5}$$

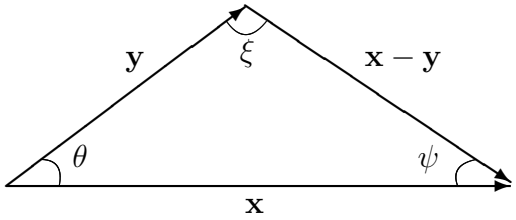


Figure 3.4: A triangle formed by the input data vectors $\mathbf{x}$, $\mathbf{y}$, and their difference $\mathbf{x} - \mathbf{y}$.

where $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is the error function. From the same observations, it also follows that kernel matrices for opposite values of b are equivalent up to centering (i.e., after subtracting out the mean in feature space). Specifically, let $\mathbf{b}$ and $\mathbf{C}$ denote a $N \times 1$ vector from eq. (3.5) and the $N \times N$ centering matrix respectively:

$$\mathbf{b} = \left[\mathrm{erf}\left(\frac{-b}{\sqrt{2}\|\mathbf{x}_1\|}\right), \mathrm{erf}\left(\frac{-b}{\sqrt{2}\|\mathbf{x}_2\|}\right), \ldots, \mathrm{erf}\left(\frac{-b}{\sqrt{2}\|\mathbf{x}_N\|}\right) \right]^\top, \tag{3.6}$$

$$\mathbf{C} = \mathbf{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top \tag{3.7}$$

where $\mathbf{x}_i$ is ith training example; $\mathbf{I}_N$ is the $N \times N$ identity matrix; $\mathbf{1}$ is the column vector of N ones. Then, a kernel matrix $\mathbf{K}^b \in \mathbb{R}^{N \times N}$ of N training examples with the arc-cosine kernel $k^b(\mathbf{x}, \mathbf{y})$ for $b \leq 0$ is centered to $\tilde{\mathbf{K}}^b$:

$$\tilde{\mathbf{K}}^b = \mathbf{C}\mathbf{K}^b\mathbf{C} \tag{3.8}$$

$$= \mathbf{C}\big(\mathbf{K}^{-b} + \mathbf{1}\mathbf{b}^\top + \mathbf{b}\mathbf{1}^\top\big)\mathbf{C} \tag{3.9}$$

$$= \mathbf{C}\mathbf{K}^{-b}\mathbf{C} + \left(\mathbf{1} - \frac{1}{N}\mathbf{1}\mathbf{1}^\top\mathbf{1}\right)\mathbf{b}^\top\mathbf{C} + \mathbf{C}\mathbf{b}\left(\mathbf{1}^\top - \frac{1}{N}\mathbf{1}^\top\mathbf{1}\mathbf{1}^\top\right) \tag{3.10}$$

$$= \mathbf{C}\mathbf{K}^{-b}\mathbf{C} \tag{3.11}$$

$$= \tilde{\mathbf{K}}^{-b}, \tag{3.12}$$

where we verified the equivalence. Thus without loss of generality, we only investigate kernels with biases $b \geq 0$ in our experiments on support vector machines.

As already noted, the arc-cosine kernel of degree $n = 0$ depends only on the angle between its inputs and not on their magnitudes. The kernel in eq. (3.4)

does not exhibit this same invariance. However, it does have the scaling property:

$$k^b(\rho\mathbf{x}, \rho\mathbf{y}) \;=\; k^{b/\rho}(\mathbf{x}, \mathbf{y}) \quad \text{for} \quad \rho > 0. \tag{3.13}$$

Eq. (3.13) shows that the effect of a different bias can be mimicked by uniformly rescaling all the inputs. This property is convenient because it can save us some numerical integrations in eq. (3.3) when we build lookup tables for certain values of b.

3.2.3 Smoothed Threshold Functions

We can extend the arc-cosine kernel of degree $n=0$ in a different way by smoothing the Heaviside step function in eq. (2.1). The simplest smooth alternative is the cumulative Gaussian function:

$$\Psi_\sigma(z) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{z} du\, e^{-\frac{u^2}{2\sigma^2}}, \tag{3.14}$$

which reduces to the Heaviside step function in the limit of vanishing variance ($\sigma^2 \to 0$). The resulting kernel is defined as:

$$k_\sigma(\mathbf{x}, \mathbf{y}) \;=\; 2\int \mathbf{dw}\, \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\; \Psi_\sigma(\mathbf{w}\cdot\mathbf{x})\; \Psi_\sigma(\mathbf{w}\cdot\mathbf{y}). \tag{3.15}$$

The variance parameter σ^2 can be tuned in this kernel just as its counterpart in a radial basis function (RBF) kernel. However, note that RBF kernels behave very differently than these kernels in the limit of vanishing variance: the former become degenerate, whereas eq. (3.15) reduces to the arc-cosine kernel of degree $n=0$.

The integral in eq. (3.15) can be performed analytically, yielding the result:

$$k_\sigma(\mathbf{x}, \mathbf{y}) = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\mathbf{x}\cdot\mathbf{y}}{\sqrt{(\|\mathbf{x}\|^2+\sigma^2)(\|\mathbf{y}\|^2+\sigma^2)}}\right). \tag{3.16}$$

Details of the calculation are given in Appendix C. The kernel in eq. (3.16) is analogous to one derived earlier by Williams (1998) in the context of Gaussian processes. However, in that work, the kernel was computed for an activation

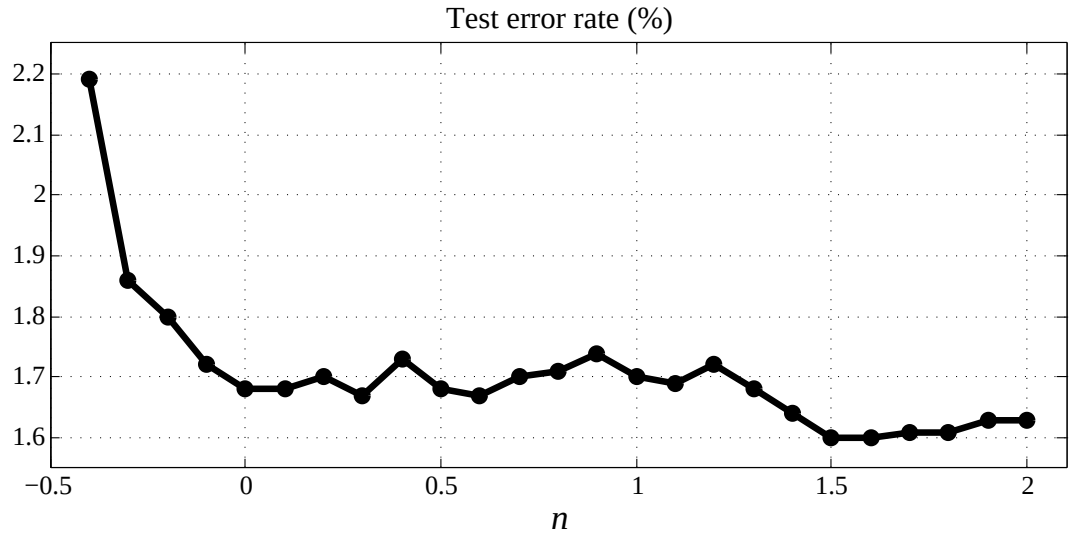


Figure 3.5: Classification error rates on the *MNIST* data set using arc-cosine kernels with fractional degrees n (horizontal axis).

function bounded between -1 and 1 (as opposed to 0 and 1, above).

3.3 Experimental Results

We evaluated the new kernels in this section by measuring their performance in support vector machines (SVMs). We also compared them to other popular kernels for large margin classification. We used the same data sets and followed the same methodology in Section 2.2.

We first experimented with fractional and negative values of the degree n in arc-cosine kernels using the *MNIST* data set (LeCun and Cortes, 1998). The kernel functions in these experiments had to be computed numerically, as described in Appendix A. Figure 3.5 shows the test error rates from SVMs in which we continuously varied the degree of the arc-cosine kernel. The best results in these experiments were obtained by using kernels with fractional degrees; however, the improvements were relatively modest.

On the other hand, we obtain more satisfying results from the other two cases of arc-cosine kernels with biased or smoothed activation functions. Table 3.1 displays the test error rates from these extensive experiments. In the majority of

Table 3.1: Classification error rates (%) on test sets from SVMs with various kernels. The first three kernels are the arc-cosine kernel of degree $n=0$ and the variations on this kernel described in sections 3.2.2 and 3.2.3. The best performing kernel for each data set is marked in bold. When different, the best performing arc-cosine kernel is marked in italics. See text for details.

Data set	Arc-cosine			RBF	Linear
	$n=0$	Bias	Smooth		
MNIST-rand	17.16	*16.49*	17.03	**14.80**	17.31
MNIST-image	23.81	*23.77*	24.09	**22.80**	25.07
Rectangles	13.08	*2.48*	11.84	**2.11**	30.30
Rectangles-image	**22.66**	23.59	24.48	23.42	49.69
Convex	20.05	20.12	*19.60*	**18.76**	45.67
20-Newsgroups	16.28	16.25	**15.73**	15.75	15.90
ISOLET	3.40	*3.34*	3.53	**3.01**	3.53
Gisette	**1.80**	1.90	1.90	2.10	2.20

cases, the parameterized variations of arc-cosine kernels achieve better performance than the original arc-cosine kernel of degree $n=0$. This demonstrates the utility of adapting the arc-cosine kernel to data sets using the bias term or the smoothness level. Though the performance degrades somewhat in the *Rectangles-image* data set, this is the case where model selection is misled by the limited number of validation examples (i.e., the new arc-cosine kernels actually return lower error rates in validation sets). Unfortunately, we find the arc-cosine kernels are still slightly inferior to the RBF kernels in the majority of data sets, but they are consistently better than the linear kernels.

3.4 Discussion

In this chapter, we explored variations of arc-cosine kernels by controlling the polynomial growth rate, the amount of shift or smoothing in the activation functions. We evaluated these new kernels extensively for large margin classification in SVMs. By tuning the continuous parameters in these kernels, we showed that they often performed better than the original arc-cosine kernel of degree $n=0$.

However, we found that these improvements were not big enough to out-

perform the RBF kernels (and the deep belief nets as well) in most of the data sets. Considering the cost incurred by cross-validation to choose the continuous parameters in the kernels[2], this is a somewhat discouraging result.

In the next chapter, we take a different approach: rather than enhancing a single arc-cosine kernel by modifying activation functions, we focus on how to combine basic arc-cosine kernels to improve the performance. Interestingly, this leads to development of new kernels that mimic the computations in large multilayer neural networks.

Chapter 3, in part, is a reprint of the material as it appears in Neural Computation 22(10). Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 3, in part, is a reprint of the material as it appears in Technical Report CS2012-0972, Department of Computer Science and Engineering, University of California, San Diego. Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

[2]The numerical integration in the first two cases is the main computational issue although this can be alleviated by storing pre-computed results in the reference tables.

Chapter 4

Combination of Arc-cosine Kernels

4.1 Introduction

In this chapter, we show how to construct more complex kernels via operations like kernel composition, multiplication, and averaging. These operations yield more interesting results when they are applied to arc-cosine kernels than polynomial or RBF kernels. For example, the composition or product of two polynomial kernels is simply another polynomial kernel, but of higher degree. Likewise, the product of two RBF kernels is just another RBF kernel, but with different variance. By contrast, when these operations are applied to arc-cosine kernels, they lead to new kernels that are not part of the original family. These new kernels can also be interpreted as interesting forms of computations in multilayer neural networks.

We evaluate the use of these extended constructions of arc-cosine kernels in support vector machines for large margin classification. Experimenting on the *MNIST* data set of handwritten digits (LeCun and Cortes, 1998), we highlight general trends in performance as the basic kernels in this family are combined in different ways. Our results show that with kernels formed by composing or taking products of other kernels, we often obtain significantly better performance, while we do not see any consistent benefits from averaging. Furthermore, in experiments

on data sets from a deep learning benchmark (Larochelle et al., 2007), we often obtain competitive results compared to the state-of-the-art methods such as SVMs with Gaussian kernels as well as deep belief nets.

4.2 Combinations of Kernels

4.2.1 Kernel Composition

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:

$$k(\mathbf{x}, \mathbf{y}) \;=\; \mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{y}). \tag{4.1}$$

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,

$$k^{(\ell)}(\mathbf{x}, \mathbf{y}) \;=\; \underbrace{\mathbf{\Phi}(\mathbf{\Phi}(...\mathbf{\Phi}}_{\ell \text{ times}}(\mathbf{x}))) \cdot \underbrace{\mathbf{\Phi}(\mathbf{\Phi}(...\mathbf{\Phi}}_{\ell \text{ times}}(\mathbf{y}))) \tag{4.2}$$

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. (4.2) 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. (4.2) 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 linear kernels $k(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$, the composition is trivial: in particular, it yields the identity map $\mathbf{\Phi}(\mathbf{\Phi}(\mathbf{x})) = \mathbf{\Phi}(\mathbf{x}) = \mathbf{x}$. For homogeneous polynomial kernels $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$, the composition yields:

$$\mathbf{\Phi}(\mathbf{\Phi}(\mathbf{x})) \cdot \mathbf{\Phi}(\mathbf{\Phi}(\mathbf{y})) = (\mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{y}))^d = (\mathbf{x} \cdot \mathbf{y})^{d^2}. \tag{4.3}$$

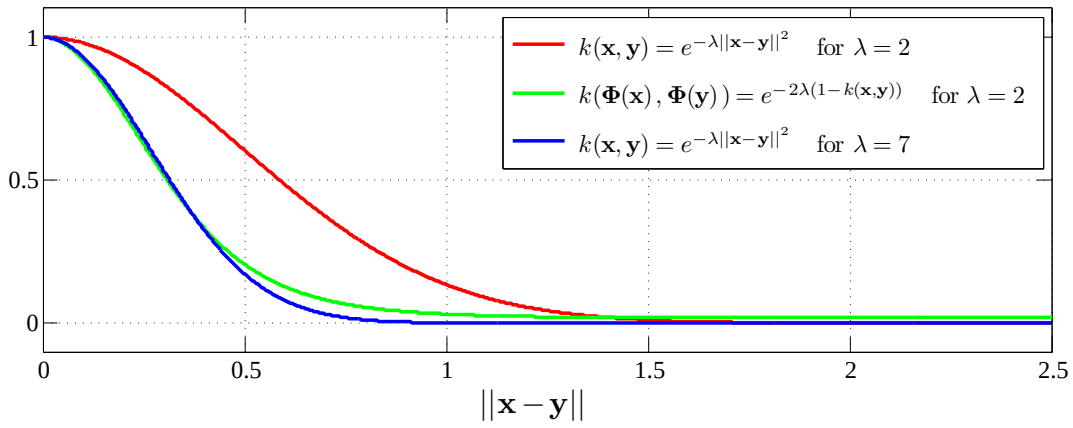


Figure 4.1: RBF kernels and their composition form as a function of $\|\mathbf{x}-\mathbf{y}\|$. The original RBF kernel with the kernel width $\lambda = 2$ (red curve) and its composition form (green) show little difference when $\|\mathbf{x}-\mathbf{y}\|$ becomes large compared to λ. Also note that the latter can be approximated well with the original form of an RBF kernel with a different kernel width $\lambda = 7$ (blue), which means the kernel composition is less likely to induce qualitatively different forms of kernels.

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:

$$\mathbf{\Phi}(\mathbf{\Phi}(\mathbf{x})) \cdot \mathbf{\Phi}(\mathbf{\Phi}(\mathbf{y})) \;=\; e^{-\lambda\|\mathbf{\Phi}(\mathbf{x})-\mathbf{\Phi}(\mathbf{y})\|^2} \;=\; e^{-2\lambda(1-k(\mathbf{x},\mathbf{y}))}. \tag{4.4}$$

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 $\mathbf{\Phi}(\mathbf{x})$. Figure 4.1 explains this point.

Next we consider the ℓ-fold composition in eq. (4.2) 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

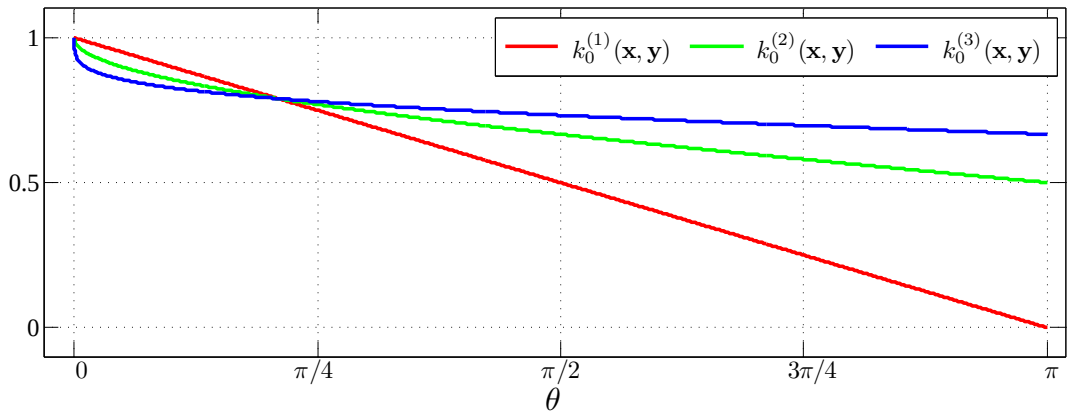


Figure 4.2: Arc-cosine kernels with degree $n=0$ and their composition forms (up to three levels) as a function of θ. In contrast to the RBF kernel in Figure 4.1, new arc-cosine kernels by kernel composition exhibit qualitatively different behavior.

angle between $\mathbf{x}$ and $\mathbf{y}$. For this kernel, it follows that:

$$\begin{aligned}
\boldsymbol{\Phi}(\boldsymbol{\Phi}(\mathbf{x})) \cdot \boldsymbol{\Phi}(\boldsymbol{\Phi}(\mathbf{y})) &= 1 - \frac{1}{\pi} \cos^{-1}\left(\frac{\boldsymbol{\Phi}(\mathbf{x}) \cdot \boldsymbol{\Phi}(\mathbf{y})}{\|\boldsymbol{\Phi}(\mathbf{x})\| \|\boldsymbol{\Phi}(\mathbf{y}))\|} \right) && (4.5) \\
&= 1 - \frac{1}{\pi} \cos^{-1}\left(\frac{k_0(\mathbf{x},\mathbf{y})}{\sqrt{k_0(\mathbf{x},\mathbf{x})\, k_0(\mathbf{y},\mathbf{y})}} \right) && (4.6) \\
&= 1 - \frac{1}{\pi} \cos^{-1}\left(1 - \frac{\theta}{\pi} \right). && (4.7)
\end{aligned}$$

Figure 4.2 shows how the $n=0$ arc-cosine kernels evolve via this kernel composition up to three levels.

More generally, we can work out a recursive formula for the ℓ-fold composition in eq. (4.2). The base case is given by eq. (2.3) for kernels of depth $\ell = 1$ and degree n. Substituting into eq. (2.3), we obtain the construction for kernels of greater depth:

$$k_n^{(\ell+1)}(\mathbf{x},\mathbf{y}) = \frac{1}{\pi} \left[k_n^{(\ell)}(\mathbf{x},\mathbf{x})\; k_n^{(\ell)}(\mathbf{y},\mathbf{y}) \right]^{n/2} J_n\big(\theta_n^{(\ell)}\big), \qquad (4.8)$$

where $\theta_n^{(\ell)}$ is the angle between the images of $\mathbf{x}$ and $\mathbf{y}$ in the feature space induced

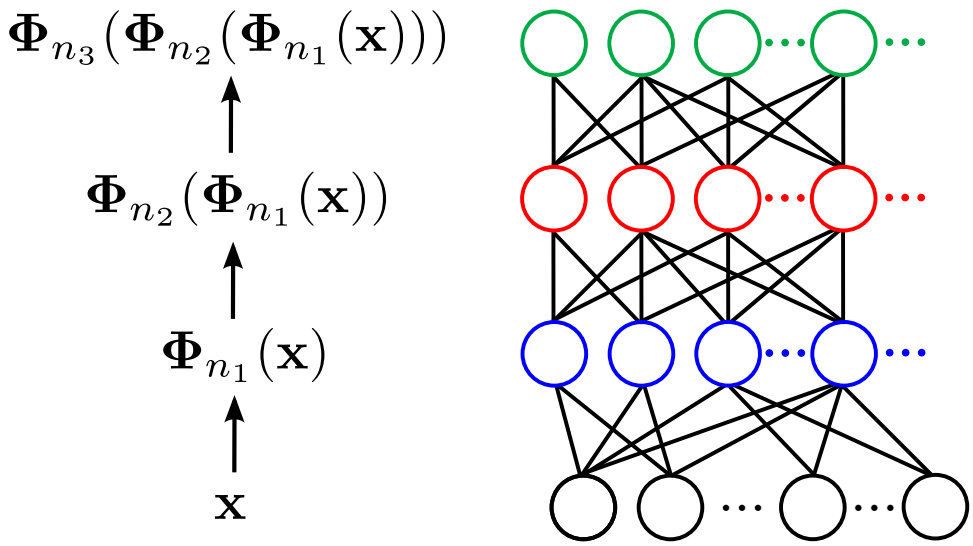


Figure 4.3: Multilayer neural networks modeled by the composition of arc-cosine kernels. Note that different nonlinear mappings (i.e., kernels) can be used at different layers.

by the ℓ-fold composition. In particular, we can write:

$$\theta_n^{(\ell)} \;=\; \cos^{-1}\left(\frac{k_n^{(\ell)}(\mathbf{x},\mathbf{y})}{\sqrt{k_n^{(\ell)}(\mathbf{x},\mathbf{x})\,k_n^{(\ell)}(\mathbf{y},\mathbf{y})}}\right). \tag{4.9}$$

The recursion in eq. (4.8) is simple to compute in practice. The resulting *multilayer* 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. However, it is straightforward to use kernels of different degrees at different levels. We denote compositions of this form by

$$k_{n_1,\ldots,n_\ell}(\mathbf{x},\mathbf{y}) \;=\; \mathbf{\Phi}_{n_\ell}(\ldots\mathbf{\Phi}_{n_1}(\mathbf{x}))\cdot\mathbf{\Phi}_{n_\ell}(\ldots\mathbf{\Phi}_{n_1}(\mathbf{y})), \tag{4.10}$$

where n_i denotes the degree of the kernel used at the ith layer in the composition. Figure 4.3 illustrates this multilayer arc-cosine kernels.

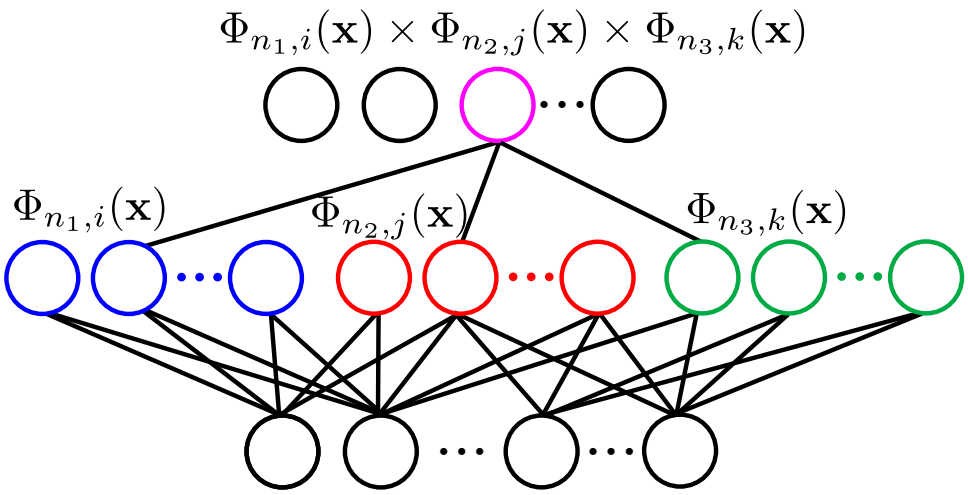


Figure 4.4: Multilayer neural networks modeled by the kernel multiplication. $\mathbf{\Phi}_{n,m}$ means the mth feature induced by the arc-cosine kernel with degree n. Different colors in the intermediate layers encode different nonlinear mappings induced by the corresponding kernels.

4.2.2 Kernel Multiplication

Next we consider the kernels obtained by taking products of arc-cosine kernels. The product operation has much richer possibilities with arc-cosine kernels than polynomial or RBF kernels. In particular, the product of homogeneous polynomial kernels of degrees p_i is simply another polynomial kernel of degree $\sum_i p_i$; the product of RBF kernels with widths σ_i^2 is another RBF kernel with width $(\sum_i \sigma_i^{-2})^{-1}$. By contrast, the products of arc-cosine kernels are not themselves members of the original family.

For m arc-cosine kernels with individual degrees n_i, we denote the product kernel by:

$$k_{n_1 \times \cdots \times n_m}(\mathbf{x}, \mathbf{y}) \;=\; \prod_i^m k_{n_i}(\mathbf{x}, \mathbf{y}). \tag{4.11}$$

The product of m arc-cosine kernels can be interpreted as computing the inner product of outputs from a particular type of *two-layer* neural network. The units in the first layer of this network encode the mapping from the individual kernels in the product. The units in the second layer compute m-wise products of the units in the first layer. Figure 4.4 illustrates this concept. The product of arc-cosine

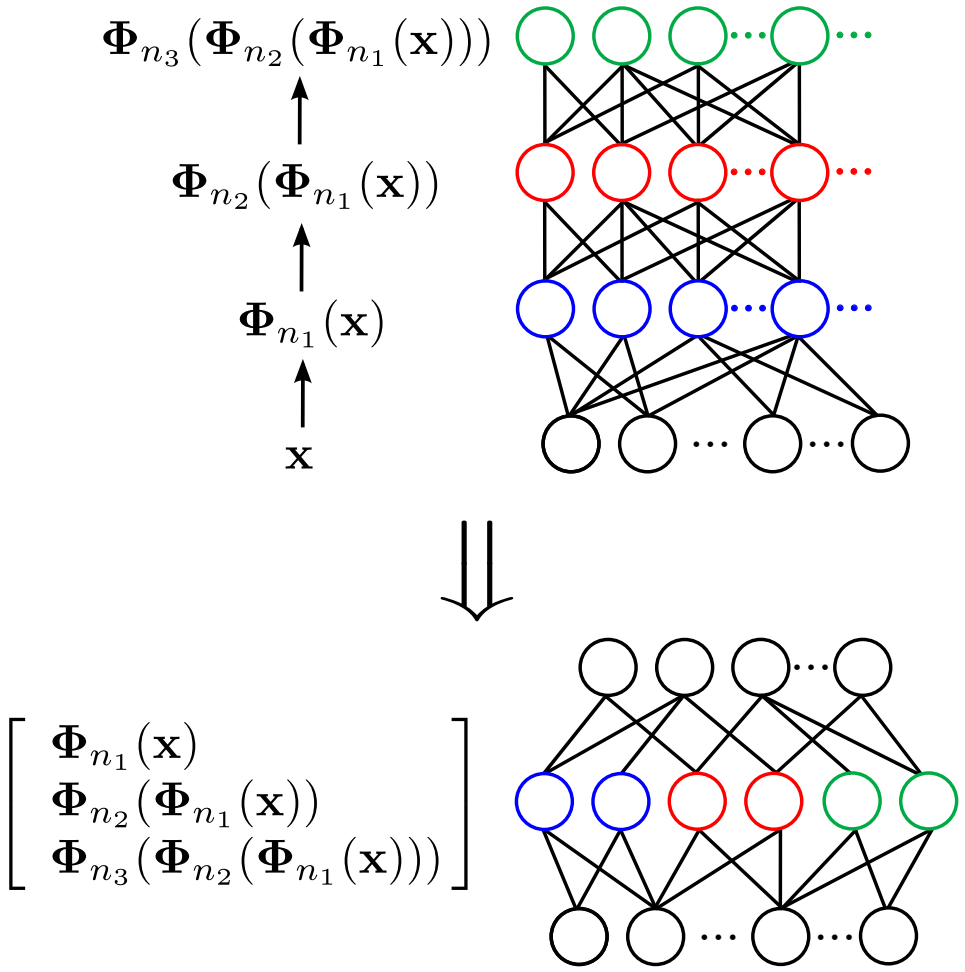


Figure 4.5: Multilayer neural networks modeled by the kernel averaging. The nonlinear mappings (i.e., kernels) at different layers of the multilayer kernel (top) are concatenated as a single feature representation (bottom) via kernel averaging.

kernels of the same degree can also be viewed as the composition of an arc-cosine kernel with a polynomial kernel.

4.2.3 Kernel Averaging

Finally, we consider the kernels formed by averaging. As shown in sections 4.2.1 and 4.2.2, there are (combinatorially) many ways to construct new kernels from compositions and products of arc-cosine kernels. It seems unlikely that any one of these constructions captures all the information needed to classify

the inputs correctly. We can look for a potentially better solution by concatenating the feature vectors obtained from individual kernel maps and computing the inner product in this "stacked" feature space (i.e., the Cartesian product). The averaged kernel performs this computation.

We are particularly interested in the effects of averaging across different multilayer kernels, as constructed in Section 4.2.1. For classification in multilayer neural networks, it is known that the units in intermediate hidden layers can provide useful information for decision-making, often in addition to the units in the final hidden layer. Thus, we hypothesize that averaging multilayer kernels of different "depth" would lead to better performance than the results of any individual kernel. Figure 4.5 illustrates this idea.

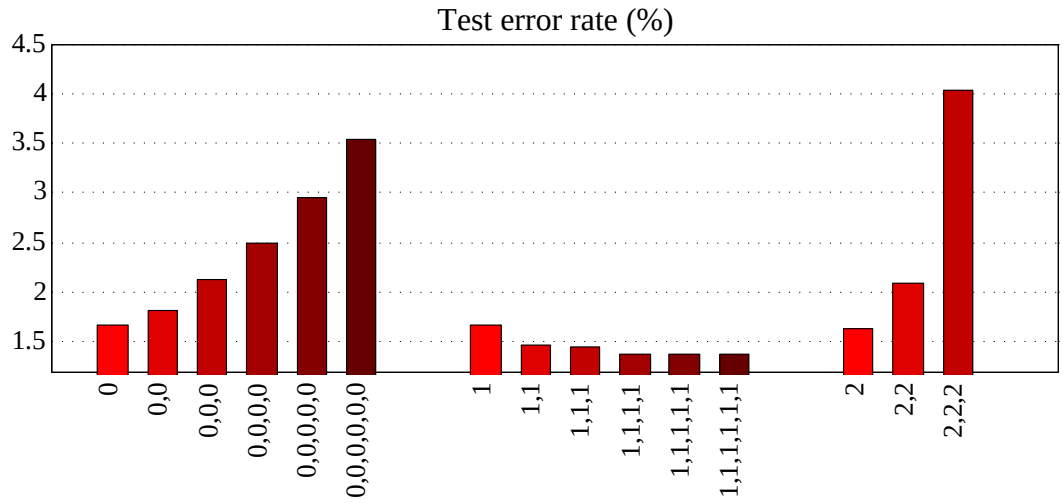
4.3 Experimental Results

4.3.1 Kernel Composition

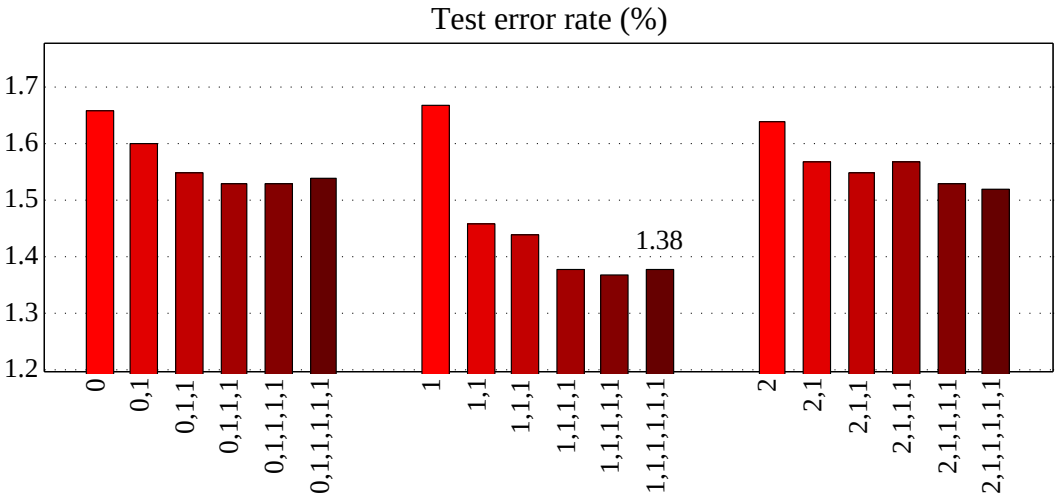
We experimented with the multilayer kernels described in Section 4.2.1, using the composition of arc-cosine kernels of degree $n = 0, 1,$ and 2 from one to six levels of recursion. Figure 4.6 shows the test set error rates on deslanted *MNIST* from arc-cosine kernels of varying degrees (n) and numbers of layers (ℓ). To show 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 experiment, 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.

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 arc-cosine kernel with degree $n = 1$, but generally worsened[1] for the other cases ($n = 0$ and 2). These results led

[1] Also, though not shown in the figure, the arc-cosine kernel with degree $n = 2$ gave essentially



(a) Same-degree multilayer kernels



(b) Mixed-degree multilayer kernels

Figure 4.6: Test error rates on the *MNIST* data 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 degrees using the notation in eq. (4.10). The error rate from the configuration that performed best on held-out set is displayed. The best comparable result is 1.22% from SVMs using polynomial kernels of degree 9 (Decoste and Schölkopf, 2002). See text for details.

random results, with roughly 90% error rates, when composed with itself three or more times.

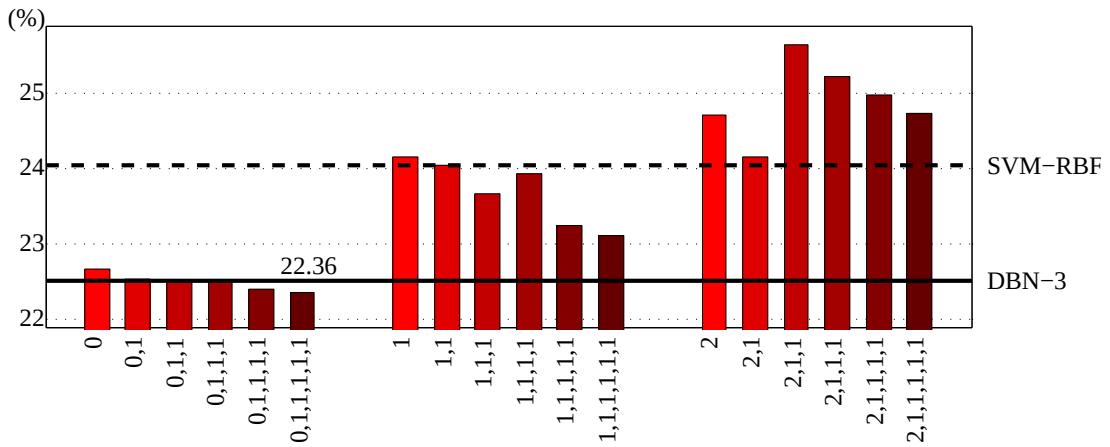


Figure 4.7: Classification error rates on the *Rectangles-image* data set. SVMs with arc-cosine kernels have error rates from 22.36–25.64%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 24.04% for SVMs with RBF kernels and 22.50% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed. See text for details.

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 2.1.1 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. (4.8–4.9). Figure 4.6 shows these sets of results in the bottom panel. In these experiments, we used arc-cosine kernels of different degrees at the first layer of nonlinearity, but only the arc-cosine kernels of degree $n=1$ at successive layers. The best of these kernels yielded a test error rate of 1.38%, comparable to many other results from SVMs (LeCun and Cortes, 1998) on deslanted *MNIST* digits, though not matching the best previous result of 1.22% (Decoste and Schölkopf, 2002). These results also reveal 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

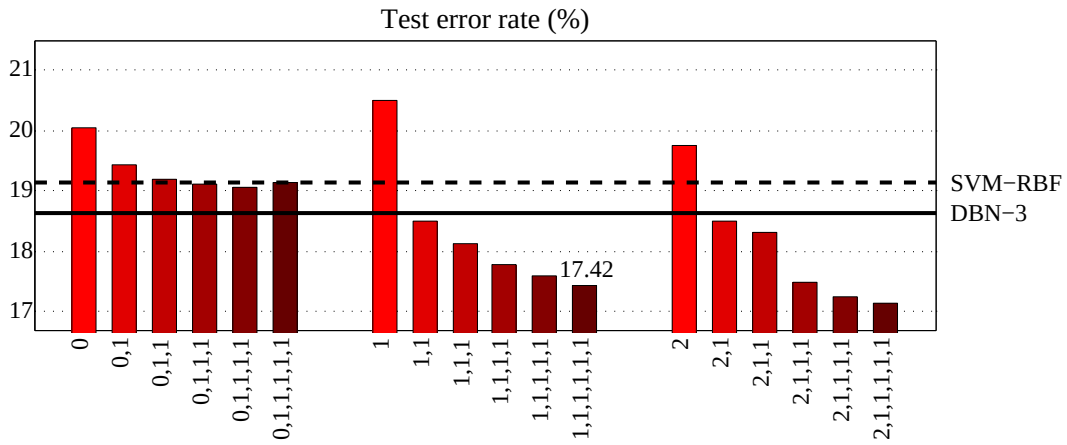


Figure 4.8: Classification error rates on the *Convex* data set. SVMs with arc-cosine kernels have error rates from 17.15–20.51%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 19.13% for SVMs with RBF kernels and 18.63% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed. See text for details.

inherently shallow architectures, this variability is reminiscent of experience with multilayer neural nets.

We explored the effects of multilayer kernels further on data sets in Section 2.2, which were specifically designed to illustrate the advantages of deep architectures. Figures 4.7 and 4.8 show the test set error rates on the *Rectangles-image* and the *Convex* data sets 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). 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 multilayer arc-cosine kernels are quite straightforward to train. Unlike SVMs with

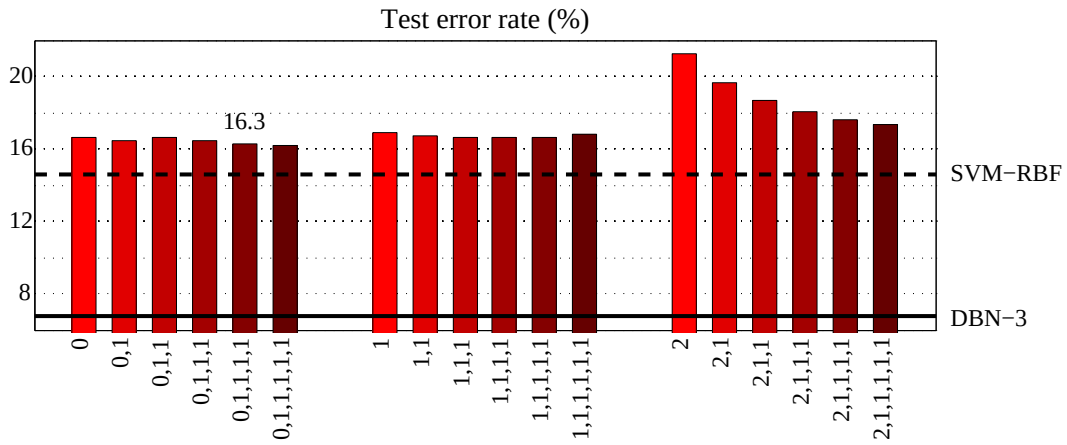


Figure 4.9: Classification error rates on the *MNIST-rand* data set. SVMs with arc-cosine kernels have error rates from 16.14–21.27%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 14.58% for SVMs with RBF kernels and 6.73% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed. See text for details.

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 multilayer 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.

We experimented last on two challenging data sets in multiway classification. Like the data sets in the previous experiments, 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*

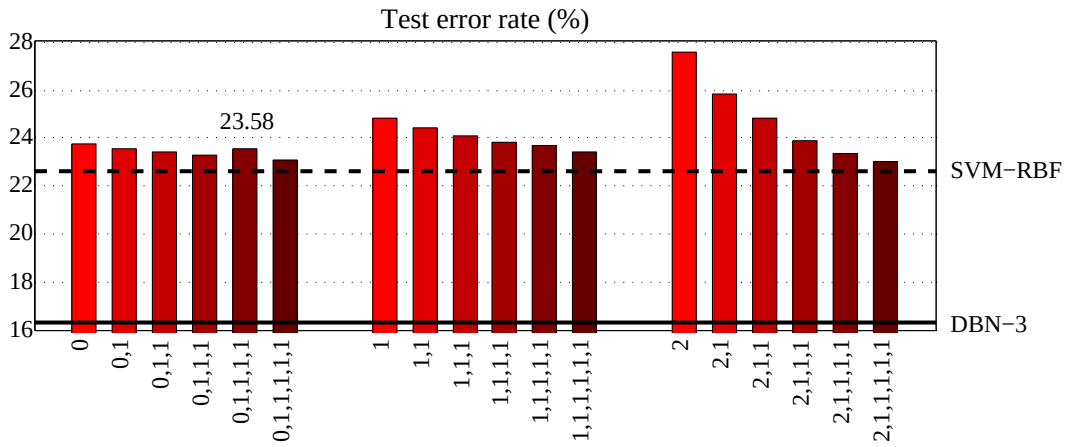


Figure 4.10: Classification error rates on the *MNIST-image* data set. SVMs with arc-cosine kernels have error rates from 23.03–27.58%. Results are shown for kernels of varying degrees (n) and numbers of layers (ℓ). The best previous results are 22.61% for SVMs with RBF kernels and 16.31% for deep belief nets (Larochelle et al., 2007). The error rate from the configuration that performed best on held-out set is displayed. See text for details.

images of handwritten digits. (See Section 2.2 for details.)

Figures 4.9 and 4.10 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. In particular, while deep belief nets can adaptively extract useful features from all the training examples on this *multiclass* problem by unsupervised pre-training methods, SVMs for each pair of different classes (i.e., one-versus-one approach) can only consider the features extracted from limited sets of examples (i.e., two classes in comparison). Even with the so-called one-versus-all approach, this problem remains because SVMs are only allowed to consider features implicitly generated by kernels, but it is hard to adapt these features explicitly according to

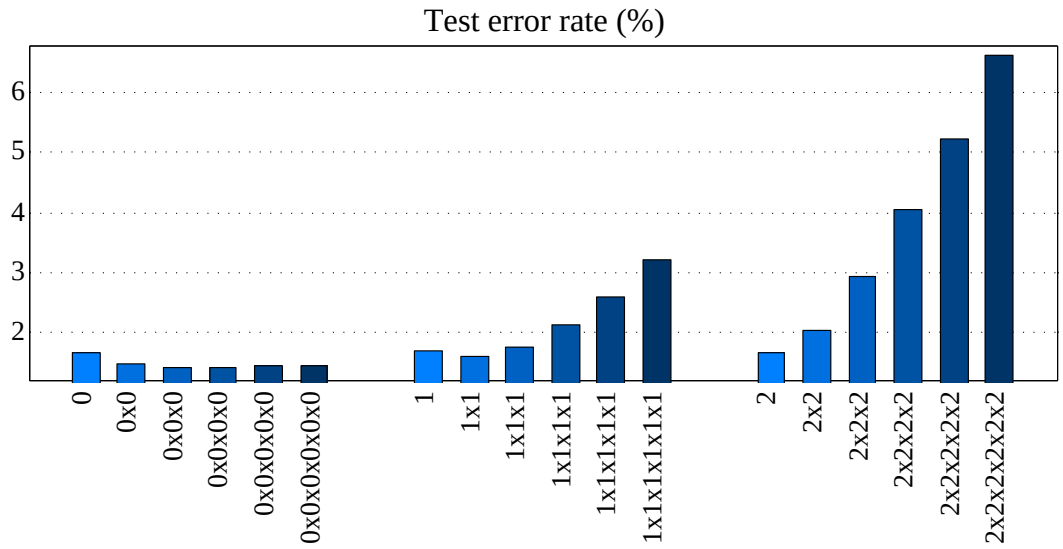
certain useful criteria like the reconstruction error in unsupervised pre-training of deep architectures.

In addition, 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. The rotational invariance of arc-cosine kernel could also be explicitly broken by integrating in eq. (2.1) 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.

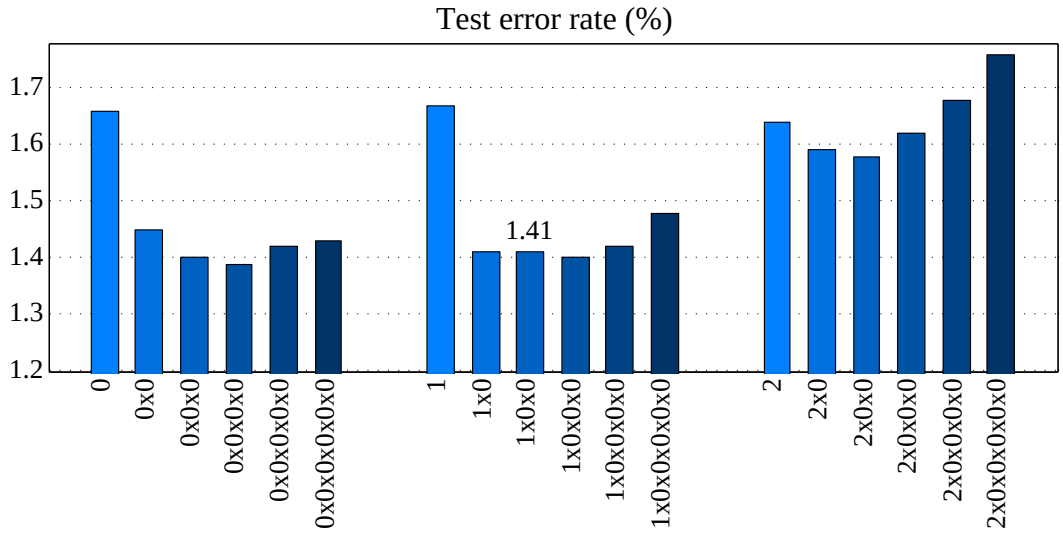
It is less clear why SVMs with arc-cosine kernels perform worse on these data sets than SVMs with RBF kernels. Our results in multiway classification were obtained by a naive combination of binary SVM classifiers, which may be a confounding effect. We also examined the SVM error rates on the 45 sub-tasks 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 4.9 and 4.10 might suggest. In future work, we may explore arc-cosine kernels in a more principled framework for multiclass SVMs (Crammer and Singer, 2001).

4.3.2 Kernel Multiplication

Our second set of experiments evaluated the product kernels described in Section 4.2.2. We first experimented with product kernels that used the arc-cosine kernels of the same degrees $n=0,1,2$ as individual factors (up to six) in the kernel multiplication. Figure 4.11 shows the error rates on the *MNIST* data set from SVMs with these kernels. From these experiments, we observed similar trends to the case of multilayer kernels in Section 4.3.1: the performance generally improved with increasing number of factors for the arc-cosine kernels with degree $n=0$, but generally worsened for the other cases. Interestingly, this trend is consistent with our previous hypothesis—preserving the magnitude information is important for

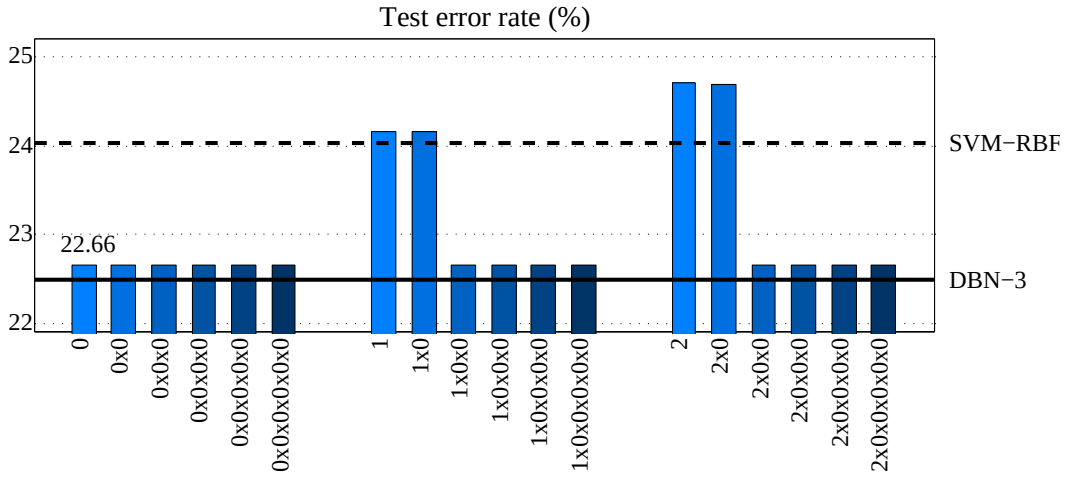


(a) Same-degree product kernels

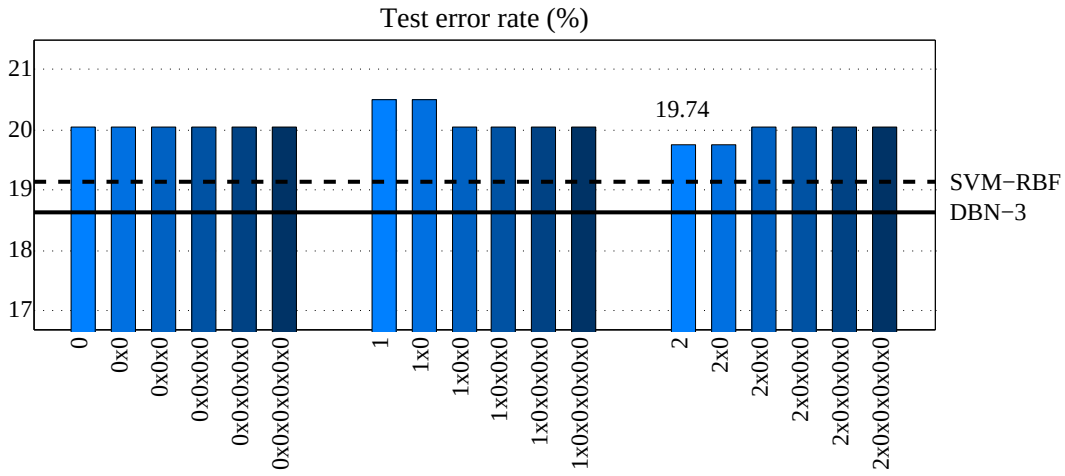


(b) Mixed-degree product kernels

Figure 4.11: Test error rates on the *MNIST* data set from SVMs with product arc-cosine kernels. The figure shows results for kernels of varying degrees (n) and numbers of factors (m). In one set of experiments, the product kernels were constructed by multiplying arc-cosine kernels of the same degree. In another set of experiments, only arc-cosine kernels of degree $n = 0$ were used as new factors. Other formatting details follow Figure 4.6.



(a) *Rectangles-image*

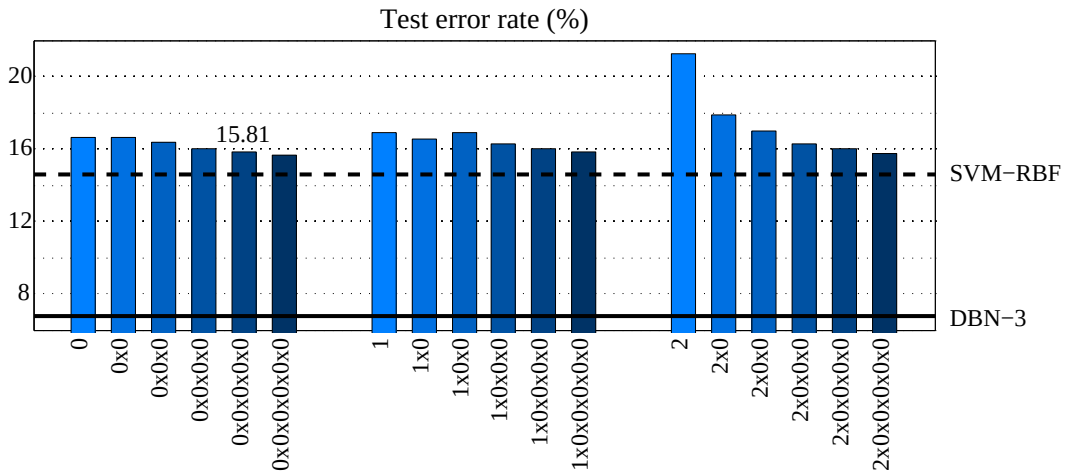


(b) *Convex*

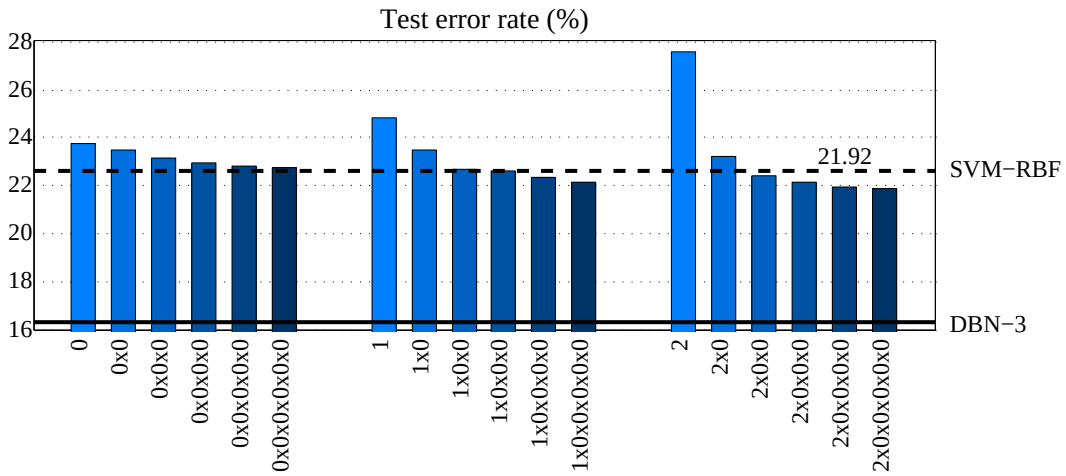
Figure 4.12: Classification error rates on the *Rectangles-image* (top) and *Convex* (bottom) data sets with product arc-cosine kernels. Results are shown for kernels of varying degrees (n) and numbers of factors (m). The error rate from the configuration that performed best on held-out set is displayed.

the classification performance. In particular, we can easily see that the multiplication by arc-cosine kernels of degree $n = 0$ does not affect the norms of mapped inputs in feature space:

$$k_{n_1 \times 0 \times 0 \ldots \times 0}(\mathbf{x}, \mathbf{x}) \;=\; k_{n_1}(\mathbf{x}, \mathbf{x}). \tag{4.12}$$



(a) *MNIST-rand*



(b) *MNIST-image*

Figure 4.13: Classification error rates on the *MNIST-rand* (top) and *MNIST-image* (bottom) data sets with product arc-cosine kernels. Same format as Figure 4.12.

Based on this observation, we focused only on the case where at most one of the arc-cosine kernels in this product had non-zero degree. Figure 4.11 shows these sets of results in the bottom panel where we find such construction often leads to much improved error rates, similar to the trend in Figure 4.6. We also extensively evaluated product kernels in this form on the other data sets (*Rectangles-image*, *Convex*, *MNIST-rand*, and *MNIST-image*) from the deep learning bench-

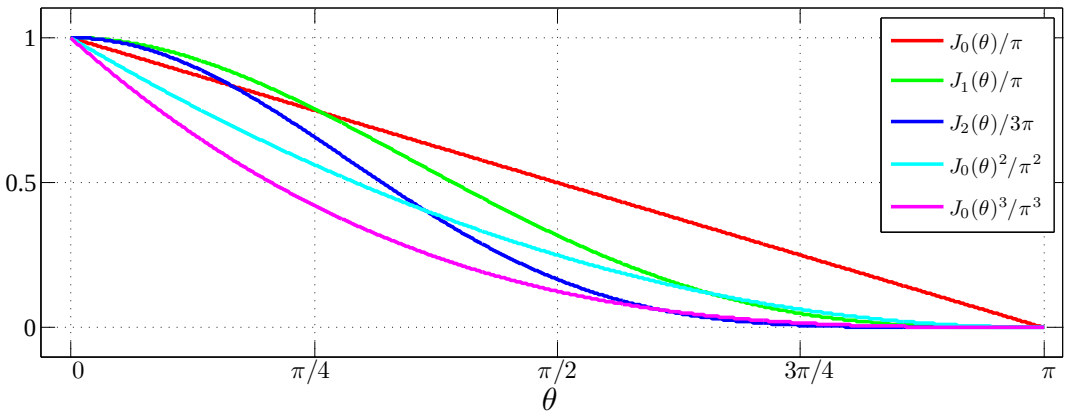


Figure 4.14: Behavior of the function $J_n(\theta)$ in eq. (2.4) including the case of multiplying by certain powers of the arc-cosine kernel of degree $n=0$.

marks (Larochelle et al., 2007). Figures 4.12–4.13 show the results from these experiments, where we find again that product kernels lead to consistent improvement in many cases. Not surprisingly, though, the gains from product kernels eventually saturate and/or reverse themselves as more products are accumulated.

We can only speculate why the product operation in eq. (4.11) leads to such significant improvement. Consider the plot in Figure 4.14, which shows the effect of multiplying by powers of the arc-cosine kernel of degree $n = 0$. Note that $(J_0(\theta)/\pi)^m$ has an increasingly negative slope at $\theta = 0$ with increasing m. Intuitively, large negative slopes at $\theta = 0$ indicate that nearby inputs (with small angles between them) are mapped to distant points in feature space. Thus, product kernels with powers of $k_0(\mathbf{x}, \mathbf{y})$ warp the input space by repelling nearby inputs while preserving their individual norms. Perhaps the classification results improve under this operation because the increased sensitivity to angular displacements in the input space leads to better separation of different (but easily confused) classes.

4.3.3 Kernel Averaging

Our last set of experiments evaluated kernels obtained by averaging. We experimented by averaging kernels over many different types of ensembles: arc-cosine kernels of different degree, multilayer kernels of different depth, product kernels

of different powers, and various combinations of the above. We also experimented with many forms of preprocessing (e.g., rescaling, centering) before averaging. The results of these experiments were uniformly negative: *in most cases, SVMs with averaged kernels showed no improvement or yielded worse error rates.* Rather than expanding the feature space and enabling individual kernels to complement each other, the averaging operation seemed only to dilute the effectiveness of the best individual kernel.

4.4 Discussion

In this chapter, we demonstrated how to extend the family of arc-cosine kernels using kernel composition, multiplication, and averaging. The resulting arc-cosine kernels could be interpreted in the context of multilayer neural networks, and we obtained promising experimental results on certain classification tasks that could support this direction of research.

As partly discussed in Section 2.3, several researchers have also explored connections between kernel machines and deep learning. 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 and 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 4.2.1.

Our work was also motivated by the ongoing debate over deep versus shallow architectures (Bengio and 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.

For future research directions, 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. This work was motivated by the weakness of SVMs in the noisy multiclass data sets in Section 4.3.1. In the next chapter, we show a proof-of-concept model for this direction by combining ideas from kernel principal component analysis (Schölkopf et al., 1998), discriminative feature selection (Guyon and Elisseeff, 2003), and large margin nearest neighbor classification (Weinberger and Saul, 2009). Finally, we are also interested in more effective schemes to combine arc-cosine kernels. In particular, multiple kernel learning (Lanckriet et al., 2004; Rakotomamonjy et al., 2008; Bach, 2009; Cortes et al., 2010) is an interesting generalization of the kernel averaging, and we hope it can address the issues raised in Section 4.3.3 and eventually provide compelling options for kernel combination.

Chapter 4, in part, is a reprint of the material as it appears in Advances in Neural Information Processing Systems 22. Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 4, in part, is a reprint of the material as it appears in Neural Computation 22(10). Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 5

Multilayer Kernel Machines

5.1 Introduction

In this chapter, we explore how to construct new hierarchical kernel machines to overcome the weakness of SVMs. As noted in Section 4.3.1, SVMs with multilayer arc-cosine kernels have difficulties in fully utilizing training sets of multiclass problems in unsupervised fashion. Also, it is not obvious for the multilayer arc-cosine kernels to suppress irrelevant input features directly because it may be hard to filter out such noise when the kernel operation intermingles it with other useful features, which can be even problematic for iterated kernel operations in the multilayer arc-cosine kernels.

In fact, these shortcomings of SVMs turn out to be the rationale behind using deep architectures. Specifically, deep architectures have been benefited much from the unsupervised pre-training stage in terms of training efficiency and test performance (Erhan et al., 2010). Also, deep architectures can adapt all the weight parameters in the network (including the ones linked to input features) with respect to the tasks they solve. In contrast, SVMs with multilayer arc-cosine kernels can adapt only the weight of the final decision boundary.

Inspired by such desirable properties of the deep architectures, we introduce an alternative kernel-based model, called a multilayer kernel machine (MKM). The main idea is to incorporate unsupervised dimensionality reduction and supervised feature selection techniques into the multilayer arc-cosine kernels; the former of

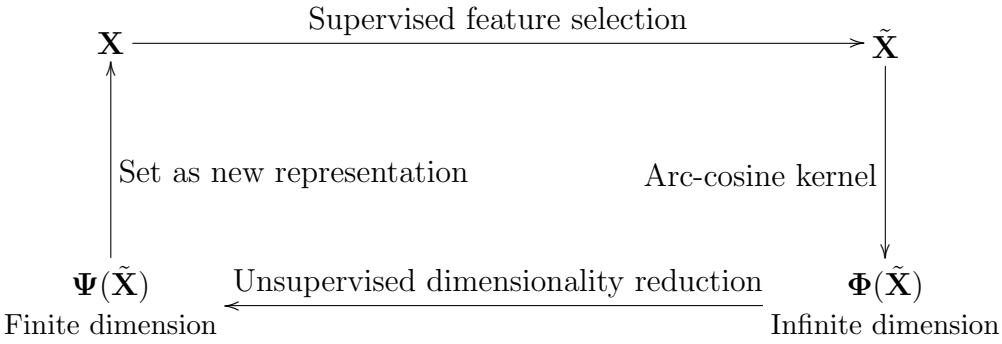


Figure 5.1: Main concept in multilayer kernel machines (MKMs). Input data $\mathbf{X}$ (upper left) are transformed sequentially by supervised feature selection, arc-cosine kernel, and unsupervised dimensionality reduction. This cycle is repeated multiple times to construct an MKM.

which integrates unsupervised learning schemes into arc-cosine kernels while the latter aims to suppress irrelevant features.

5.2 Principles of Multilayer Kernel Machines

5.2.1 Main Idea

Figure 5.1 illustrates the main concept of MKMs: first, the original input data $\mathbf{X}$ are filtered to contain only the features relevant to the label information; second, the resulting data $\tilde{\mathbf{X}}$ are fed to arc-cosine kernels which (implicitly) construct infinite dimensional representation $\mathbf{\Phi}(\tilde{\mathbf{X}})$; third, unsupervised dimensionality reduction is applied on $\mathbf{\Phi}(\tilde{\mathbf{X}})$ in feature space and finite dimensional representation $\mathbf{\Psi}(\tilde{\mathbf{X}})$ is obtained as a result; fourth, we set $\mathbf{\Psi}(\tilde{\mathbf{X}})$ as new input features and repeat the previous steps multiple, say L, times to construct L-layer feature hierarchy of MKMs. Finally, we feed the features at the last layer to any off-the-shelf classifiers to make a final decision.

Note that our goal in this work is to demonstrate kernel methods can also be extended to hierarchical structures without requiring complicated machinery, but simply with leveraging existing machine learning methods for feature selection,

dimensionality reduction, and multiclass classification. Therefore, it is definitely possible to implement this high-level concept of MKMs by choosing different machine learning techniques for certain components of the model. Our implementation of the concept is presented in Section 5.2.3 and we also briefly discuss other alternative configurations in Section 5.3.

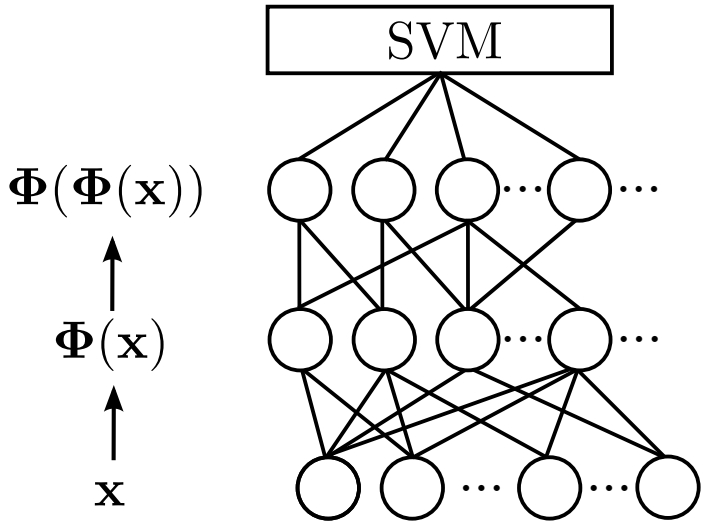
5.2.2 Comparison with Multilayer Arc-cosine Kernels

Multilayer kernel machines were inspired by multilayer arc-cosine kernels in Section 4.2.1 and actually they can be regarded as a denoised version of the multilayer arc-cosine kernels.

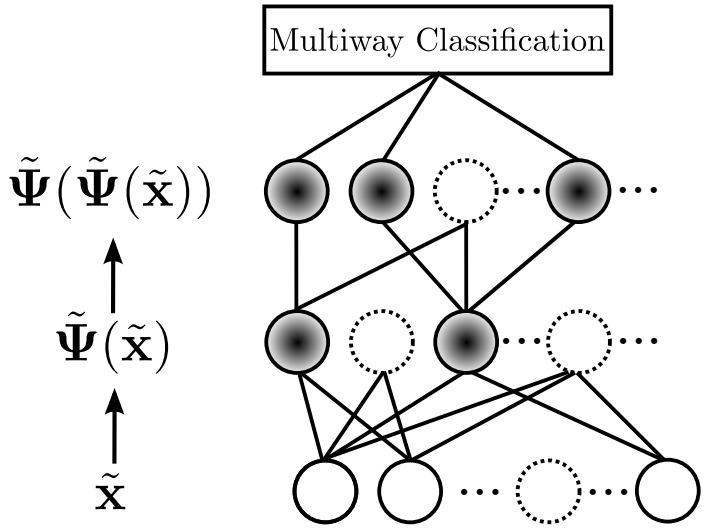
Figure 5.2 compares these two architectures. Basically, both models apply arc-cosine kernels $\mathbf{\Phi}(\cdot)$ at each layer to sequentially transform the input data to the final representations. While these final features are just fed to SVMs in multilayer arc-cosine kernels, MKMs have additional procedures to modify the features at every layer. Most significantly, MKMs reduce the dimensionality of the kernel representation to decrease noise and also provide compact representations of the data. This unsupervised dimensionality reduction stage turns an infinite dimensional representation $\mathbf{\Phi}(\cdot)$ into a finite dimensional representation $\mathbf{\Psi}(\cdot)$, which has two effects in the MKM architecture as follows.

First, it becomes possible to explicitly evaluate individual features based on their relevance to label information—which may not be straightforward for implicit representation of the data in the kernel space—and remove irrelevant ones among them. In Figure 5.2, such pruned features are denoted as dotted circles, which do not have upward weights to the next layer.

Second, we can use any general classifiers at the final layer other than nonlinear SVMs because again the data at the final layer are not expressed implicitly in the kernel space. For example, we use Mahalanobis distance metric learning for this purpose in our implementation, which may be hard to apply with the "kernel trick". In general, this means we are provided more flexibility in incorporating MKMs with other parts of a bigger learning system.



(a) SVMs with multilayer arc-cosine kernels.



(b) Multilayer kernel machines.

Figure 5.2: Comparison between (a) multilayer arc-cosine kernels and (b) multilayer kernel machines (MKMs). As in multilayer arc-cosine kernels, MKMs use arc-cosine kernels $\mathbf{\Phi}(\cdot)$ repeatedly to build a kernel hierarchy. In MKMs, however, irrelevant features (dotted circles) are pruned at every layer, and the dimensionality of data is reduced by $\mathbf{\Psi}(\cdot)$ (shaded circles) at every layer except the original input layer. For the final classification layer, MKMs can use any general classifiers, not necessarily SVMs.

Prune uninformative features from inputs $\mathbf{X}_0$ using mutual information.
$\ell = 0$.
while $\ell < L$ **do**
  Compute $k(\mathbf{X}_\ell, \mathbf{X}_\ell)$. (e.g., arc-cosine kernel)
  Extract principal components $\mathbf{V}_\ell$ in the feature space induced by $k(\mathbf{X}_\ell, \mathbf{X}_\ell)$.
  $\mathbf{X}_{\ell+1} = \mathbf{V}_\ell \cdot \mathbf{\Phi}(\mathbf{X}_\ell)$.
  Prune uninformative components from $\mathbf{X}_{\ell+1}$.
  $\ell = \ell + 1$.
end
Learn Mahalanobis distance metric for k-NN classification of $\mathbf{X}_L$.

Figure 5.3: Training procedures for multilayer kernel machines. See text for details.

5.2.3 Implementation

In this section, we show how we have implemented MKMs by supplementing multilayer arc-cosine kernels with kernel principal components analysis (kernel PCA) (Schölkopf et al., 1998) and feature selection (Guyon and Elisseeff, 2003) at intermediate hidden layers and large-margin nearest neighbor classification (Weinberger and Saul, 2009) at the final output layer. Specifically, for L-layer MKMs, we consider the training procedure in Figure 5.3.

The individual steps in this procedure are well-established methods; only their combination is new. While many other approaches are worth investigating, our positive results from the above procedure provide a first proof-of-concept. We discuss each of these steps in greater detail below.

Kernel PCA

Deep learning in MKMs is achieved by iterative applications of kernel principal components analysis (Schölkopf et al., 1998). This use of kernel PCA was suggested over a decade ago (Schölkopf et al., 1996) and more recently inspired by the pre-training of deep belief nets by unsupervised methods. In MKMs, the outputs (or features) from kernel PCA at one layer are the inputs to kernel PCA

at the next layer. However, we do not strictly transmit each layer's top principal components to the next layer; some components are discarded if they are deemed uninformative. While any nonlinear kernel can be used for the layerwise PCA in MKMs, arc-cosine kernels are natural choices to mimic the computations in large neural nets.

Feature Selection

The layers in MKMs are trained by interleaving a supervised method for feature selection with the unsupervised method of kernel PCA. The feature selection is used to prune away uninformative features at each layer in the MKM (including the zeroth layer which stores the raw inputs). Intuitively, this feature selection helps to focus the unsupervised learning in MKMs on statistics of the inputs that actually contain information about the class labels. We prune features at each layer by a simple two-step procedure that first ranks them by estimates of their mutual information, then truncates them using cross-validation. More specifically, in the first step, we discretize each real-valued feature and construct class-conditional and marginal histograms of its discretized values; then, using these histograms, we estimate each feature's mutual information with the class label and sort the features in order of these estimates (Guyon and Elisseeff, 2003). In the second step, considering only the first w features in this ordering, we compute the error rates of a basic k-nearest neighbor (k-NN) classifier using Euclidean distances in feature space. We compute these error rates on a held-out set of validation examples for many values of k and w and record the optimal values for each layer. The optimal w determines the number of informative features passed onto the next layer; this is essentially the *width* of the layer. In practice, we varied k from 1 to 15 and w from 10 to 300; though exhaustive, this cross-validation can be done quickly and efficiently by careful bookkeeping. Note that this procedure determines the architecture of the network in a greedy, layer-by-layer fashion.

Distance Metric Learning

Test examples in MKMs are classified by a variant of k-NN classification on the outputs of the final layer. Specifically, we use large margin nearest neighbor (LMNN) classification (Weinberger and Saul, 2009) to learn a Mahalanobis distance metric for these outputs, though other methods are equally viable (Goldberger et al., 2005). The use of LMNN is inspired by the supervised fine-tuning of weights in the training of deep architectures (Bengio et al., 2007). In MKMs, however, this supervised training only occurs at the final layer (which underscores the importance of feature selection in earlier layers). LMNN learns a distance metric by solving a problem in semidefinite programming; one advantage of LMNN is that the required optimization is convex. Test examples are classified by the energy-based decision rule for LMNN (Weinberger and Saul, 2009), which was itself inspired by earlier work on multilayer neural nets (Chopra et al., 2005).

5.3 Experimental Results

We evaluated MKMs on the two multiclass data sets *MNIST-rand* and *MNIST-image* from deep learning benchmarks (Larochelle et al., 2007) in previous sections, which exhibited the largest performance gap between deep architectures and shallow ones including SVMs with multilayer arc-cosine kernels in Section 4.3.1.

We trained MKMs with arc-cosine kernels in each layer. For each data set, we initially withheld the last 2000 training examples as a validation set. Performance on this validation set was used to determine each MKM's architecture, as described in the previous section. Once these parameters were set by cross-validation, we re-inserted the validation examples into the training set and used all 12000 training examples for feature selection and distance metric learning. For kernel PCA, we were limited by memory requirements to processing only 6000 out of 12000 training examples. We chose these 6000 examples randomly, but repeated each experiment five times to obtain a measure of average performance. The results we report for each MKM are the average performance over these five runs.

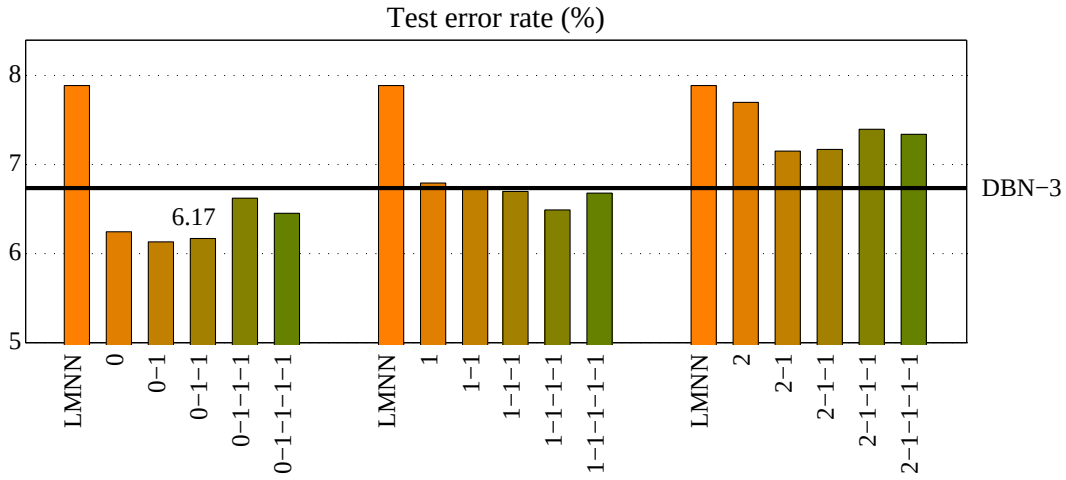


Figure 5.4: Classification error rates on the *MNIST-rand* data set for MKMs with different kernels and numbers of layers. The notation n_1–n_2–...–n_L in the horizontal axis means an MKM whose ith layer uses the arc-cosine kernel of degree n_i, while "LMNN" corresponds to the result of using LMNN with feature selection only (but without arc-cosine kernels or kernel PCA). MKMs with arc-cosine kernel have error rates from 6.13–7.69%. The error rate from the configuration that performed best on held-out set is displayed. The best previous results are 14.58% for SVMs with RBF kernels (not shown) and 6.73% for deep belief nets (Larochelle et al., 2007). See text for details.

Figures 5.4 and 5.5 show the test set error rates of MKMs with different kernels and numbers of layers. For reference, we also show the best previously reported results (Larochelle et al., 2007) using traditional SVMs with RBF kernels and deep belief nets with three layers. MKMs perform significantly better than shallow architectures such as SVMs with RBF kernels or LMNN with feature selection only (reported as "LMNN"). Compared to deep belief nets, the leading MKMs obtain slightly lower error rates on one data set and slightly higher error rates on another. MKMs worked best with arc-cosine kernels of degree $n=0$ and $n=1$. The kernel of degree $n=2$ performed less well in MKMs.

Finally, we briefly summarize experiments on MKMs for further analysis. In particular, we attempted to evaluate the individual contributions to performance from feature selection and LMNN classification. Feature selection helped significantly on the *MNIST-image* data set, but only slightly on the *MNIST-rand* data

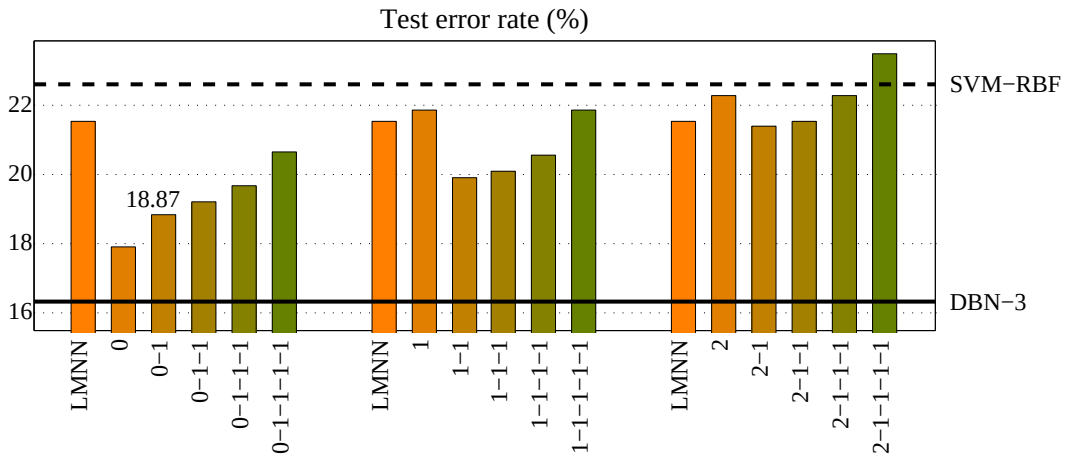


Figure 5.5: Classification error rates on the *MNIST-image* data set for MKMs with different kernels and numbers of layers. Same horizontal axis as Figure 5.4. MKMs with arc-cosine kernel have error rates from 17.91–23.50%. The error rate from the configuration that performed best on held-out set is displayed. The best previous results are 22.61% for SVMs with RBF kernels and 16.31% for deep belief nets (Larochelle et al., 2007). See text for details.

set. On the other hand, LMNN classification in the output layer yielded consistent improvements over basic k-NN classification provided that we used the energy-based decision rule (Weinberger and Saul, 2009). Also, we tested the idea of using RBF kernels in place of arc-cosine kernels to build MKMs (even though arc-cosine kernels are natural choices for hierarchical structure). It turned out, however, MKMs with RBF kernels were difficult to train due to the sensitive dependence on kernel width parameters. It was extremely time-consuming to cross-validate the kernel width at each layer of the MKM. We only obtained meaningful results for one and two-layer MKMs with RBF kernels, which were still inferior to MKMs with arc-cosine kernels.

5.4 Discussion

In this chapter, we explored a novel approach to use kernel methods in hierarchical fashion. We showed how to train deep kernel-based architectures by a simple combination of supervised and unsupervised methods. Using the arc-

cosine kernels as the main part of the model, these multilayer kernel machines (MKMs) performed very competitively on multiclass data sets designed to foil shallow architectures (Larochelle et al., 2007).

Despite such state-of-the-art performance, MKMs still have rooms for further improvement. In particular, we discovered the greedy choice of hyperparameters (e.g., network structure) was somewhat prone to overfitting to training examples, and especially this was misleading for the *MNIST-image* data set. For example, the performance of MKMs had higher correlation with the number of layers in training set, but we found this trend was not followed closely in test set as shown in Figure 5.5. It seems overfitted choice of hyperparameters at lower layers has limited the possibilities for the upper layers to produce meaningful features.

Besides such hyperparameter issue, the feature selection step of MKMs was also revealed to be rather inefficient because we assumed the independence of features given the label, which is not true in practice. As a result, the criteria by which the features were selected might be biased towards individual features having high mutual information with labels, even when some of them provided redundant information (e.g., nearby pixels in images). Considering that m best features are not the best m features in general, we need a more sophisticated strategy for feature selection in MKMs.

Chapter 5, in part, is a reprint of the material as it appears in Advances in Neural Information Processing Systems 22. Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 6

Analysis of Arc-cosine Kernels with Differential Geometry

6.1 Introduction

Given their competitive results from the previous chapters, it seems worth exploring theoretical properties of arc-cosine kernels further. In this chapter, we investigate the geometric properties of arc-cosine kernels in much greater detail. Specifically, we analyze the geometry of surfaces in Hilbert space that are induced by these kernels. These surfaces are the images of the input space under the implicit nonlinear mapping performed by the kernel; see Figure 6.1. Our analysis yields a richer understanding of the geometry of these surfaces (and by association, the nonlinear transformations parameterized by large neural networks). We also compare and contrast our results to those previously obtained for RBF and polynomial kernels (Amari and Wu, 1999; Burges, 1999).

As one important theoretical contribution, our analysis shows that arc-cosine kernels of different degrees have qualitatively different geometric properties. In particular, for some kernels in this family, the surface in Hilbert space is described by a curved Riemannian manifold; for another kernel, this surface is flat, with zero intrinsic curvature; finally, for the simplest member of the family, this surface cannot be described as a manifold at all. It seems that the family of arc-

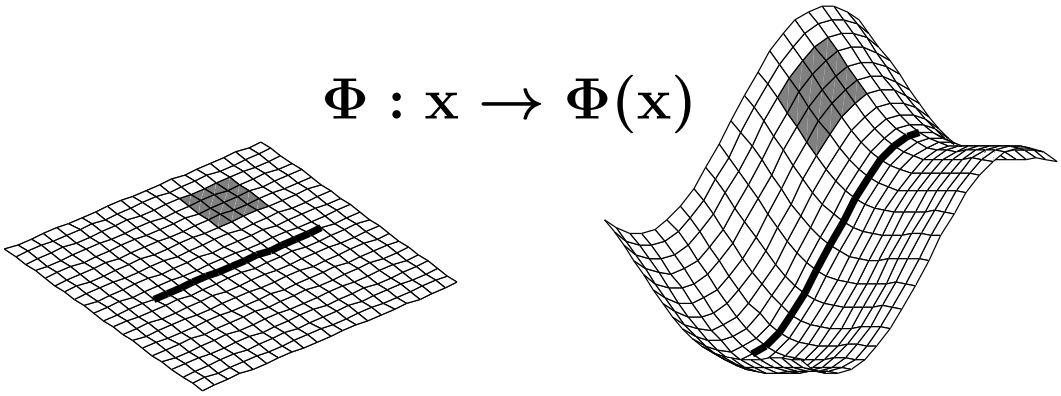


Figure 6.1: The kernel function induces a mapping from the input space into a nonlinear feature space. We can study the geometry of this surface in feature space—for example, asking how arc lengths and volume elements transform under this mapping.

cosine kernels exhibits a larger variety of behaviors than other popular families of kernels.

One preliminary, but interesting, result is that for certain arc-cosine kernels, the implicit mapping into feature space preserves local distances and volume elements. In Section 6.3, we show how the geometric properties of these kernels open up new directions for deep learning and independent component analysis (ICA) (Jutten and Karhunen, 2004). In particular, we use these kernels to define nonlinear probabilistic models that mimic the transformations of multilayer neural networks but do not involve complicated Jacobians.

6.2 Analysis

In this section, we examine the properties of arc-cosine kernels using tools from differential geometry (Amari and Wu, 1999; Burges, 1999). Specifically, we analyze the geometry of surfaces in Hilbert space that are induced by these kernels. When this geometry is described by a Riemannian manifold, we derive results for the metric, curvature, and volume element. We also examine a kernel in this family that does not admit such an interpretation.

6.2.1 Riemannian Geometry

We can understand the family of arc-cosine kernels better by analyzing the geometry of surfaces in Hilbert space. For surfaces that can be described as Riemannian manifolds, Burges (1999) and Amari and Wu (1999) showed how to derive the metric, volume element, and curvature directly from the kernel function. In this section, we use these methods to study arc-cosine kernels of degree $n \geq 1$. As some of the calculations are lengthy, we sketch the main results here while providing more detailed derivations in Appendix D.

Metric

We briefly review the relation of the metric to the kernel function $k(\mathbf{x}, \mathbf{y})$. Consider the surface in Hilbert space parameterized by the input coordinates x_μ. The line element on the surface is given by:

$$\begin{aligned} ds^2 &= \|\mathbf{\Phi}(\mathbf{x}+\mathbf{dx}) - \mathbf{\Phi}(\mathbf{x})\|^2 \\ &= k(\mathbf{x}+\mathbf{dx}, \mathbf{x}+\mathbf{dx}) - 2k(\mathbf{x}, \mathbf{x}+\mathbf{dx}) + k(\mathbf{x}, \mathbf{x}). \end{aligned} \tag{6.1}$$

We identify the metric $g_{\mu\nu}$ by expanding the right hand side to second order in the displacement $\mathbf{dx}$:

$$ds^2 = k(\mathbf{x}, \mathbf{x}) + k(\mathbf{x}+\mathbf{dx}, \mathbf{x}+\mathbf{dx}) - 2k(\mathbf{x}, \mathbf{x}+\mathbf{dx}) \tag{6.2}$$

$$= k(\mathbf{x}, \mathbf{x}) + k(\mathbf{x}, \mathbf{x}) + \sum_\mu dx_\mu \frac{\partial}{\partial x_\mu} k(\mathbf{x}, \mathbf{x}) + \frac{1}{2!} \sum_{\mu\nu} dx_\mu dx_\nu \frac{\partial}{\partial x_\mu} \frac{\partial}{\partial x_\nu} k(\mathbf{x}, \mathbf{x}) \tag{6.3}$$

$$-2 \left(k(\mathbf{x}, \mathbf{x}) + \sum_\mu dx_\mu \left[\frac{\partial}{\partial y_\mu} k(\mathbf{x}, \mathbf{y}) \right]_{\mathbf{y}=\mathbf{x}} + \frac{1}{2!} \sum_{\mu\nu} dx_\mu dx_\nu \left[\frac{\partial}{\partial y_\mu} \frac{\partial}{\partial y_\nu} k(\mathbf{x}, \mathbf{y}) \right]_{\mathbf{y}=\mathbf{x}} \right)$$

$$= \frac{1}{2} \sum_{\mu\nu} dx_\mu dx_\nu \frac{\partial}{\partial x_\mu} \frac{\partial}{\partial x_\nu} k(\mathbf{x}, \mathbf{x}) - \sum_{\mu\nu} dx_\mu dx_\nu \left[\frac{\partial}{\partial y_\mu} \frac{\partial}{\partial y_\nu} k(\mathbf{x}, \mathbf{y}) \right]_{\mathbf{y}=\mathbf{x}}. \tag{6.4}$$

In terms of the metric, the line element is given by:

$$ds^2 = g_{\mu\nu} dx^\mu dx^\nu, \tag{6.5}$$

where a sum over repeated indices is implied. Finally, equating the last two expression gives:

$$g_{\mu\nu} = \frac{1}{2}\frac{\partial}{\partial x_\mu}\frac{\partial}{\partial x_\nu}k(\mathbf{x},\mathbf{x}) - \left[\frac{\partial}{\partial y_\mu}\frac{\partial}{\partial y_\nu}k(\mathbf{x},\mathbf{y})\right]_{\mathbf{y}=\mathbf{x}} \tag{6.6}$$

provided that the kernel function $k(\mathbf{x},\mathbf{y})$ is twice-differentiable.

We now consider the metrics induced by the family of arc-cosine kernels $k_n(\mathbf{x},\mathbf{y})$ in eq. (2.3) of degree $n \geq 1$. As a first step, we analyze the behavior of these kernels for nearby inputs $\mathbf{x} \approx \mathbf{y}$. This behavior is in turn determined by the behavior of the functions $J_n(\theta)$ in eq. (2.4) for small values of θ. For $n \geq 1$, this behavior is locally quadratic with a maximum at $\theta = 0$. In particular, by expanding the integral representation in eq. (2.1) for small values of θ, it can be shown that:

$$J_n(0) = \pi\,(2n-1)!!, \tag{6.7}$$

$$J_n(\theta) \approx J_n(0)\left(1 - \frac{n^2\theta^2}{2(2n-1)}\right), \tag{6.8}$$

where $(2n-1)!! = \frac{(2n)!}{2^n\,n!}$ is known as the double-factorial function. Together with eq. (2.3), the quadratic expansion in eq. (6.8) captures the behavior of the arc-cosine kernels $k_n(\mathbf{x},\mathbf{y})$ for nearby inputs $\mathbf{x} \approx \mathbf{y}$. It follows from eq. (6.6) that this behavior also determines the form of the metric.

The metrics for arc-cosine kernels of degree $n \geq 1$ can be derived by substituting the general form in eq. (2.3) into eq. (6.6). After some algebra (see Appendix D), this calculation gives:

$$g_{\mu\nu} = n^2(2n-3)!!\,\|\mathbf{x}\|^{2n-2}\left(\delta_{\mu\nu} + 2(n-1)\frac{x_\mu x_\nu}{\|\mathbf{x}\|^2}\right). \tag{6.9}$$

In general, this metric differs from the Euclidean metric in the prefactor $\|\mathbf{x}\|^{2n-2}$ and the projection component $x_\mu x_\nu/\|\mathbf{x}\|^2$. However, it is worth distinguishing two qualitatively different regimes of behavior.

Table 6.1: Comparison of the metric $g_{\mu\nu}$ and scalar curvature S for different kernels over inputs $\mathbf{x} \in \mathbb{R}^d$. The results for polynomial (the second row) and RBF kernels (the third row) were derived by Burges (1999). The results for arc-cosine kernels (the bottom row) are valid for kernels of order $n \geq 1$.

$k(\mathbf{x}, \mathbf{y})$	$g_{\mu\nu}$	S
$\mathbf{x} \cdot \mathbf{y}$	$\delta_{\mu\nu}$	0
$(\mathbf{x} \cdot \mathbf{y})^p$	$p \left\|\mathbf{x}\right\|^{2p-2} \left(\delta_{\mu\nu} + (p-1)\frac{x_\mu x_\nu}{\left\|\mathbf{x}\right\|^2}\right)$	$\frac{(p-1)(2-d)(d-1)}{p\left\|\mathbf{x}\right\|^{2p}}$
$e^{-\left\|\mathbf{x}-\mathbf{y}\right\|^2/\sigma^2}$	$(2/\sigma^2)\delta_{\mu\nu}$	0
$\frac{\left\|\mathbf{x}\right\|^n \left\|\mathbf{y}\right\|^n}{\pi} J_n(\theta)$	$n^2(2n-3)!! \left\|\mathbf{x}\right\|^{2n-2} \left(\delta_{\mu\nu} + 2(n-1)\frac{x_\mu x_\nu}{\left\|\mathbf{x}\right\|^2}\right)$	$\frac{3(n-1)^2\,(2-d)\,(d-1)}{n^2\,(2n-1)!!\,\left\|\mathbf{x}\right\|^{2n}}$

Case $n = 1$: the manifold is flat.

Though the metric in eq. (6.9) is generally non-Euclidean, an interesting result emerges in the special case $n=1$: it reduces to the Euclidean metric $g_{\mu\nu} = \delta_{\mu\nu}$. Thus, despite performing a highly nonlinear mapping, the $n = 1$ arc-cosine kernel induces a surface in Hilbert space that is intrinsically flat. In Section 2.1.1, we observed that the $n = 1$ arc-cosine kernel preserves the norm of inputs with $k_1(\mathbf{x}, \mathbf{x}) = \left\|\mathbf{x}\right\|^2$. In this section, we have shown that also like the purely linear kernel, it preserves the fully Euclidean metric.

In fact, it is not unusual for nonlinear kernels to preserve the Euclidean metric. Burges (1999) observed that all translationally invariant kernels of the form $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x}-\mathbf{y})$ have this property, including the popular family of RBF kernels. Note, however, that the $n=1$ arc-cosine kernel is not translationally invariant. It represents a different form of nonlinearity that nonetheless preserves the Euclidean metric.

Case $n \geq 2$: the manifold is curved.

The metrics from arc-cosine kernels of degree $n \geq 2$ have a similar form as metrics from homogeneous polynomial kernels of degree $p \geq 2$. Table 6.1 compares our results to previous results obtained for polynomial and RBF kernels (Amari and Wu, 1999; Burges, 1999). We will see later that the metric in eq. (6.9) describes

a manifold with non-zero intrinsic curvature if the inputs $\mathbf{x} \in \mathbb{R}^d$ live in three or more dimensions ($d > 2$).

Volume element

The metric $g_{\mu\nu}$ determines other interesting quantities as well. For example, the volume element dV on the manifold is given by:

$$dV = \sqrt{\det g_{\mu\nu}}\, d\mathbf{x}. \tag{6.10}$$

Assuming that the mapping from inputs to features is one-to-one, the volume element determines how a probability density transforms under this mapping.

The determinant of the metric for arc-cosine kernels is straightforward to compute. In particular, noting that the metric in eq. (6.9) is proportional to the identity matrix plus a projection matrix:

$$g = n^2(2n-3)!!\, \|\mathbf{x}\|^{2n-2} \left(\mathbf{I} + \frac{2(n-1)}{\|\mathbf{x}\|^2} \mathbf{x}\mathbf{x}^\top \right), \tag{6.11}$$

we find:

$$\det(g) = (2n-1)\, \left(n^2(2n-3)!!\, \|\mathbf{x}\|^{2n-2} \right)^d. \tag{6.12}$$

For the special case $n = 1$, this expression reduces to $\det(g) = 1$, consistent with the previous observation that in this case, the metric is Euclidean.

Curvature

The metric $g_{\mu\nu}$ also determines the intrinsic curvature of the manifold. The curvature is expressed in terms of the Christoffel elements of the second kind:

$$\Gamma^\alpha_{\beta\gamma} = \frac{1}{2} g^{\alpha\mu} \Big(\partial_\beta g_{\gamma\mu} - \partial_\mu g_{\beta\gamma} + \partial_\gamma g_{\mu\beta} \Big), \tag{6.13}$$

where $\partial_\mu = \partial / \partial x_\mu$ denotes the partial derivative and $g^{\alpha\mu}$ denotes the matrix inverse of the metric. In terms of these quantities, the Riemann curvature tensor is

given by:

$$R_{\nu\alpha\beta}{}^{\mu} = \partial_{\alpha}\Gamma^{\mu}_{\beta\nu} - \partial_{\beta}\Gamma^{\mu}_{\alpha\nu} + \Gamma^{\rho}_{\alpha\nu}\Gamma^{\mu}_{\beta\rho} - \Gamma^{\rho}_{\beta\nu}\Gamma^{\mu}_{\alpha\rho} \; . \tag{6.14}$$

The elements of $R_{\nu\alpha\beta}{}^{\mu}$ vanish for a manifold with no intrinsic curvature. The scalar curvature is given by:

$$S = g^{\nu\beta} R_{\nu\mu\beta}{}^{\mu}. \tag{6.15}$$

The scalar curvature describes the amount by which the volume of a geodesic ball on the manifold deviates from that of a ball in Euclidean space.

Substituting the metric in eq. (6.9) into eqs. (6.13–6.15), we obtain the scalar curvature for surfaces in Hilbert space induced by arc-cosine kernels:

$$S = \frac{3(n-1)^2\,(2-d)\,(d-1)}{n^2\,(2n-1)!!\,\|\mathbf{x}\|^{2n}}. \tag{6.16}$$

Note that the curvature vanishes for the kernel of degree $n=1$, as well as for all kernels in this family when the inputs $\mathbf{x} \in \mathbb{R}^d$ lie in $\mathbb{R}^1$ or $\mathbb{R}^2$. The vanishing of the curvature in these circumstances is also observed in the family of homogeneous polynomial kernels, as shown in Table 6.1.

6.2.2 Non-analytic Kernels

Next we show that the $n = 0$ arc-cosine kernel does not induce a surface in Hilbert space whose geometry can be described as a Riemannian manifold. Consider the squared distance between the feature vectors $\mathbf{\Phi}(\mathbf{x})$ and $\mathbf{\Phi}(\mathbf{x}+\mathbf{dx})$ induced by this kernel for an infinitesimal displacement $\mathbf{dx}$. To begin, we compute this distance for non-radial displacements $\mathbf{dx}_{\perp}$ that are orthogonal to $\mathbf{x}$, satisfying $\mathbf{x} \cdot \mathbf{dx}_{\perp} = 0$. Recall that the $n = 0$ arc-cosine kernel maps all inputs $\mathbf{x}$ to the unit

hypersphere in feature space, with $k_0(\mathbf{x}, \mathbf{x}) = 1$. Exploiting this property, we find:

$$\begin{aligned}
&\|\mathbf{\Phi}(\mathbf{x} + \mathbf{dx}_\perp) - \mathbf{\Phi}(\mathbf{x})\|^2 \\
&\quad = \quad k_0(\mathbf{x}+\mathbf{dx}_\perp, \mathbf{x}+\mathbf{dx}_\perp) + k_0(\mathbf{x}, \mathbf{x}) - 2k_0(\mathbf{x}, \mathbf{x}+\mathbf{dx}_\perp) \\
&\quad = \quad (2/\pi) \cos^{-1}(\|\mathbf{x}\|/\|\mathbf{x} + \mathbf{dx}_\perp\|) \\
&\quad = \quad (2/\pi) \sin^{-1}(\|\mathbf{dx}_\perp\|/\|\mathbf{x} + \mathbf{dx}_\perp\|) \\
&\quad \approx \quad (2/\pi)\|\mathbf{dx}_\perp\|/\|\mathbf{x}\|, \qquad (6.17)
\end{aligned}$$

where in the last line we have approximated the right hand side by its first-order Taylor series and kept only leading terms.

To generalize eq. (6.17) to arbitrary displacements, we note that the $n=0$ arc-cosine kernel is invariant to the magnitude of its arguments, depending only on the angle θ between them. Thus, eq. (6.17) generalizes easily to displacements $\mathbf{dx}$ that include a radial component. In particular, we can write:

$$\|\mathbf{\Phi}(\mathbf{x} + \mathbf{dx}) - \mathbf{\Phi}(\mathbf{x})\|^2 = \frac{2}{\pi\|\mathbf{x}\|} \sqrt{\mathbf{dx}^\top \left(\mathbf{I}_d - \frac{\mathbf{x}\mathbf{x}^\top}{\|\mathbf{x}\|^2} \right) \mathbf{dx}}, \qquad (6.18)$$

which merely projects out the radial component before computing the infinitesimal squared distance; here $\mathbf{I}_d$ is the $d \times d$ identity matrix.

Note that the right hand side of eq. (6.18) does not have the form of a Riemannian metric. In particular, the infinitesimal squared distance in feature space scales *linearly* not quadratically with $\|\mathbf{dx}_\perp\|$. This behavior arises from the non-analyticity of the arc-cosine function, which does not admit a Taylor series expansion around its root at unity: $\cos^{-1}(1 - \epsilon) \approx \sqrt{2\epsilon}$ for $0 < \epsilon \ll 1$. This non-analyticity not only distinguishes the $n = 0$ arc-cosine kernel from higher-order kernels in this family, but also from all polynomial and RBF kernels.

Interestingly, it turns out the arc-cosine kernel with smoothed threshold functions in eq. (3.15) removes the non-analyticity of the arc-cosine kernel of degree $n=0$. In fact, it is straightforward to compute the Riemannian metric and volume

element associated with the kernel in eq. (3.16) (see Appendix D):

$$g_{\mu\nu} = \frac{1}{\pi\sigma\sqrt{2\|\mathbf{x}\|^2+\sigma^2}}\left(\delta_{\mu\nu} - \frac{2\mathbf{x}_\mu\mathbf{x}_\nu}{2\|\mathbf{x}\|^2+\sigma^2}\right), \tag{6.19}$$

$$\det(g) = \frac{1}{\pi^d\sigma^{d-2}(2\|\mathbf{x}\|^2+\sigma^2)^{\frac{d}{2}+1}}. \tag{6.20}$$

Two observations are worth making here. First, from eq. (6.19), we see that the metric diverges as σ vanishes, reflecting the non-analyticity of the arc-cosine kernel of degree $n=0$. Second, from eq. (6.20), we see that the volume element shrinks with increasing distance from the origin in input space (i.e., with increasing $\|\mathbf{x}\|$); this property distinguishes this kernel from all the other kernels in Section 6.2.1.

6.3 Kernel Geometry and Probabilistic Modeling

6.3.1 Jacobians for Deep Learning

Consider the mapping induced by a multilayer arc-cosine kernel composed of degree $n=1$ kernels. Since the volume element is preserved under this mapping, any probability density $p(\mathbf{x})$ over the inputs $\mathbf{x}$ transforms in a trivial way. In particular, for all ℓ, we have:

$$p(\mathbf{x}) = p(\underbrace{\mathbf{\Phi}(\mathbf{\Phi}(...\mathbf{\Phi}}_{\ell \text{ times}}(\mathbf{x}))). \tag{6.21}$$

This identity suggests a role for kernel-based methods in probabilistic models of deep learning, where a multilayer arc-cosine kernel is used to mimic the computation in a deep neural net.

What can be *learned* in such a setting? Consider a further linear projection from the feature space of the multilayer kernel to a finite-dimensional output space. In particular, let $\mathbf{z} = \mathbf{L}\mathbf{\Phi}(\mathbf{\Phi}(...\mathbf{\Phi}(\mathbf{x})))$. Appealing to representer theorems (Kimeldorf and Wahba, 1971), we suppose that the linear transformation $\mathbf{L}$ acts only on

the subspace spanned by the images of training examples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ in feature space. In this case, the linear transformation can be parameterized by a finite-dimensional matrix $\mathbf{A}$, in terms of which the ith element of $\mathbf{z} = \mathbf{L}\mathbf{\Phi}(\mathbf{\Phi}(...\mathbf{\Phi}(\mathbf{x})))$ is given by:

$$z_i = \sum_{j=1}^{N} A_{ij}\, k_1^{(\ell)}(\mathbf{x}_j, \mathbf{x}). \tag{6.22}$$

Next, we consider how probability densities over the input space transform to the output space. Assuming the map from inputs $\mathbf{x}$ to outputs $\mathbf{z}$ is invertible, then by the chain rule and eq. (6.21), we can write:

$$p(\mathbf{z}) = p(\mathbf{x})\sqrt{\det(\mathbf{L}\mathbf{L}^\top)} = p(\mathbf{x})\sqrt{\det(\mathbf{A}\mathbf{K}\mathbf{A}^\top)}, \tag{6.23}$$

where $K_{ij} = k_1^{(\ell)}(\mathbf{x}_i, \mathbf{x}_j)$ denote the elements of the kernel matrix. It is straightforward to differentiate the Jacobian in this expression with respect to the matrix elements of $\mathbf{A}$. Thus this matrix can be easily adapted to learn highly nonlinear probabilistic models; we sketch one such application in the next section.

6.3.2 From Linear to Nonlinear ICA

Independent component analysis (ICA) is a popular method for linear feature extraction (Hyvärinen and Oja, 2000). From inputs $\mathbf{x}$, the goal of ICA is to compute linear projections $\mathbf{z} = \mathbf{W}\mathbf{x}$ such that the different components of $\mathbf{z}$ are not only uncorrelated, but statistically independent:

$$P(\mathbf{z}) \approx \prod_i P(z_i). \tag{6.24}$$

The goodness of the approximation in eq. (6.24) is measured by the mutual information between different components of $\mathbf{z}$. Letting $H(\cdot)$ denote the entropy, we can write the mutual information as:

$$I(\mathbf{z}) \;=\; \sum_i H(z_i) - H(\mathbf{z}). \tag{6.25}$$

Eq. (6.25) is the starting point for many formulations of ICA, each involving different ways of approximating the univariate entropies $H(z_i)$ (e.g., based on contrast functions (Hyvärinen, 1999), interval spacings (Learned-Miller and Fisher III, 2003)). The matrix $\mathbf{W}$ is optimized to minimize the surrogate function for mutual information. Note that the rightmost term in eq. (6.25) simplifies due to linearity:

$$H(\mathbf{z}) \;=\; H(\mathbf{x}) + \log\det \mathbf{W}. \tag{6.26}$$

Treating $H(\mathbf{x})$ as a constant, the optimization in linear ICA balances the two remaining terms in eq. (6.25), the first involving estimates of the univariate entropies $H(z_i)$, the second involving the logarithm of the Jacobian, $\log\det \mathbf{W}$.

A natural extension of ICA is to compute *nonlinear* features that are statistically independent (Jutten and Karhunen, 2004). However, such extensions are generally complicated by the non-trivial Jacobians that arise from nonlinear feature maps:

$$H(\mathbf{z}) = H(\mathbf{x}) + \int \mathbf{dx}\; p(\mathbf{x}) \log\left|\partial \mathbf{z}/\partial \mathbf{x}\right|. \tag{6.27}$$

Parra suggested to use symplectic (volume-conserving) transformations to parameterize nonlinear feature maps (Parra, 1996); while these transformations do not introduce a complicated Jacobian, they must still be parameterized, thus complicating other terms in the objective functions for ICA.

Multilayer arc-cosine kernels suggest an intriguing solution to this long-standing problem (Jutten et al., 2010). For nonlinear ICA, we propose the feature maps in eq. (6.22), where projections depend linearly on the matrix elements A_{ij}. The Jacobian for these feature maps takes the simple form in eq. (6.23). Substituting these two results into eq. (6.25), we obtain an objective function for nonlinear ICA that is of similar complexity to the one for linear ICA. Clearly, many issues remain to be explored in this context, such as the estimation of univariate entropies and the procedures for optimization. Here, our main goal is to provoke consideration of these and related applications of our theoretical analysis.

Several points are worth emphasizing about this "kernel trick" for nonlinear ICA. First, the simplifications hinge on the volume-preserving property of the

mapping into feature space, established in Section 6.2, which does not hold for general kernels. Second, the choice of volume-preserving kernel may affect the types of independent components that can be extracted. Multilayer arc-cosine kernels can mimic the computation in deep neural nets; by contrast, Gaussian kernels act on inputs in a similar way as soft vector quantizers. Finally, this use of kernels for nonlinear ICA is different than what is commonly known as "kernel ICA" (Bach and Jordan, 2003); the latter is a sophisticated algorithm for linear ICA where statistical dependence is measured by computing canonical correlations in a reproducing kernel Hilbert space.

6.4 Discussion

In this chapter, we have investigated the geometric properties of arc-cosine kernels. Our main theoretical results are summarized in Table 6.1. The two most intriguing results concern the surfaces in Hilbert space induced by arc-cosine kernels of degree $n=0$ and $n=1$. For the arc-cosine kernels of degree $n=0$, the surface cannot be described as a Riemannian manifold due to the non-analyticity of the kernel function. In this case, the breakdown of manifold structure implies that (differentially) nearby inputs do not always map to (differentially) nearby regions of feature space; an interesting question for future work is whether this observation can be exploited for large margin classification. For the arc-cosine kernel of degree $n=1$, we observed that the surface in Hilbert space has a Euclidean metric. This property also extends to *multilayer* kernels constructed by composition of degree $n=1$ kernels. As developed in Section 6.3, this property suggests novel applications of these kernels for deep learning and nonlinear ICA. We are studying these in ongoing work.

Other than nonlinear ICA, our theoretical and experimental results suggest many possible directions for future work. One direction is to leverage the geometric properties of the arc-cosine kernels for better classification performance. Such an idea was proposed earlier by Amari and Wu (1999) and Wu and Amari (2002), who used a conformal transformation to increase the spatial resolution around the

decision boundary induced by RBF kernels. The volume elements in eq. (6.12) and eq. (6.20) allow us to explore similar methods for the kernels analyzed in this paper.

Given the relatively simple form of the volume element, another possible direction is to explore the use of arc-cosine kernels for probabilistic modeling other than nonlinear ICA. Such an approach might exploit the connection with neural computation to provide a kernel-based alternative to inference and learning in deep belief networks (Hinton et al., 2006). Though our experimental results have not revealed a clear connection between the geometric properties of arc-cosine kernels and their performance in SVMs, it is worth emphasizing that kernels are used in many settings beyond classification, including clustering, dimensionality reduction, and manifold learning. In these other settings, the geometric properties of arc-cosine kernels may play a more prominent role.

Chapter 6, in part, is a reprint of the material as it appears in Technical Report CS2012-0972, Department of Computer Science and Engineering, University of California, San Diego. Cho, Y. and Saul, L. K. The thesis author was the primary investigator and author of this work.

Chapter 7

Conclusion

7.1 Contributions

In this thesis, we have developed a new family of kernel functions that mimic the computation in large, multilayer neural nets. On challenging data sets, we have obtained results that outperform previous SVMs and compare favorably to deep belief nets. More significantly, our experiments validate the basic intuitions behind deep learning in the altogether different context of kernel-based architectures. We conclude by reviewing our main contributions as follows.

In Chapter 2, we defined arc-cosine kernels—a new family of positive-definite kernels that mimic the computation in a special type of large neural networks with a single layer. Assuming the network weights were randomly drawn from standard Gaussian distributions, we derived analytical expressions for the kernel function (i.e., inner product of output units from two examples), which are easy to evaluate in practice. We also extended the family of arc-cosine kernels by generalizing the activation function in the network to one-sided polynomial functions. Evaluating these kernels in large margin classification by SVMs, we obtained satisfactory results on well-known classification data sets.

In Chapter 3, we extended the family of arc-cosine kernels even further by varying the activation functions in terms of the fractional growth rate of the polynomials and the amount of shifting or smoothing. These modifications enabled us to tune the amount of the variations for better data modeling, which we observed

by improved performance on the classification tasks. Besides such performance gain, our contribution in this work is that by definition of the arc-cosine kernels, we actually modeled the computation in large neural networks with interesting activation functions such as sparsity-inducing or sigmoidal ones.

In Chapter 4, we constructed new types of arc-cosine kernels using kernel composition, multiplication, and averaging. Though it was well known that these operations could build new kernels from existing ones, we discovered such operations produced non-trivial extensions only for arc-cosine kernels in contrast to other popular kernels. In particular, these extensions could be interpreted as mimicking computations in various types of multilayer neural networks. Also, using SVMs with those kernels, we achieved state-of-the-art performance in difficult deep learning benchmarks (Larochelle et al., 2007), which suggests these kernels can capture useful properties of deep architectures with much less training cost.

In Chapter 5, we explored how to construct multilayer kernel machines (MKMs), new hierarchical models based on kernel methods. For MKMs, we iterated kernel principal component analysis with feature selection, which was aimed to supplement multilayer arc-cosine kernels with unsupervised and supervised learning. By obtaining competitive performance from MKMs in the deep learning benchmarks, we verified the importance of unsupervised learning in kernel-based architectures, which has been actually a crucial factor for recent advances in deep architectures (Erhan et al., 2010). We hope that our results inspire more work on this direction of building deep feature hierarchies using models traditionally regarded as shallow (e.g., kernel methods).

Finally, in Chapter 6, we investigated the geometric properties of arc-cosine kernels, which were studied by analyzing the surfaces that these kernels induced in Hilbert space. We discovered the family of arc-cosine kernels exhibits a large variety of behaviors—described by non-analytic, flat, or curved Riemannian manifold. Even though we have not found any direct links between the kernel geometry and classification performance yet, we believe our analysis lays a theoretical foundation to use arc-cosine kernels in various settings other than typical support vector machines (e.g., nonlinear ICA in Section 6.3).

7.2 Future Work

There are many possible directions for future work. Currently, we are investigating links from kernel methods to more general deep architectures. The standard Gaussian prior for the weight of infinite neural networks enables us to derive simple analytical expressions for the arc-cosine kernels; however, this prior seems limited in a sense that the resulting network could explain only a very special type of nonlinear mapping (i.e., massive random projection), which is predefined independently from tasks we would like to solve.

We have made our efforts to overcome this issue by varying the activation functions (see Chapters 2 and 3) or combining kernels (see Chapter 4), and this can be viewed as providing more kernel options to choose from for particular problems. However, we believe it is still relevant to derive kernels that can be adjusted to each task by leveraging problem dependent information (e.g., domain specific knowledge).

In the context of arc-cosine kernels, we can attempt to define the weight prior distributions in new parametric forms and learn the parameters according to the task. For instance, we can use mixtures of Gaussian distributions instead of a single Gaussian for the weight prior. Due to the additivity of the integral in eq. (2.1), such Gaussian mixture modeling leads to additive forms of arc-cosine kernels. Interestingly, this is a special case of multiple kernel learning in Section 4.4 (Lanckriet et al., 2004; Rakotomamonjy et al., 2008; Bach, 2009; Cortes et al., 2010), where we learn the mixture coefficients with additional regularization.

Another example is to use a general covariance matrix in the Gaussian prior and learn the covariance matrix (or only its diagonal entries for high dimensional data) to use in the arc-cosine kernels, which are still derived in analytical forms (Williams, 1998). Note that this is actually related with input feature selection (Guyon and Elisseeff, 2003) or Mahalanobis distance metric learning in general (Weinberger and Saul, 2009), both of which are components of MKMs in Chapter 5.

On a related note, an intriguing question is how to extend and specialize this concept to the computer vision domain (e.g., image classification tasks

in Section 2.2). Specifically, convolutional neural networks (LeCun et al., 1989; Ranzato et al., 2007) have been found to be powerful models in many computer vision tasks (LeCun and Cortes, 1998), and we hope to incorporate the invariances modeled by the convolutional feature mapping into the arc-cosine kernels.

Borrowing more concepts from neural networks, we have additional directions to explore in arc-cosine kernels. In particular, we are interested in different techniques to regularize the multilayer kernels in Section 4.2.1. We empirically discovered preserving magnitudes of inputs was important to prevent the kernels from becoming unfeasible (e.g., kernel values decaying too quickly or exploding out of ranges), but there are other ways to regularize such sequential nonlinear processing. For instance, we can normalize the kernel outputs at every layer to be inside a predetermined range by division of entries, which is similar to the divisive normalization in image processing and convolutional neural networks (Lyu and Simoncelli, 2008; Jarrett et al., 2009).

Alternatively, we can use arc-cosine kernels with smoothed activation functions in Section 3.2.3 as individual components in kernel composition to build multilayer kernels. This is similar to traditional neural networks where sigmoidal activation functions are used at every layer. Then, by adapting the amount of smoothing at every layer properly, we can achieve similar effect of normalization. Note that how to optimize these hyperparameters is another challenging problem as well.

Besides such regularization in multilayer kernels, it seems interesting to study the effect of large numbers of layers in multilayer arc-cosine kernels because some experiments showed the performance kept increasing while adding layers, but we do not know about the limiting behavior of our models in the vertical direction. Interestingly, this makes a good comparison with deep architectures, where it is typically hard to keep adding layers as many as we want without involving huge training cost.

Finally, how to make our kernel-based models scalable to large data sets is an interesting topic to investigate. While our methods are fairly straightforward to train, they inherit certain disadvantages of all kernel-based approaches—quadratic

expansion of kernel matrices in the size of training sets. Specifically, we had difficulties in applying nonlinear support vector machines with the arc-cosine kernels on very large data sets, and we also had to sample training examples to construct a kernel matrix for kernel PCA in MKMs due to limited memory. In fact, this topic has been studied quite extensively in general; Nyström approximation of kernel matrices (Kumar et al., 2012), iterative kernel PCA (Kim et al., 2005), parallelization of support vector machines (Catanzaro et al., 2008), and approximation of kernel operations with explicit nonlinear mapping (Rahimi and Recht, 2008) are representative work in this direction, and we are considering these methods to make arc-cosine kernels more scalable to large data sets.

Appendix A

Derivation of Kernel Function

In this appendix, we show how to evaluate the multidimensional integral in eq. (2.1) 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 $w_1 w_2$-plane to contain the input $\mathbf{y}$:

$$\mathbf{x} = \mathbf{e}_1 \|\mathbf{x}\|, \tag{A.1}$$

$$\mathbf{y} = \big(\mathbf{e}_1 \cos\theta + \mathbf{e}_2 \sin\theta\big) \|y\|, \tag{A.2}$$

where $\mathbf{e}_i$ is the ith standard basis. Integrating out the orthogonal coordinates of the weight vector $\mathbf{w}$, we obtain the result in eq. (2.3) where $J_n(\theta)$ is the remaining integral:

$$\begin{aligned} J_n(\theta) = \int_{-\infty}^{\infty} dw_1 \int_{-\infty}^{\infty} dw_2 \Bigg[& e^{-\frac{w_1^2 + w_2^2}{2}} \\ & \times \Theta(w_1)\, \Theta(w_1 \cos\theta + w_2 \sin\theta)\, w_1^n\, (w_1 \cos\theta + w_2 \sin\theta)^n \Bigg]. \end{aligned} \tag{A.3}$$

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:

$$J_n(\theta) = \frac{1}{\sin\theta} \int_0^{\infty} du \int_0^{\infty} dv\; e^{-(u^2 + v^2 - 2uv\cos\theta)/(2\sin^2\theta)}\; u^n v^n. \tag{A.4}$$

The prefactor of $(\sin\theta)^{-1}$ in eq. (A.4) is due to the Jacobian. We reduce the two dimensional integral in eq. (A.4) to a one dimensional integral by adopting polar coordinates:

$$u = r\cos\phi, \tag{A.5}$$

$$v = r\sin\phi. \tag{A.6}$$

The integral over the radius coordinate r is straightforward, yielding:

$$J_n(\theta) = \frac{1}{\sin\theta}\int_0^{\frac{\pi}{2}} d\phi \int_0^{\infty} r dr\ e^{-r^2(1-\cos\theta\sin 2\phi)/\left(2\sin^2\theta\right)}\left(r^2\sin\phi\cos\phi\right)^n \tag{A.7}$$

$$= \frac{1}{\sin\theta}\int_0^{\frac{\pi}{2}} d\phi\ \left(\frac{\sin 2\phi}{2}\right)^n \int_0^{\infty} dr\ r^{2n+1} e^{-r^2(1-\cos\theta\sin 2\phi)/\left(2\sin^2\theta\right)} \tag{A.8}$$

$$= \frac{1}{\sin\theta}\int_0^{\frac{\pi}{2}} d\phi\ \left(\frac{\sin 2\phi}{2}\right)^n \frac{(2\sin^2\theta)^{n+1}}{2(1-\cos\theta\sin 2\phi)^{n+1}}\ \Gamma(n+1) \tag{A.9}$$

$$= n!\,(\sin\theta)^{2n+1}\int_0^{\frac{\pi}{2}} d\phi\ \frac{\sin^n 2\phi}{(1-\cos\theta\sin 2\phi)^{n+1}}. \tag{A.10}$$

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:

$$J_n(\theta) = n!\,(\sin\theta)^{2n+1}\int_0^{\frac{\pi}{2}} d\psi\ \frac{\cos^n\psi}{(1-\cos\theta\cos\psi)^{n+1}}. \tag{A.11}$$

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. (A.11) for the special case $n=0$. The following result can be derived by contour integration in the complex plane (Carrier et al., 2005):

$$\int_0^{\xi} \frac{d\psi}{1-\cos\theta\cos\psi} = \frac{1}{\sin\theta}\ \tan^{-1}\left(\frac{\sin\theta\ \sin\xi}{\cos\xi-\cos\theta}\right), \tag{A.12}$$

The integral in eq. (A.12) can also be verified directly by differentiating the right

hand side. Evaluating the above result at $\xi = \frac{\pi}{2}$ gives:

$$\int_0^{\pi/2} \frac{d\psi}{1 - \cos\theta\cos\psi} \;=\; \frac{\pi - \theta}{\sin\theta}. \tag{A.13}$$

Substituting eq. (A.13) into our expression for the angular part of the kernel function in eq. (A.11), 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. (A.11) 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:

$$\int_0^{\frac{\pi}{2}} d\psi \; \frac{\cos^n\psi}{(1 - \cos\theta\cos\psi)^{n+1}} \;=\; \frac{1}{n!}\,\frac{\partial^n}{\partial(\cos\theta)^n} \int_0^{\pi/2} \frac{d\psi}{1 - \cos\theta\cos\psi}. \tag{A.14}$$

Substituting eq. (A.14) into eq. (A.11), then appealing to the previous result in eq. (A.13), we recover the expression for $J_n(\theta)$ as stated in eq. (2.4):

$$\begin{aligned} J_n(\theta) &= n!\,(\sin\theta)^{2n+1}\,\frac{1}{n!}\,\frac{\partial^n}{\partial(\cos\theta)^n} \int_0^{\pi/2} \frac{d\psi}{1 - \cos\theta\cos\psi} && \text{(A.15)} \\ &= (\sin\theta)^{2n+1}\,\frac{\partial^n}{\partial(\cos\theta)^n}\left(\frac{\pi - \theta}{\sin\theta}\right) && \text{(A.16)} \\ &= (-1)^n(\sin\theta)^{2n+1}\left(\frac{1}{\sin\theta}\,\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi - \theta}{\sin\theta}\right). && \text{(A.17)} \end{aligned}$$

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. (A.11) does not have a simple analytical form. However, we can evaluate the original representation in eq. (2.1) 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 $\mathbf{x}$:

$$\mathbf{x} = \mathbf{e}_1\|\mathbf{x}\|. \tag{A.18}$$

Integrating out the orthogonal coordinates of the weight vector $\mathbf{w}$, we obtain:

$$k_n(\mathbf{x}, \mathbf{x}) = \sqrt{\frac{2}{\pi}} \|\mathbf{x}\|^{2n} \int_0^\infty dw_1 \, e^{-\frac{1}{2}w_1^2} \, w_1^{2n} \tag{A.19}$$

$$= \frac{2^n}{\sqrt{\pi}} \Gamma\left(n + \frac{1}{2}\right) \|\mathbf{x}\|^{2n}. \tag{A.20}$$

The gamma function on the right hand side of eq. (A.20) diverges as its argument approaches zero; thus $k_n(\mathbf{x}, \mathbf{x})$ diverges as $n \to -\frac{1}{2}$ for all inputs $\mathbf{x}$. This divergence shows that the integral representation of the arc-cosine kernel in eq. (2.1) 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. (2.3) and (A.11) remain generally valid. Thus, for non-integer n, the integral representation in eq. (2.1) 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. (A.11) by numerical methods. This was done for the experiments in Section 3.3.

Appendix B

Derivation of Kernel with Biased Threshold Functions

In this appendix we show how to evaluate the integral in eq. (3.2). As in Appendix A, we start by adopting coordinates in which $\mathbf{x}$ aligns with the w_1 axis and $\mathbf{y}$ lies in the w_1w_2-plane:

$$\mathbf{x} = \mathbf{e}_1\|\mathbf{x}\|, \tag{B.1}$$

$$\mathbf{y} = \big(\mathbf{e}_1\cos\theta + \mathbf{e}_2\sin\theta\big)\|\mathbf{y}\|, \tag{B.2}$$

where $\mathbf{e}_i$ is the unit vector along the ith axis and θ is defined in eq. (2.2). Next we substitute these expressions into eq. (3.2) and integrate out the remaining orthogonal coordinates of the weight vector $\mathbf{w}$. What remains is the two dimensional integral:

$$\begin{aligned} k^b(\mathbf{x},\mathbf{y}) = &\frac{1}{\pi}\int_{-\infty}^{\infty}dw_1\int_{-\infty}^{\infty}dw_2\ \Bigg[\ e^{-\frac{w_1^2+w_2^2}{2}} \\ &\times\Theta(w_1\|\mathbf{x}\|-b)\ \Theta(w_1\|\mathbf{y}\|\cos\theta + w_2\|\mathbf{y}\|\sin\theta - b)\ \Bigg]. \end{aligned} \tag{B.3}$$

We can simplify this further by adopting polar coordinates (r,ϕ) in the w_1w_2–plane of integration, where $w_1 = r\cos\phi$ and $w_2 = r\sin\phi$. With this change of variables,

we obtain the polar integral:

$$k^b(\mathbf{x},\mathbf{y}) = \frac{1}{\pi}\int_{-\pi}^{\pi} d\phi \int_0^{\infty} dr \left[re^{-\frac{r^2}{2}} \times \Theta(r\|\mathbf{x}\|\cos\phi - b)\ \Theta(r\|\mathbf{y}\|\cos(\theta-\phi) - b) \right]. \tag{B.4}$$

This integral can be evaluated in the feasible region of the plane that is defined by the arguments of the step functions. In what follows, we assume $b > 0$ since the opposite case can be derived by symmetry (as shown in Section 3.2.2). We identify the feasible region by conditioning the arguments of the step functions to be positive:

$$\cos\phi > 0, \tag{B.5}$$

$$\cos(\phi-\theta) > 0, \tag{B.6}$$

$$r > \max\left(\frac{b}{\|\mathbf{x}\|\cos\phi}, \frac{b}{\|\mathbf{y}\|\cos(\phi-\theta)}\right). \tag{B.7}$$

The first two of these inequalities limit the range of the angular integral; in particular, we require $\theta - \frac{\pi}{2} < \phi < \frac{\pi}{2}$. The third bounds the range of the radial integral from below. We can perform the radial integral analytically to obtain:

$$k^b(\mathbf{x},\mathbf{y}) = \frac{1}{\pi}\int_{\theta-\frac{\pi}{2}}^{\frac{\pi}{2}} d\phi \left[-e^{-\frac{r^2}{2}}\right]_{\max\left[\frac{b}{\|\mathbf{x}\|\cos\phi}, \frac{b}{\|\mathbf{y}\|\cos(\phi-\theta)}\right]}^{\infty} \tag{B.8}$$

$$= \frac{1}{\pi}\int_{\theta-\frac{\pi}{2}}^{\frac{\pi}{2}} d\phi \ \min\left[e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{x}\|\cos\phi}\right)^2},\ e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{y}\|\cos(\phi-\theta)}\right)^2}\right] \tag{B.9}$$

The term that is selected by the minimum operation in eq. (B.9) depends on the value of ϕ. The crossover point ϕ_c occurs where the exponents are equal, namely at $\|\mathbf{x}\|\cos\phi_c = \|\mathbf{y}\|\cos(\phi_c - \theta)$. Solving for ϕ_c yields:

$$\phi_c = \tan^{-1}\left(\frac{\|\mathbf{x}\|}{\|\mathbf{y}\|\sin\theta} - \cot\theta\right). \tag{B.10}$$

To disentangle the min-operation in the integrand, we break the range of integration into two intervals:

$$k^b(\mathbf{x},\mathbf{y}) \;=\; \frac{1}{\pi}\int_{\theta-\frac{\pi}{2}}^{\phi_c} d\phi\, e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{y}\|\cos(\phi-\theta)}\right)^2} \;+\; \frac{1}{\pi}\int_{\phi_c}^{\frac{\pi}{2}} d\phi\, e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{x}\|\cos\phi}\right)^2}. \tag{B.11}$$

Finally, we obtain a more symmetric expression by appealing to the angles ξ and ψ defined in Figure 3.4. Note that ϕ_c and ψ are complementary angles, with $\phi_c = \frac{\pi}{2} - \psi$. Writing eq. (B.11) in terms of the angles ξ and ψ yields the final form in eq. (3.4):

$$k^b(\mathbf{x},\mathbf{y}) \;=\; \frac{1}{\pi}\int_0^{\phi_c+\frac{\pi}{2}-\theta} d\phi\; e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{y}\|\sin\theta}\right)^2} + \frac{1}{\pi}\int_0^{\frac{\pi}{2}-\phi_c} d\phi\; e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{x}\|\sin\phi}\right)^2} \tag{B.12}$$

$$\;=\; \frac{1}{\pi}\int_0^{\xi} d\phi\; e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{y}\|\sin\theta}\right)^2} + \frac{1}{\pi}\int_0^{\psi} d\phi\; e^{-\frac{1}{2}\left(\frac{b}{\|\mathbf{x}\|\sin\phi}\right)^2}. \tag{B.13}$$

Appendix C

Derivation of Kernel with Smoothed Threshold Functions

A simple transformation of the integral in eq. (3.15) reduces it to essentially the same integral as eq. (2.1). We begin by appealing to the integral representation of the cumulative Gaussian function:

$$\Psi_\sigma(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} d\mu\; e^{-\frac{\mu^2}{2\sigma^2}}\;\Theta(\mathbf{w}\cdot\mathbf{x}-\mu), \tag{C.1}$$

$$\Psi_\sigma(\mathbf{w}\cdot\mathbf{y}) = \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} d\nu\; e^{-\frac{\nu^2}{2\sigma^2}}\;\Theta(\mathbf{w}\cdot\mathbf{y}-\nu). \tag{C.2}$$

After substituting these representations into eq. (3.15), we obtain an expanded integral over the weight vector $\mathbf{w}$ and the new auxiliary variables μ and ν:

$$k_\sigma(\mathbf{x},\mathbf{y}) = 2\int\mathbf{dw}\int d\mu\int d\nu\;\frac{e^{-\frac{\|\mathbf{w}\|^2}{2}-\frac{\mu^2}{2\sigma^2}-\frac{\nu^2}{2\sigma^2}}}{\sigma^2(2\pi)^{(d+2)/2}}\;\Theta(\mathbf{w}\cdot\mathbf{x}-\mu)\;\Theta(\mathbf{w}\cdot\mathbf{y}-\nu). \tag{C.3}$$

Let $\bar{\mathbf{w}}\in\mathbb{R}^{d+2}$ denote the extended weight vector obtained by appending two new elements to $\mathbf{w}\in\mathbb{R}^d$ as follows:

$$\bar{\mathbf{w}} = \left(\mathbf{w},\;\mu\sigma^{-1},\nu\sigma^{-1}\right). \tag{C.4}$$

Also let $\bar{\mathbf{x}}\in\mathbb{R}^{d+2}$ and $\bar{\mathbf{y}}\in\mathbb{R}^{d+2}$ denote the extended inputs defined by appending

two new elements to $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$ as follows:

$$\bar{\mathbf{x}} = (\mathbf{x},\ -\sigma,\ 0), \tag{C.5}$$

$$\bar{\mathbf{y}} = (\mathbf{y},\ 0,\ -\sigma). \tag{C.6}$$

The transformation is completed by writing the required integral for eq. (C.3) in terms of $\bar{\mathbf{w}}$, $\bar{\mathbf{x}}$, and $\bar{\mathbf{y}}$. This change of variables yields an integral analogous to eq. (2.1), with $\bar{\mathbf{w}}$, $\bar{\mathbf{x}}$, and $\bar{\mathbf{y}}$ playing the same roles as $\mathbf{w}$, $\mathbf{x}$, and $\mathbf{y}$. The result in eq. (3.16) follows.

Appendix D

Derivation of Riemannian Metric

In this appendix, we show how to derive the results for the Riemannian metric and curvature that appear in Section 6.2.1. We begin by deriving the individual terms that appear in the expression for the metric in eq. (6.6). Substituting the form of the arc-cosine kernels in eq. (2.3) into this expression, we obtain:

$$\partial_{x_\mu}\partial_{x_\nu}k_n(\mathbf{x},\mathbf{x}) = \frac{2}{\pi}\,\|\mathbf{x}\|^{2n-2}\,J_n(0)\Big[n\,\delta_{\mu\nu}+2n(n-1)\,\frac{x_\mu x_\nu}{\|\mathbf{x}\|^2}\Big], \tag{D.1}$$

$$\Big[\partial_{y_\mu}\partial_{y_\nu}k_n(\mathbf{x},\mathbf{y})\Big]_{\mathbf{y}=\mathbf{x}} = \frac{1}{\pi}\,\|\mathbf{x}\|^{2n-2}\Bigg(J_n(0)\Big[n\,\delta_{\mu\nu}+n(n-2)\,\frac{x_\mu x_\nu}{\|\mathbf{x}\|^2}\Big] + \Big[\frac{J_n'(\theta)}{\sin\theta}\Big]_{\theta=0}\Big[\delta_{\mu\nu}-\frac{x_\mu x_\nu}{\|\mathbf{x}\|^2}\Big]\Bigg) \tag{D.2}$$

where ∂_{x_μ} is shorthand for the partial derivative with respect to x_μ. To complete the derivation of the metric, we must evaluate the terms $J_n(0)$ and $\lim_{\theta\to 0}\big[J_n'(\theta)/\sin\theta\big]$ that appear in these expressions. As shown in Appendix A, an expression for $J_n(\theta)$ is given by the two-dimensional integral:

$$J_n(\theta) = \int_{-\infty}^{\infty}dw_1\int_{-\infty}^{\infty}dw_2\;\Big[\,e^{-\frac{w_1^2+w_2^2}{2}}\;\Theta(w_1)\;\Theta(w_1\cos\theta+w_2\sin\theta) \times w_1^n(w_1\cos\theta+w_2\sin\theta)^n\,\Big]. \tag{D.3}$$

It is straightforward to evaluate this integral at $\theta = 0$, which yields the result in eq. (2.6). Differentiating under the integral sign and evaluating at $\theta = 0$, we

obtain:

$$J_n'(0) \;=\; n\int_{-\infty}^{\infty} dw_1 \int_{-\infty}^{\infty} dw_2 \;\; e^{-\frac{w_1^2+w_2^2}{2}} \;\Theta(w_1)\, w_1^{2n-1}\, w_2 \;\;=\;\; 0, \tag{D.4}$$

where the integral vanishes due to symmetry. To evaluate the rightmost term in eq. (D.2), we avail ourselves of l'Hôpital's rule:

$$\lim_{\theta\to 0}\left[\frac{J_n'(\theta)}{\sin\theta}\right] \;=\; \lim_{\theta\to 0} J_n''(\theta) \;=\; J_n''(0). \tag{D.5}$$

Differentiating eq. (D.3) twice under the integral sign and evaluating at $\theta = 0$, we obtain:

$$\begin{aligned} J_n''(0) \;&=\; n\int_{-\infty}^{\infty} dw_1 \int_{-\infty}^{\infty} dw_2 \; e^{-\frac{w_1^2+w_2^2}{2}} \;\; \Theta(w_1) \;\; w_1^{2n-2} \;\; \left[(n-1)w_2^2 - w_1^2\right] \\ &=\; -\pi n^2 (2n-3)!!. \end{aligned} \tag{D.6}$$

Substituting these results into eq. (6.6), we obtain the expression for the metric in eq. (6.9). The remaining calculations for the curvature are tedious but straightforward. Using the Woodbury matrix identity, we can compute the matrix inverse of the metric as:

$$g^{\mu\nu} = \frac{1}{\|\mathbf{x}\|^{2n-2} n^2 (2n-3)!!}\left(\delta_{\mu\nu} - \frac{x_\mu x_\nu}{\|\mathbf{x}\|^2}\frac{2(n-1)}{2n-1}\right). \tag{D.7}$$

The partial derivatives of the metric are also easily computed as:

$$\begin{aligned} \partial_\beta g_{\gamma\mu} \;&=\; 2n^2(n-1)(2n-3)!! \;\|\mathbf{x}\|^{2n-4} \\ &\quad\times\left(x_\beta\delta_{\gamma\mu} + x_\gamma\delta_{\beta\mu} + x_\mu\delta_{\beta\gamma} + (2n-4)\frac{x_\beta x_\gamma x_\mu}{\|\mathbf{x}\|^2}\right). \end{aligned} \tag{D.8}$$

Substituting these results for the metric inverse and partial derivatives into eq. (6.13), we obtain the Christoffel elements of the second kind:

$$\begin{aligned}
\Gamma^{\alpha}_{\beta\gamma} &= g^{\alpha\mu}\Gamma_{\beta\gamma\mu} && \text{(D.9)}\\
&= \frac{1}{2}\, g^{\alpha\mu}(\partial_\beta g_{\gamma\mu} - \partial_\mu g_{\beta\gamma} + \partial_\gamma g_{\mu\beta}) && \text{(D.10)}\\
&= \frac{n-1}{\|\mathbf{x}\|^2}\left(x_\beta\delta_{\alpha\gamma} + x_\gamma\delta_{\alpha\beta} + \frac{x_\alpha\delta_{\beta\gamma}}{2n-1} - \frac{2n}{2n-1}\frac{x_\alpha x_\beta x_\gamma}{\|\mathbf{x}\|^2}\right). && \text{(D.11)}
\end{aligned}$$

Substituting these Christoffel elements into eq. (6.14), we obtain the Riemann curvature tensor:

$$\begin{aligned}
R_{\nu\alpha\beta}{}^{\mu} =& \frac{3}{\|\mathbf{x}\|^2}\frac{(n-1)^2}{2n-1}\left(\frac{x_\mu x_\alpha\delta_{\beta\nu} - x_\mu x_\beta\delta_{\nu\alpha} + x_\nu x_\beta\delta_{\mu\alpha} - x_\nu x_\alpha\delta_{\mu\beta}}{\|\mathbf{x}\|^2}\right.\\
&\left. + \delta_{\mu\beta}\delta_{\nu\alpha} - \delta_{\mu\alpha}\delta_{\beta\nu}\right).
\end{aligned} \qquad \text{(D.12)}$$

Then, combining eqs. (D.7) and (D.12), we obtain the scalar curvature S in eq. (6.16):

$$\begin{aligned}
S &= g^{\nu\beta}R_{\nu\mu\beta}{}^{\mu} && \text{(D.13)}\\
&= \frac{3(n-1)^2\,(2-d)\,(d-1)}{n^2\,(2n-1)!!\,\|\mathbf{x}\|^{2n}}. && \text{(D.14)}
\end{aligned}$$

Finally, we briefly report how to derive the Riemannian metric and volume element for the arc-cosine kernel with smoothed activation functions in Section 3.2.3. We first calculate second-order partial derivatives of $k(\mathbf{x},\mathbf{y})$ as follows:

$$\begin{aligned}
\partial_{x_\mu}\partial_{x_\nu}k_n(\mathbf{x},\mathbf{x}) &= \frac{1}{\pi}\left[\frac{2\sigma}{\sqrt{2\|\mathbf{x}\|^2+\sigma^2}(\|\mathbf{x}\|^2+\sigma^2)}\right. && \text{(D.15)}\\
&\qquad \left.\times\left(\delta_{\mu\nu} - 2x_\mu x_\nu\left[\frac{1}{2\|\mathbf{x}\|^2+\sigma^2} + \frac{1}{\|\mathbf{x}\|^2+\sigma^2}\right]\right)\right],\\
\left[\partial_{y_\mu}\partial_{y_\nu}k_n(\mathbf{x},\mathbf{y})\right]_{\mathbf{y}=\mathbf{x}} &= \frac{1}{\pi(\|\mathbf{x}\|^2+\sigma^2)\;\sigma\sqrt{2\|\mathbf{x}\|^2+\sigma^2}} && \text{(D.16)}\\
&\qquad \times\left(2x_\mu x_\nu\left[\frac{\|\mathbf{x}\|^4 - \|\mathbf{x}\|^2\sigma^2 - \sigma^4}{(2\|\mathbf{x}\|^2+\sigma^2)(\|\mathbf{x}\|^2+\sigma^2)}\right] - \|\mathbf{x}\|^2\delta_{\mu\nu}\right).
\end{aligned}$$

By plugging in those results into eq. (6.6), we obtain the Riemannian metric

for the kernel:

$$g_{\mu\nu} = \frac{1}{2}\partial_{x_\mu}\partial_{x_\nu}k_n(\mathbf{x},\mathbf{x}) - \left[\partial_{y_\mu}\partial_{y_\nu}k_n(\mathbf{x},\mathbf{y})\right]_{\mathbf{y}=\mathbf{x}} \tag{D.17}$$

$$= \frac{1}{\pi\sigma\sqrt{2\|\mathbf{x}\|^2+\sigma^2}}\left(\delta_{\mu\nu} - \frac{2x_\mu x_\nu}{2\|\mathbf{x}\|^2+\sigma^2}\right). \tag{D.18}$$

Also, using the matrix determinant lemma, we can compute the volume element as:

$$\det(g) = \frac{1}{\pi^d\sigma^d(\sqrt{2\|\mathbf{x}\|^2+\sigma^2})^d}\left(1 - \frac{2\|\mathbf{x}\|^2}{2\|\mathbf{x}\|^2+\sigma^2}\right) \tag{D.19}$$

$$= \frac{1}{\pi^d\sigma^{d-2}(2\|\mathbf{x}\|^2+\sigma^2)^{\frac{d}{2}+1}}. \tag{D.20}$$

Bibliography

Aizerman, M. A., Braverman, E. M., and Rozonoér, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837.

Amari, S. and Wu, S. (1999). Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789.

Bach, F. (2009). Exploring large feature spaces with hierarchical multiple kernel learning. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 105–112.

Bach, F. R. and Jordan, M. I. (2003). Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48.

Bellman, R. E. (1961). *Adaptive control processes - A guided tour*. Princeton University Press.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press.

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., Decoste, D., and Weston, J., editors, *Large-Scale Kernel Machines*. MIT Press.

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

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

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

Burges, C. J. C. (1999). Geometry and invariance in kernel based methods. In Schölkopf, B., Burges, C. J. C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press.

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

Catanzaro, B., Sundaram, N., and Keutzer, K. (2008). Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, pages 104–111.

Chang, C.-C. and Lin, C.-J. (2001). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Cho, Y. and Saul, L. K. (2009). Kernel methods for deep learning. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 342–350.

Cho, Y. and Saul, L. K. (2010). Large margin classification in infinite neural networks. *Neural Computation*, 22(10):2678–2697.

Cho, Y. and Saul, L. K. (2012). Analysis and extension of arc-cosine kernels for large margin classification. Technical Report CS2012-0972, Department of Computer Science and Engineering, University of California, San Diego.

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-05)*, pages 539–546.

Collobert, R. and 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)*, pages 160–167.

Cortes, C., Mohri, M., and Rostamizadeh, A. (2010). Two-Stage Learning Kernel Algorithms. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 239–246.

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

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

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

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

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Frank, A. and Asuncion, A. (2010). UCI machine learning repository.

Goldberger, J., Roweis, S., Hinton, G. E., and Salakhutdinov, R. (2005). Neighbourhood components analysis. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

Guyon, I., Gunn, S., Ben-Hur, A., and Dror, G. (2005). Result analysis of the nips 2003 feature selection challenge. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press.

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

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

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

Hyvärinen, A. (1999). Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128.

Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proceedings of International Conference on Computer Vision (ICCV'09)*. IEEE.

Johnson, W. and Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society.

Jutten, C., Babaie-Zadeh, M., and Karhunen, J. (2010). Nonlinear mixtures. In Comon, P. and Jutten, C., editors, *Handbook of Blind Source Separation, Independent Component Analysis and Applications*, pages 549–592. Academic Press.

Jutten, C. and Karhunen, J. (2004). Advances in blind source separation (BSS) and independent component analysis (ICA) for nonlinear mixtures. *International Journal of Neural Systems*, 14(5):267–292.

Kim, K. I., Franz, M. O., and Scholkopf, B. (2005). Iterative kernel principal component analysis for image modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(9):1351–1366.

Kimeldorf, G. S. and Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95.

Kumar, S., Mohri, M., and Talwalkar, A. (2012). Sampling methods for the nyström method. *Journal of Machine Learning Research*, 13:981–1006.

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

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning (ICML-95)*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and 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)*, pages 473–480.

Learned-Miller, E. G. and Fisher III, J. W. (2003). ICA using spacings estimates of entropy. *Journal of Machine Learning Research*, 4:1271–1295.

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

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

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

Lyu, S. and Simoncelli, E. P. (2008). Nonlinear image representation using divisive normalization. In *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-08)*, pages 1–8.

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)*, pages 78–85.

Parra, L. C. (1996). Symplectic nonlinear component analysis. In *Advances in Neural Information Processing Systems 8*, pages 437–443. MIT Press.

Platt, J. C. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Advances in Kernel Methods - Support Vector Learning.

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

Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. MIT Press.

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

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

Ranzato, M. A., Huang, F. J., Boureau, Y. L., and LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-07)*, pages 1–8.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. (2011). On random weights and unsupervised feature learning. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1089–1096.

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

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

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

Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley & Sons.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning (ICML-08)*, pages 1096–1103.

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

Weinberger, K. Q. and 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., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, pages 1168–1175.

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

Wu, S. and Amari, S. (2002). Conformal transformation of kernel functions: a data-dependent way to improve support vector machine classifiers. *Neural Processing Letters*, 15(1):59–67.

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