SHALLOW AND DEEP LEARNING FOR AUDIO AND NATURAL LANGUAGE PROCESSING

BY

PO-SEN HUANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Mark A. Hasegawa-Johnson, Chair
Professor Thomas S. Huang
Assistant Professor Paris Smaragdis
Assistant Professor Maxim Raginsky

Abstract

Many machine learning algorithms can be viewed as optimization problems that seek the optimum hypothesis in a hypothesis space. To model the complex dependencies in real-world artificial intelligence tasks, machine learning algorithms are required to have high expressive power (high degrees of freedom or richness of a family of functions) and a large amount of training data. Deep learning models and kernel machines are regarded as models with high expressive power through the composition of multiple layers of nonlinearities and through nonlinearly mapping data to a high-dimensional space, respectively.

While the majority of deep learning work is focused on pure classification problems given input data, there are many other challenging Artificial Intelligence (AI) problems beyond classification tasks. In real-world applications, there are cases where we have structured relationships between and among input data and output targets, which have not been fully taken into account in deep learning models. On the other hand, though kernel machines involve convex optimization and have strong theoretical grounding in tractable optimization techniques, for large-scale applications, kernel machines often suffer from significant memory requirements and computational expense. Resolving the computational limitation and thereby enhancing the expressibility of kernel machines are important for large-scale real-world applications.

Learning models based on deep learning and kernel machines for audio and natural language processing tasks are developed in this dissertation. In particular, we address the challenges for deep learning with structured relationships among data and the computational limitations of large-scale kernel machines. A general framework is proposed to consider the relationship among output predictions and enforce constraints between a mixture input and output predictions for monaural source separation tasks. To model the structured relationships among inputs, the deep structured semantic models

are introduced for an information retrieval task. Queries and documents are modeled as inputs to the deep learning models and the relevance is measured through the similarity at the output layer. A discriminative objective function is proposed to exploit the similarity and dissimilarity between queries and web documents. To address the scalability and efficiency of large-scale kernel machines, using deep architectures, ensemble models, and a scalable parallel solver are investigated to further scale-up kernel machines approximated by randomized feature maps. The proposed techniques are shown to match the expressive power of deep neural network based models in spoken language understanding and speech recognition tasks.

To my girlfriend, my sister, and my parents

Acknowledgments

I am sincerely thankful to many people for helping and supporting me during my PhD journey. First of all, I would like to thank my advisor and role model, Professor Mark Hasegawa-Johnson. Mark's broad knowledge in different disciplines and passion in research have greatly inspired me for being a good researcher. I am very grateful for his patience and kindness in providing invaluable guidance and encouragement throughout my graduate studies. I enjoy and cherish the discussions we had over these years, the research environment and freedom in our group for pursuing our research interests.

I am thankful to Professor Thomas Huang and Professor Paris Smaragdis for their collaboration, support, and constructive feedback during my studies. It has been my great honor to collaborate with them and learn from them. I am also grateful to Professor Maxim Raginsky for being on my defense committee and providing insightful comments and suggestions for this dissertation.

Besides UIUC, I have gained invaluable experiences from internships. It is my honor to work with many excellent researchers and engineers, including (in chronological order) Thyagaraju Damarla from Army Research Lab, Ajay Divakaran from SRI International Sarnoff, Gerald Friedland from ICSI Berkeley, Kshitiz Kumar, Chaojun Liu, and Yifan Gong from Microsoft, Li Deng, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck from Microsoft Research, Haim Avron, Tara Sainath, Vikas Sindhwani, and Bhuvana Ramabhadran from IBM T.J. Watson Research, and Thad Huges from Google. I specially want to thank Li Deng. Through my two internships with him, I have learned not only technique skills in deep learning but research insights from him.

Over the past few years, many friends have helped me make PhD study a memorable and exciting journey. I sincerely express gratitude to my col-

laborators (in chronological order): Jui-Ting Huang, Yoonsook Mo, Xiaodan Zhuang, Robert Mertens, Luke Gottlieb, Scott Deeann Chen, Jianchao Yang, Minje Kim, Zhaowen Wang, and Zhangyang Wang. I am also thankful for the discussions within SST and IFP groups, especially, (in alphabetical order) Sujeeth Bharadwaj, Amit Das, Preethi Jyothi, Lae-Hoon Kim, Sarah King, Zhen Li, Jianping Wang, Dan Soberal, Harsh Sharma, Haichao Zhang, Yang Zhang, and Xi Zhou. I specially want to thank Jia-Bin Huang and Yen-Huan Li for providing constructive feedback and comments. I am thankful to many other friends in UIUC, and friends in the United States and Taiwan for their support and encouragement during my studies.

Last, I thank my beloved sister, Ya-Ting Huang, and my parents, Yuan-Mei Chiu and Wen-Huang Huang, for their love and support during my studies. Without them, I would not have the courage, confidence, and ability to pursuit my dream. I also want to thank my girlfriend, Ginger Lai, for her love, encouragement, and sharing the ups and downs of my journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

A learning algorithm can be viewed as a function that maps input data into targets. Many machine learning algorithms can be seen as optimization problems that seek the optimum hypothesis in a hypothesis space by striking a balance between minimizing empirical error and minimizing the richness of a set of functions which contains the solution. For complex artificial intelligence tasks (e.g. understanding natural language, image, video, or speech), there are a large number of variables with complex dependencies. In order to model the complex dependencies and resolve these complex tasks, machine learning algorithms are required to have high expressive power (high degrees of freedom or richness of a family of functions) and a large amount of training data.

Deep learning models and kernel machines are regarded as models with high expressive power. Specifically, deep learning models can efficiently approximate a rich sets of functions through the composition of multiple layers of nonlinear functions [2]. With the same number of parameters, deeper architectures can achieve a higher complexity compared to shallow ones [3]. On the other hand, though kernel machines can be expressed as shallow architectures, kernel machines nonlinearly map data to a high-dimensional space, and thereby increase the richness and variability of the representations [4]. In theory, kernel machines can approximate any function or decision boundary arbitrarily well given enough training data [5].

1.1.1 Challenges

Deep Learning for Structured Inputs and Outputs

While the majority of deep learning work is focused on pure classification problems given input data, there are many other challenging Artificial Intelligence (AI) problems beyond classification tasks. In the classification tasks, such as image classification [6], models predict an output vector where each dimension represents a single class without assuming the relationships between different classes. In real-world applications, there are cases where we have structured relationships (i.e., correlated relationships) between and among input data and output targets. For example, in the monaural singing voice separation problems, the goal is to predict singing voice and background music accompaniment given a mixed input consisting of the singing voice and background music. The output predictions correspond to the spectral representation of targets, and thereby each output dimension is correlated with the others. In addition, when we model singing voice and background music simultaneously, there is a constraint that the sum of output predictions is the same as the input mixture. In information retrieval tasks, given a query, models retrieve the top-ranked documents based on the similarity between the query and documents. A query should have a high similarity to some documents which capture similar semantic meanings; on the other hand, the query should have a low similarity to some documents with dissimilar semantic meanings. Given the prior knowledge about the relationships between and among inputs and outputs, it is important to exploit these intrinsic structured relationships and thereby learn better models from data.

Computational Limitations in Large-scale Kernel Machines

Kernel machines involve convex optimization and have strong theoretical grounding in tractable optimization techniques. For large-scale applications, kernel machines often suffer from significant memory requirements and computational expense. In recent years, randomized Fourier feature methods have emerged as an elegant mechanism to scale-up kernel methods [7, 8]. Instead of implicitly mapping data using a kernel function, randomized Fourier feature methods explicitly map data to a higher-dimensional space and ap-

proximate the kernel function. Random Fourier feature methods result in $O(nD)$ in memory and $O(nD^2)$ in training time, where n is the number of training samples and D is the random Fourier feature dimension. The linear complexity to the number of samples is desirable provided that the number of random Fourier features is small. In practice, a large number of random Fourier features are required to obtain acceptable accuracy in predictive tasks, and this requirement induces computation challenges. Resolving the computational limitations and thereby enhancing the expressibility of kernel machines are important for large-scale real-world applications.

1.2 Outline

In this dissertation, we develop learning models based on deep learning and kernel machines for audio and natural language processing tasks. In particular, we address the challenges for deep learning with structured relationships among data and the computational limitations of large-scale kernel machines. The organization and summarization of the dissertation are follows:

Chapter 2 briefly reviews the background and theoretical comparison between neural networks and kernel machines in the statistical learning framework. We introduce the terminology and concepts used in the successive chapters, and summarize previous theoretical analysis results regarding deep learning and kernel machines.

To address deep learning beyond classification challenges, in the first part of the dissertation, Chapters 3 and 4, we consider deep learning for problems with structured relationships between and among inputs and outputs. We can further categorize them into two cases: structured relationships among outputs and structured relationships among inputs. In Chapter 3, we consider monaural source separation tasks for problems with structured relationships among outputs. We model different competing sources as outputs of deep learning models and consider the constraint that the sum of the output predictions is equal to the input. We propose a general framework to enforce the constraint by jointly optimizing a time-frequency mask with deep recurrent neural networks. Chapter 4 introduces the deep structured semantic models to consider the structured relationships among inputs. We model queries and documents as inputs of the deep learning models and measure

the relevance by the similarity between output activations. We propose a discriminative objective function to exploit the similarity and dissimilarity between inputs (queries and web documents) for the web search task.

In the second part of the dissertation, Chapters 5 and 6, we explore improving the computational efficiency of large-scale kernel machines. To resolve the computational limitations and thereby enhance the expressibility of kernel machines, we investigate using deep architectures, ensemble models, and a scalable parallel solver to further scale-up kernel machines approximated by randomized feature maps. In Chapter 5, we propose a random-feature kernel deep convex network architecture by integrating random feature approximated kernels under deep convex network architectures. Our models resolve the scalability problems when the number of samples are large and outperform previous approaches on spoken language understanding and speech recognition tasks. Chapter 6 explores the limit of shallow kernel machines and proposes two complimentary techniques: a parallel least squares solver that is specialized for kernel ridge regression with random Fourier feature approximations and an ensemble learning approach where multiple low-dimensional random Fourier models are combined together. Our results show that the shallow kernel machines can match the performance of deep neural network on speech recognition tasks.

Chapter 7 concludes and summarizes the contribution of this dissertation and discusses future work.

Chapter 2

Background

In this chapter, we briefly review the terminology and concepts used in the successive chapters, and summarize previous theoretical analysis results regarding neural networks, deep learning, and kernel machines.

2.1 Introduction to Statistical Learning Theory

In the statistical learning framework, there are tradeoffs in optimization, estimation, and approximation errors on learning problems [9–11]. In this section, we formally review some basic definitions following notations from Bottou and Bousquet [11].

Given input-output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ drawn from a probability distribution $P(x, y)$, machine learning algorithms aim to find a suitable function $f : \mathcal{X} \mapsto \mathcal{Y}$ that approximates the input-output relationship. Denote the predicted output as $\hat{y} = f(x)$. We can define a loss function $l(\hat{y}, y)$ that measures the difference between the predicted output $\hat{y}$ and the real output y.

Define f^* as the optimal function which minimizes the expected risk,

$$E(f) = \mathbb{E}[l(f(x), y)] = \int l(f(x), y) dP(x, y) \tag{2.1}$$

i.e.,

$$f^\star(x) = \underset{\hat{y}}{\operatorname{argmin}}\, \mathbb{E}[l(\hat{y}, y)|x] \tag{2.2}$$

Since the distribution $P(x, y)$ is unknown in practice, we only have access to a set of examples, $(x_i, y_i),\ i = 1, \ldots, n$, independently drawn from the

distribution, which is called the training set. We define the empirical risk,

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i), y_i) \tag{2.3}$$

Suppose the learning algorithm chooses a family $\mathcal{F}$ of candidate prediction functions and finds the function $f_n = \arg\min_{f\in\mathcal{F}} E_n(f)$ that minimizes the empirical risk. Because the optimal function, $f^\star$, might not belong to the family $\mathcal{F}$, we define

$$f^\star_{\mathcal{F}} = \arg\min_{f\in\mathcal{F}} E(f) \tag{2.4}$$

Since there are optimization errors (local vs. global optimum; computational limitations) in the minimization algorithm, any algorithm with finite computation results in a solution $\tilde{f}_n \neq f_n$ such that $E_n(\tilde{f}_n) \leq E_n(f_n) + \rho$, where $\rho \geq 0$ is the optimization tolerance. Therefore, the excess error can be defined as

$$\begin{aligned} \mathcal{E} &= \mathbb{E}[E(\tilde{f}_n) - E(f^*)] \\ &= \underbrace{\mathbb{E}[E(f^*_{\mathcal{F}}) - E(f^*)]}_{\mathcal{E}_{app}} + \underbrace{\mathbb{E}[E(f_n) - E(f^*_{\mathcal{F}})]}_{\mathcal{E}_{est}} + \underbrace{\mathbb{E}[(\tilde{f}_n) - E(f_n)]}_{\mathcal{E}_{opt}} \end{aligned} \tag{2.5}$$

The approximation error $\mathcal{E}_{app}$ measures how functions in $\mathcal{F}$ can approximate the optimal function f^*. The estimation error $\mathcal{E}_{est}$ measures how the empirical risk minimizer f_n can approximate the function $f^\star_{\mathcal{F}}$ among $\mathcal{F}$, which is determined by the capacity of $\mathcal{F}$ [9] and by the number of training examples. The optimization error $\mathcal{E}_{opt}$ reflects the effect of approximate optimization on the generalization performance. Given the constraints on the total number of training examples n_{max} and the computation time T_{max}, the optimization problem can be formalized as

$$\begin{array}{ll} \min_{\mathcal{F},\rho,n} & \mathcal{E} = \mathcal{E}_{app} + \mathcal{E}_{est} + \mathcal{E}_{opt} \\ \text{subject to} & n \leq n_{max}, T(\mathcal{F},\rho,n) \leq T_{max} \end{array} \tag{2.6}$$

Table 2.1 summarizes the tradeoffs between approximation error, estimation error, optimization error, and computation time when variables $\mathcal{F}$, n, and ρ increase [11]. In Sections 2.2 and 2.3, we introduce the basic background of neural networks and kernel machines, and review their theoretical properties in the statistical learning framework.

Table 2.1: Tradeoffs when $\mathcal{F}$, n, and ρ increase.

		$\mathcal{F}$ increases	n increases	ρ increases
$\mathcal{E}_{app}$	(approximation error)	$\searrow$		
$\mathcal{E}_{est}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{opt}$	(optimization error)	...	...	$\nearrow$
T	(computation time)	$\nearrow$	$\nearrow$	$\searrow$

2.2 Neural Networks and Deep Learning

In this section, we first introduce basic terminology of neural networks and deep learning, and review their theoretical properties under the statistical setting in Section 2.1.

Artificial neural networks (also known as neural networks) are a set of models in machine learning, inspired by biological neural networks [12]. A neural network is composed of interconnected nodes (called neurons), which can be organized into layers. Usually, there are three types of layers in the architecture: an input layer, hidden layers, and an output layer, as shown in Figure 2.1.

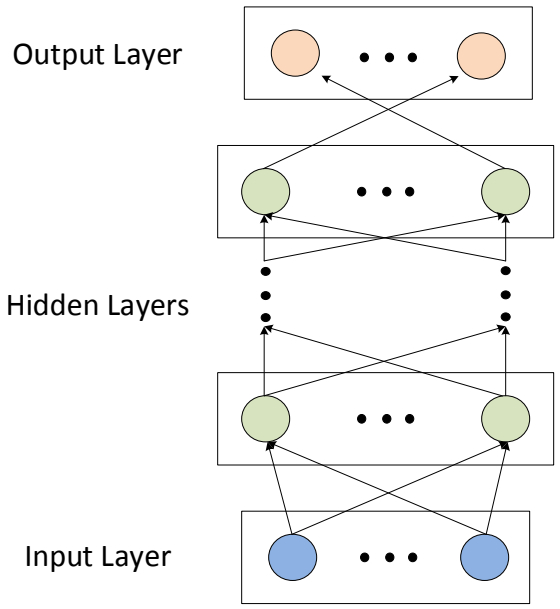


Figure 2.1: Neural network architecture.

In Figure 2.1, each circle represents a neuron, and a row of circles represents a layer. Input data is fed to the input layer of the model. A hidden layer is composed of a set of neurons taking inputs from its previous layer

and converting those inputs to the next layer. Finally, the values in the output layer represent predictions from the model. The outputs from a neural network can be viewed as a nonlinear transformation of the input data.

Formally, a neuron activation $\mathbf{a}_i$ is computed by the function $\mathbf{a}_i = f(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)$, where $\mathbf{x} \in \mathbb{R}^n$ is the input to the neuron, $\mathbf{W}_i$ and $\mathbf{b}_i$ are the parameters and bias of the neuron, respectively. The function $f(\cdot)$ is called an activation function. Commonly used functions are the sigmoid, hyperbolic tangent, or rectified linear units (ReLUs) [13]:

$$f(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.8}$$

$$f(x) = \text{ReLU}(x) = \max(0, x) \tag{2.9}$$

We can further write the multiplication in a matrix notation for m neurons together:

$$\mathbf{a} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.10}$$

where $\mathbf{W} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$, and the function $f(\cdot)$ is applied element-wise.

The connection of a neural network is usually at the layer level. A recurrent neural network is the model with cyclic connections, while a feedforward neural network is the model without cyclic connections. Multi-layer perceptron (MLP) is a feedforward neural network model and consists of one or more hidden layers between input and output layers. In terms of architecture, we categorize a neural network with more than one hidden layer as a deep architecture; on the other hand, we categorize a neural network with one hidden layer as a shallow architecture. By unfolding the recurrent neural networks in the temporal domain, recurrent neural networks can be regarded as deep architectures.

The term "deep learning" rose from the work by Hinton in 2006 [14]. One of the properties of deep learning (or deep architecture) is that the models are composed of several layers of nonlinear transformation; in other words, multiple layers are stacked on top of each other to learn complex functions. By learning with multiple nonlinear transformations, the networks are able to discover more abstract representations in the higher layers, which

makes data easier to separate from each other and hence provides a way of automated feature learning [2, 15, 16]. Recently, deep architecture has shown more effective experimental results for many applications, e.g., image recognition, speech recognition, etc., compared to shallow architectures [6, 17].

Theoretical Properties

Multi-layer perceptron (MLP), a feedforward neural network with one or more hidden layers, has been proven to be a universal approximator. Any continuous function can be approximated arbitrarily well by a feedforward neural network with single hidden layer [18]. To approximate any continuous function arbitrarily well, it suggests the need to allocate a neuron to every infinitesimal volume of the input space. Such universal approximation construction is inefficient as the dimensionality of the input space increases.

Little work has been done to analyze the optimal network architectures. Barron has analyzed the relationship between network architectures and approximation and estimation error [19]. Barron proved the approximation and estimation bounds on the risk for a single hidden layer of feedforward neural network with sigmoid nonlinearities and one-dimensional output, as follows:

$$\mathbb{E}\left[\|\hat{f}_{n,N} - f^{\star}\|_2^2\right] = \underbrace{\mathcal{O}(\frac{C_f^2}{n})}_{\text{approx. error}} + \underbrace{\mathcal{O}\left(\frac{nd}{N}\log N\right)}_{\text{est. error}} \tag{2.11}$$

where n is the number of hidden units, d is the dimensionality of the input space, N is the number of training samples, $\hat{f}_{n,N}$ is the minimum complexity estimator given N training samples with n nodes, and C_f is the first absolute moment of the Fourier magnitude distribution of the target function $f^{\star}$. The terms $\mathcal{O}(\frac{C_f^2}{n})$ and $\mathcal{O}\left(\frac{nd}{N}\log N\right)$ refer to the approximation error and estimation error, respectively. The bound suggests that when the number of hidden units increases, the approximation error decreases while the estimation error increases. It also suggests the need to use more training samples when the number of hidden units increases.

Regarding the optimal number of training samples, Baum and Haussler [20] provided bounds on the minimal number of training samples for neural networks. Assuming $0 \leq \epsilon \leq 1/8$, if there are $N \geq O(\frac{W}{\epsilon}\log\frac{n}{\epsilon})$ random

examples, a feedforward network of linear threshold functions with n nodes and W weights can correctly classify at least a fraction $1-\epsilon/2$ of the samples and correctly classify a fraction $1-\epsilon$ of future test examples drawn from the same distribution.

What is the benefit of deep architectures given the universal approximation property of a single hidden layer MLP? Recently, some theoretical comparisons between shallow and deep architectures have been studied. In logic networks, Hastad has shown that there exist Boolean functions, whose realization requires polynomial number of logic gates using deep architectures, while an exponential number of gates is required for a smaller number of layers [21]. In sum-product networks, Delalleau and Bengio have shown using two families of functions, the number of units required in the deep sum-product networks has a linear growth, compared to an exponential growth in the shallow networks, to represent the same functions [22]. Anselmi et al. proved that the image representations, which are translation and scaling invariant, learned from hierarchical architectures in an unsupervised way can reduce the sample complexity of learning [23]. Bianchini and Scarselli analyzed the complexity between shallow and deep architectures in conventional neural networks [3]. They found that with the same number of hidden units, for networks with arctangent and polynomial activation functions, deep architectures can realize maps with a higher complexity compared to shallow ones. Since the Vapnik-Chervonenkis (VC) dimension of neural networks depends on the number of network parameters (range from $O(p \log p)$ to $O(p^4)$ for different activation functions, where p is the number of network parameters [3, 24]), their finding suggests that deep architectures increase the complexity of the classifiers without significantly affecting the generalization performance. These results suggest deep architectures realize the same function more efficiently compared to shallow architectures.

2.3 Kernel Machines

In this section, we briefly introduce reproducing kernel Hilbert space, describe the kernel ridge regression, and review some theoretical properties regarding kernel machines. Readers can refer to books such as [25–27] for more details on the topic.

A reproducing kernel Hilbert space is defined by any symmetric and positive semidefinite kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ [25,26]. Given a distribution $\mathbb{P}$ on $\mathcal{X}$, the Hilbert space is strictly contained within $L^2(\mathbb{P})$. For each $x \in \mathcal{X}$, the Hilbert space $\mathcal{H}$ contains the function $z \mapsto k(z, x)$. The Hilbert space is defined by the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ such that $k(\cdot, x)$ is as the representer of evaluation, i.e.,

$$\langle f, k(x, \cdot) \rangle_{\mathcal{H}} = f(x) \text{ for } f \in \mathcal{H} \tag{2.12}$$

Under suitable conditions, Mercer's theorem guarantees that a kernel can be represented as an eigen-expansion form

$$k(x, x') = \sum_{j=1}^{\infty} \mu_j \phi_j(x) \phi_j(x') \tag{2.13}$$

where $\mu_1 \geq \mu_2 \geq \cdots \geq 0$ and $\{\phi_j\}_{j=1}^{\infty}$ are an infinite sequence of eigenvalues and eigenfunctions, respectively.

Kernel functions can be applied to many algorithms including support vector machines (SVM), principal components analysis (PCA), ridge regression, spectral clustering, etc. Here we give a basic introduction to the kernel ridge regression, which is related to Chapters 5 and 6. Kernel ridge regression integrates kernel functions with the ridge regression (linear least squares with an L2 norm regularization). Given a dataset $\{(x_i, y_i)\}_{i=1}^{N}$ with N i.i.d. samples drawn from an unknown distribution $\mathbb{P}$ over $\mathcal{X} \times \mathbb{R}$, the goal is to estimate the function that minimizes the mean-square error objective, $\mathbb{E}_{(X,Y)}[(f(X) - Y)^2]$. The optimal function for the mean-squared error is the conditional mean $f^\star(x) = \mathbb{E}[Y|X = x]$. To estimate the unknown function $f^\star$, we consider an estimator based on minimizing the least-squares loss over the dataset and a squared Hilbert norm penalty,

$$\hat{f} = \operatorname*{argmin}_{f \in \mathcal{H}} \left\{ \frac{1}{N} \sum_{i=1}^{N} (f(x_i) - y_i)^2 + \lambda \|f\|_{\mathcal{H}}^2 \right\} \tag{2.14}$$

where $\lambda > 0$ is a regularization parameter. The estimator (2.14) is known as the kernel ridge regression estimate, when $\mathcal{H}$ is a reproducing kernel Hilbert space.

Theoretical Properties

For the kernel ridge regression, under some proper assumptions [28], the mean-squared error of the estimated $\hat{f}$ in Eq. (2.14) is upper bounded as

$$\mathbb{E}\left[\|\hat{f} - f^\star\|_2^2\right] = \mathcal{O}\Big(\underbrace{\lambda \|f^\star\|_{\mathcal{H}}^2}_{\text{squared bias}} + \underbrace{\frac{\sigma^2 \gamma(\lambda)}{N}}_{\text{variance}} \Big) \tag{2.15}$$

where σ is a constant such that $\mathbb{E}\left[(Y - f^\star(x))^2 | x\right] \leq \sigma^2$ for $f^\star \in \mathcal{H}$ and $x \in \mathcal{X}$, and $\gamma(\lambda)$ is known as the "effective dimensionality" of the kernel k with respect to $L^2(\mathbb{P})$ [29]. The terms $\lambda \|f^\star\|_{\mathcal{H}}^2$ and $\sigma^2\gamma(\lambda)/N$ can be interpreted as the squared bias and variance, respectively. For the Gaussian kernel $k(x, x') = \exp(-\|x - x'\|_2^2)$, Zhang et al. [28] further showed that the term $\mathcal{O}\left(\sigma^2\gamma(\lambda)/N\right)$ can be bounded as $\mathcal{O}\left(\sigma^2\sqrt{\log N}/N\right)$, which indicates that the variance decreases as the number of samples increases.

In addition to integrating kernel methods with the ridge regression, related to Section 2.2, kernel functions can be also used in the neural network architectures. A radial-basis-function (RBF) network [30] uses radial basis functions as activation functions. The output of the RBF network is a linear combination of radial basis functions of the inputs and parameters. Park and Sandberg showed that RBF networks are capable of universal approximation [31]. Let kernel $k : \mathbb{R}^r \mapsto \mathbb{R}$ be an integrable bounded function such that k is continuous almost everywhere and $\int_{\mathbb{R}^r} k(x)dx \neq 0$, RBF networks can approximate arbitrarily well any function in $L^p(\mathbb{R}^r)$ for every $p \in [1, \infty)$.

Chapter 3

Deep Learning for Structured Outputs: Joint Optimization of Masks and Deep Recurrent Neural Networks

3.1 Introduction

Source separation is a problem in which several signals have been mixed together and the objective is to recover the original signals from the combined signals. Source separation is important for several real-world applications. For example, the accuracy of chord recognition and pitch estimation can be improved by separating the singing voice from the music accompaniment [32]. The accuracy of automatic speech recognition (ASR) can be improved by separating speech signals from noise [33]. Monaural source separation, i.e., source separation from monaural recordings, is particularly challenging because, without prior knowledge, there is an infinite number of solutions. In this chapter, we focus on source separation from monaural recordings with applications to speech separation, singing voice separation, and speech denoising tasks.

Several approaches have been proposed to address the monaural source separation problem. We categorize them into *domain-specific* and *domain-agnostic* approaches. For domain-specific approaches, models are designed according to the prior knowledge and assumptions of the tasks. For example, in singing voice separation tasks, several approaches have been proposed to exploit the assumption of the low rank and sparsity of the music and speech signals, respectively [32, 34–36]. In speech denoising tasks, spectral subtraction [37] subtracts a short-term noise spectrum estimate to generate the spectrum of a clean speech. By assuming the underlying properties of speech and noise, statistical model-based methods infer speech spectral coefficients given noisy observations [38]. However, in real-world scenarios, these strong assumptions may not always hold. For example, in the singing voice separation task, the drum sounds may lie in sparse subspaces instead of be-

ing low rank. In speech denoising tasks, the models often fail to predict the acoustic environments due to the non-stationary nature of noise.

For domain-agnostic approaches, models are learned from data directly without having any prior assumption in the task domain. Non-negative matrix factorization (NMF) [39] and probabilistic latent semantic indexing (PLSI) [40, 41] learn the non-negative reconstruction bases and weights of different sources and use them to factorize time-frequency spectral representations. NMF and PLSI can be viewed as a linear transformation of the given mixture features (e.g. magnitude spectra) during the prediction time. However, based on the minimum mean squared error (MMSE) estimate criterion, the optimal estimator $\mathbb{E}[\mathbf{Y}|\mathbf{X}]$ is a linear model in $\mathbf{X}$ only if $\mathbf{X}$ and $\mathbf{Y}$ are jointly Gaussian, where $\mathbf{X}$ and $\mathbf{Y}$ are the mixture and separated signals, respectively. In real-world scenarios, since signals might not always follow Gaussian distributions, linear models are not expressive enough to model the complicated relationship between separated and mixture signals. We consider the mapping relationship between the mixture signals and separated sources as a nonlinear transformation, and hence nonlinear models such as deep neural networks (DNNs) are desirable.

In this chapter, we propose a general monaural source separation framework to jointly model all sources within a mixture as targets to a deep recurrent neural network (DRNN). We propose to utilize the constraints between the original mixture and the output predictions through time-frequency mask functions and jointly optimize the time-frequency functions along with the deep recurrent neural network. Given a mixture signal, the proposed approach directly reconstructs the predictions of target sources in an end-to-end fashion. In addition, given that there are predicted results of competing sources in the output layer, we further propose a discriminative training criterion for enhancing the source to interference ratio. We extend our previous work in [42] and [43] and propose a general framework for monaural source separation tasks with applications to speech separation, singing voice separation, and speech denoising. We further extend our speech separation experiments in [42] to a larger speech corpus, the TSP dataset [44], with different model architectures and different speaker genders, and we extend our proposed framework to speech denoising tasks under various matched and mismatched conditions.

The organization of this chapter is as follows: Section 3.2 reviews and

compares recent monaural source separation work based on deep learning models. Section 3.3 introduces the proposed methods, including the deep recurrent neural networks, joint optimization of deep learning models and soft time-frequency masking functions, and the training objectives. Section 3.4 presents the experimental setting and results using the TSP [44], MIR-1K [45], and TIMIT [46] datasets for speech separation, singing voice separation, and speech denoising tasks, respectively. We summarize this chapter in Section 3.5.

3.2 Related Work

Recently, deep learning based methods have started to attract much attention in the source separation research community by modeling the nonlinear mapping relationship between mixture and separated signals. Prior work on deep learning based source separation can be categorized into three categories, depending on the interaction between input mixture and output targets.

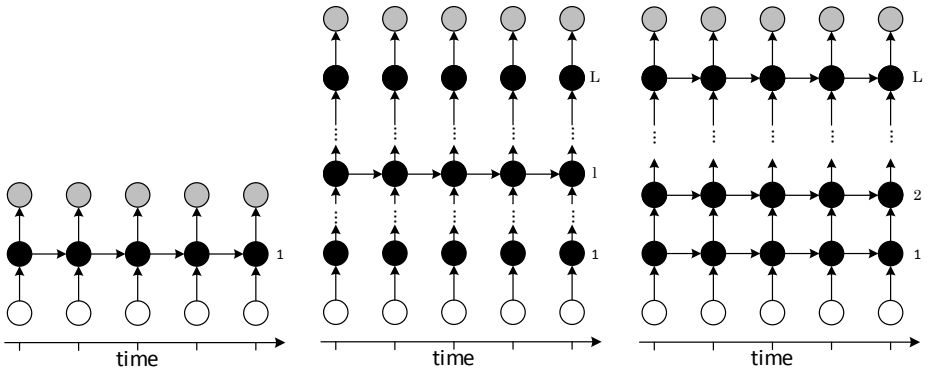
Denoising-based approaches: These methods utilize deep learning based models to learn the mapping from the mixture signals to one of the sources among the mixture signals. In the speech recognition task, given noisy features, Maas et al. [33] proposed to apply a DRNN to predict clean speech features. In the speech enhancement task, Xu et al. [47] and Liu et al. [1] proposed to use a DNN for predicting clean speech signals given noisy speech signals. The denoising methods do not consider the relationships between target and other sources in the mixture, which is suboptimal in the source separation framework where all the sources are important. In contrast, our proposed model considers *all* sources in the mixture and utilizes the relationship among the sources to formulate time-frequency masks.

Time-frequency mask based approaches: A time-frequency mask [48] considers the relationships among the sources in a mixture signal, enforces the constraints between an input mixture and the output predictions, and hence results in smooth prediction results. Weninger et al. [49] trained two long short-term memory (LSTM) RNNs for predicting speech and noise, respectively. A final prediction is made by applying a time-frequency mask based on the speech and noise predictions. Instead of training a model for

each source and applying the time-frequency mask separately, our proposed model jointly optimizes time-frequency masks with a network which models all the sources directly.

Another type of approach is to apply deep learning models to predict a time-frequency mask for one of the sources. After the time-frequency mask is learned, the estimated source is obtained by multiplying the learned time-frequency mask with an input mixture. Nie et al. [50] utilized deep stacking networks with time series inputs and a re-threshold method to predict an ideal binary mask. Narayanan and Wang [51] and Wang and Wang [52] proposed a two-stage framework (DNNs with a one-layer perceptron and DNNs with an SVM) for predicting a time-frequency mask. Wang et al. [53] recently proposed to train deep neural networks for different targets, including ideal ratio mask, FFT-mask, and Gammatone frequency power spectrum for speech separation tasks. Our proposed approach learns time-frequency masks for all the sources internally with the DRNNs and directly optimizes separated results with respect to ground truth signals in an end-to-end fashion.

Multiple-target based approaches: These methods model all output sources in a mixture as deep learning model training targets. Tu et al. [54] proposed modeling clean speech and noise as the output targets for a robust ASR task. However, the authors do not consider the constraint that the sum of all the sources is the original mixture. Grais et al. [55] proposed using a deep neural network to predict two scores corresponding to the probabilities of two different sources respectively given a frame of normalized magnitude spectrum. Our proposed method also models all sources as training targets. We further enforce the constraints between an input mixture and the output predictions through time-frequency masks which are learned along with DRNNs.



(a) 1-layer RNN (b) L-layer DRNN-l (c) L-layer stacked RNN

Figure 3.1: Deep Recurrent Neural Network (DRNN) architectures: Arrows represent connection matrices. Black, white, and gray circles represent input frames, hidden states, and output frames, respectively. The architecture in (a) is a standard recurrent neural network, (b) is an L hidden layer DRNN with recurrent connection at the l-th layer (denoted by DRNN-l), and (c) is an L hidden layer DRNN with recurrent connections at all levels (denoted by stacked RNN).

3.3 Joint Optimization of Masks and Deep Recurrent Neural Networks

3.3.1 Deep Recurrent Neural Networks

Given that audio signals are time series in nature, we propose to model the temporal information using deep recurrent neural networks for monaural source separation tasks. To capture the contextual information among audio signals, one way is to concatenate neighboring audio features, e.g., magnitude spectra, together as input features to a deep neural network. However, the number of neural network parameters increases proportionally to the input dimension and the number of neighbors in time. Hence, the size of the concatenating window is limited. Another approach is to utilize recurrent neural networks (RNNs) for modeling the temporal information. An RNN can be considered as a DNN with indefinitely many layers, which introduce the memory from previous time steps, as shown in Figure 3.1 (a). The potential weakness for RNNs is that RNNs lack hierarchical processing of the input at the current time step. To further provide the hierarchical information through multiple time scales, deep recurrent neural networks (DRNNs)

are explored [56, 57]. We formulate DRNNs in two schemes as shown in Figure 3.1 (b) and Figure 3.1 (c). The Figure 3.1 (b) is an L hidden layer DRNN with temporal connection at the l-th layer. The Figure 3.1 (c) is an L hidden layer DRNN with full temporal connections (called stacked RNN (sRNN) in [57]). Formally, we define the two DRNN schemes as follows. Suppose there is an L hidden layer DRNN with the recurrent connection at the l-th layer, the l-th hidden activation at time t, $\mathbf{h}_t^l$, is defined as:

$$\begin{aligned}\mathbf{h}_t^l &= f_{\mathrm{h}}(\mathbf{x}_t, \mathbf{h}_{t-1}^l) \\ &= \phi_l\left(\mathbf{U}^l\mathbf{h}_{t-1}^l + \mathbf{W}^l\phi_{l-1}\left(\mathbf{W}^{l-1}\left(\dots\phi_1\left(\mathbf{W}^1\mathbf{x}_t\right)\right)\right)\right)\end{aligned} \tag{3.1}$$

and the output $\mathbf{y}_t$ is defined as:

$$\begin{aligned}\mathbf{y}_t &= f_{\mathrm{o}}(\mathbf{h}_t^l) \\ &= \mathbf{W}^L\phi_{L-1}\left(\mathbf{W}^{L-1}\left(\dots\phi_l\left(\mathbf{W}^l\mathbf{h}_t^l\right)\right)\right)\end{aligned} \tag{3.2}$$

where f_{h} and f_{o} are a state transition function and an output function, respectively, $\mathbf{x}_t$ is the input to the network at time t, $\phi_l(\cdot)$ is an element-wise nonlinear function at the l-th layer, $\mathbf{W}^l$ is the weight matrix for the l-th layer, and $\mathbf{U}^l$ is the weight matrix for the recurrent connection at the l-th layer. The recurrent weight matrix $\mathbf{U}^k$ is a zero matrix for the rest of the layers where $k \neq l$. The output layer is a linear layer.

The stacked RNNs, as shown in Figure 3.1 (c), have multiple levels of transition functions, defined as:

$$\begin{aligned}\mathbf{h}_t^l &= f_{\mathrm{h}}(\mathbf{h}_t^{l-1}, \mathbf{h}_{t-1}^l) \\ &= \phi_l(\mathbf{U}^l\mathbf{h}_{t-1}^l + \mathbf{W}^l\mathbf{h}_t^{l-1})\end{aligned} \tag{3.3}$$

where $\mathbf{h}_t^l$ is the hidden state of the l-th layer at time t, $\phi_l(\cdot)$ is an element-wise nonlinear function at the l-th layer, $\mathbf{W}^l$ is the weight matrix for the l-th layer, and $\mathbf{U}^l$ is the weight matrix for the recurrent connection at the l-th layer. When the layer $l = 1$, the hidden activation $\mathbf{h}_t^1$ is computed using $\mathbf{h}_t^0 = \mathbf{x}_t$. For the nonlinear function $\phi_l(\cdot)$, similar to [58], we empirically found that using the rectified linear unit $\phi_l(\mathbf{x}) = \max(\mathbf{0}, \mathbf{x})$ performs better compared to using a sigmoid or tanh function in our experiments. Note that a DNN can be regarded as a DRNN with the temporal weight matrix $\mathbf{U}^l$ as

a zero matrix.

For the computation complexity, given the same input features, during the forward-propagation stage, a DRNN with L hidden layers, m hidden units, and a temporal connection at the l-th layer requires an extra $\Theta(m^2)$ IEEE floating point storage buffer to store the temporal weight matrix $\mathbf{U}^l$, and extra $\Theta(m^2)$ multiply-add operations to compute the hidden activations in Eq. (3.3) at the l-th layer, compared to a DNN with L hidden layers and m hidden units. During the back-propagation stage, DRNN uses back-propagation through time (BPTT) [59, 60] to update network parameters. Given an input sequence with T time steps in length, the DRNN with an l-th layer temporal connection requires an extra $\Theta(Tm)$ space to keep hidden activations in memory and requires $\Theta(Tm^2)$ operations ($\Theta(m^2)$ operations per time step) for updating parameters, compared to a DNN [61]. Indeed, the only pragmatically significant computational cost of a DRNN with respect to a DNN is that the recurrent layer limits the granularity with which back-propagation can be parallelized. As gradient updates based on sequential steps cannot be computed in parallel, for improving the efficiency of DRNN training, utterances are chopped into sequences of at most 100 time steps.

3.3.2 Model Architecture

We consider the setting where there are two sources additively mixed together, though our proposed framework can be generalized to more than two sources. At time t, the training input $\mathbf{x}_t$ of the network is the concatenation of features, e.g., logmel features or magnitude spectra, from a mixture within a window. The output targets $\mathbf{y}_{\mathbf{1}_t} \in \mathbb{R}^F$ and $\mathbf{y}_{\mathbf{2}_t} \in \mathbb{R}^F$ and the output predictions $\hat{\mathbf{y}}_{\mathbf{1}_t} \in \mathbb{R}^F$ and $\hat{\mathbf{y}}_{\mathbf{2}_t} \in \mathbb{R}^F$ of the deep learning models are the magnitude spectra of different sources, where F is the magnitude spectral dimension.

Since our goal is to separate different sources from a mixture, instead of learning one of the sources as the target, we propose to simultaneously model all the sources. Figure 3.2 shows an example of the architecture, which can be viewed as the t-th column in Figure 3.1.

Moreover, we find it useful to further smooth the source separation results with a time-frequency masking technique, for example, binary time-frequency masking or soft time-frequency masking [32, 42, 48, 62]. The time-frequency

masking function enforces the constraint that the sum of the prediction results is equal to the original mixture. Given the input features $\mathbf{x}_t$ from the mixture, we obtain the output predictions $\hat{\mathbf{y}}_{\mathbf{1}_t}$ and $\hat{\mathbf{y}}_{\mathbf{2}_t}$ through the network. The soft time-frequency mask $\mathbf{m}_t \in \mathbb{R}^F$ is defined as follows:

$$\mathbf{m}_t = \frac{|\hat{\mathbf{y}}_{\mathbf{1}_t}|}{|\hat{\mathbf{y}}_{\mathbf{1}_t}| + |\hat{\mathbf{y}}_{\mathbf{2}_t}|} \tag{3.4}$$

where the addition and division operators are element-wise operations.

Similar to [49], a standard approach is to apply the time-frequency masks $\mathbf{m}_t$ and $\mathbf{1} - \mathbf{m}_t$ to the magnitude spectra $\mathbf{z}_t \in \mathbb{R}^F$ of the mixture signals, and obtain the estimated separation spectra $\hat{\mathbf{s}}_{\mathbf{1}_t} \in \mathbb{R}^F$ and $\hat{\mathbf{s}}_{\mathbf{2}_t} \in \mathbb{R}^F$, which correspond to sources 1 and 2, as follows:

$$\begin{aligned} \hat{\mathbf{s}}_{\mathbf{1}_t} &= \mathbf{m}_t \odot \mathbf{z}_t \\ \hat{\mathbf{s}}_{\mathbf{2}_t} &= (\mathbf{1} - \mathbf{m}_t) \odot \mathbf{z}_t \end{aligned} \tag{3.5}$$

where the subtraction and $\odot$ (Hadamard product) operators are element-wise operations.

Given the benefit of smoothing separation and enforcing the constraints between an input mixture and the output predictions using time-frequency masks, we propose to incorporate the time-frequency masking functions as a layer in the neural network. Instead of training the neural network and applying the time-frequency masks to the predictions separately, we propose to jointly train the deep learning model with the time-frequency masking functions. We add an extra layer to the original output of the neural network as follows:

$$\begin{aligned} \tilde{\mathbf{y}}_{\mathbf{1}_t} &= \frac{|\hat{\mathbf{y}}_{\mathbf{1}_t}|}{|\hat{\mathbf{y}}_{\mathbf{1}_t}| + |\hat{\mathbf{y}}_{\mathbf{2}_t}|} \odot \mathbf{z}_t \\ \tilde{\mathbf{y}}_{\mathbf{2}_t} &= \frac{|\hat{\mathbf{y}}_{\mathbf{2}_t}|}{|\hat{\mathbf{y}}_{\mathbf{1}_t}| + |\hat{\mathbf{y}}_{\mathbf{2}_t}|} \odot \mathbf{z}_t \end{aligned} \tag{3.6}$$

where the addition, division, and $\odot$ (Hadamard product) operators are element-wise operations. The architecture is shown in Figure 3.2. In this way, we can integrate the constraints into the network and optimize the network with the masking functions jointly. Note that although this extra layer is a deterministic layer, the network weights are optimized for the error metric between $\tilde{\mathbf{y}}_{\mathbf{1}_t}$, $\tilde{\mathbf{y}}_{\mathbf{2}_t}$ and $\mathbf{y}_{\mathbf{1}_t}$, $\mathbf{y}_{\mathbf{2}_t}$, using the back-propagation algorithm. The time domain

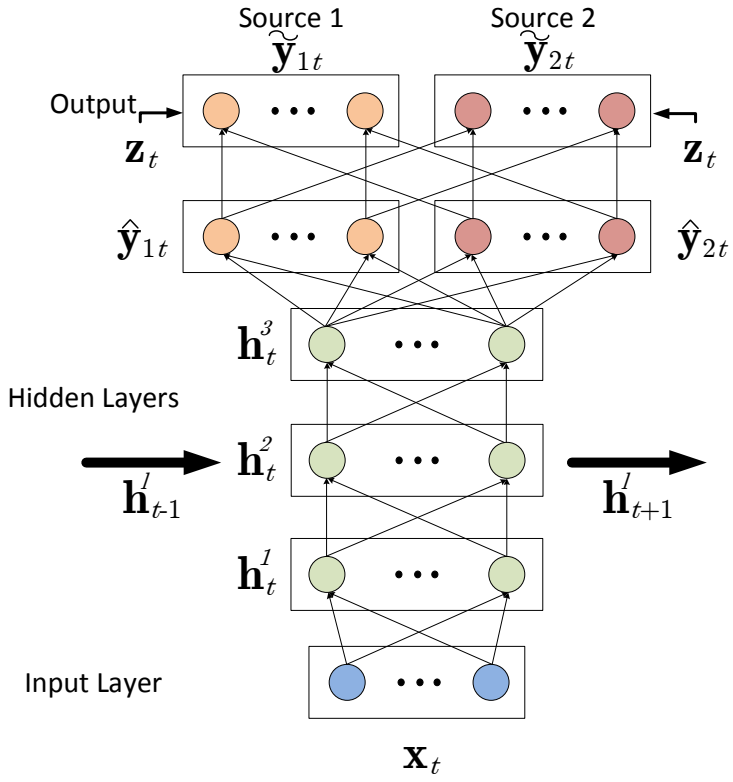


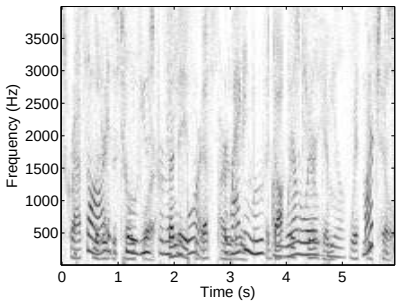
Figure 3.2: Proposed neural network architecture, which can be viewed as the t-th column in Figure 3.1. We propose to jointly optimize time-frequency masking functions as a layer with a deep recurrent neural network.

signals are reconstructed based on the inverse short-time Fourier transform (ISTFT) of the estimated magnitude spectra along with the original mixture phase spectra.

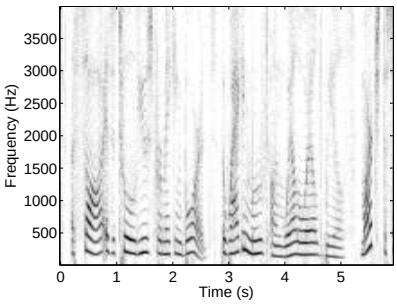
3.3.3 Training Objectives

Given the output predictions $\tilde{\mathbf{y}}_{\mathbf{1}_t}$ and $\tilde{\mathbf{y}}_{\mathbf{2}_t}$ (or $\hat{\mathbf{y}}_{\mathbf{1}_t}$ and $\hat{\mathbf{y}}_{\mathbf{2}_t}$) of the original sources $\mathbf{y}_{\mathbf{1}_t}$ and $\mathbf{y}_{\mathbf{2}_t}$, $t = 1, \ldots, T$, where T is the length of an input sequence, we optimize the neural network parameters by minimizing the squared error:

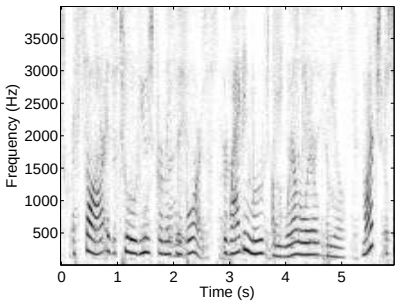
$$J_{\text{MSE}} = \frac{1}{2} \sum_{t=1}^{T} \left(\|\tilde{\mathbf{y}}_{\mathbf{1}_t} - \mathbf{y}_{\mathbf{1}_t}\|_2^2 + \|\tilde{\mathbf{y}}_{\mathbf{2}_t} - \mathbf{y}_{\mathbf{2}_t}\|_2^2 \right) \tag{3.7}$$



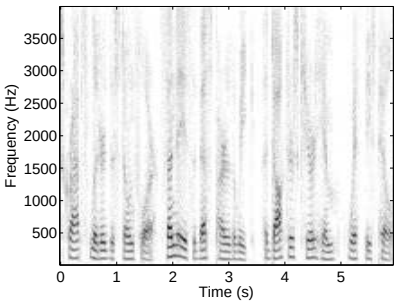
(a) Mixture



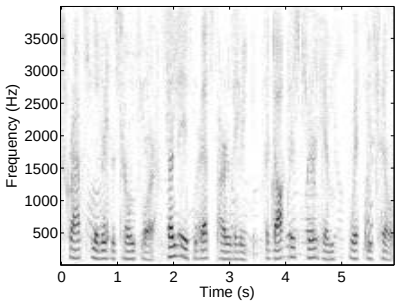
(b) Original female voice



(c) Recovered female voice



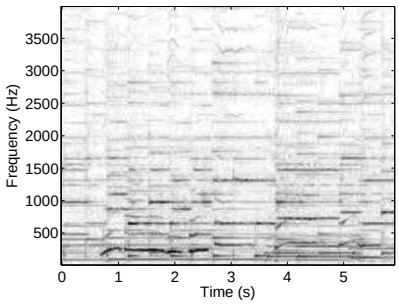
(d) Original male voice



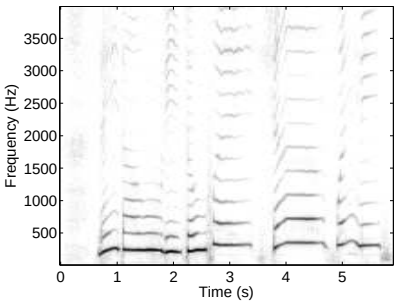
(e) Recovered male voice

Figure 3.3: A speech separation example using the TSP dataset. (a) The mixture (female (FA) and male (MC) speech) magnitude spectrogram for a test clip in TSP; (b) the ground truth spectrogram of the female speech; (c) the separated female speech spectrogram from our proposed model (DRNN-1 + discrim); (d) the ground truth spectrogram of the male speech; (e) the separated male speech spectrogram from our proposed model (DRNN-1 + discrim).

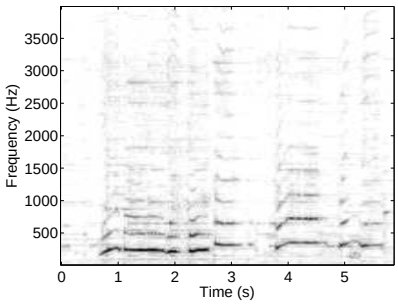
In Eq. (3.7), we measure the difference between the predicted and the actual targets. When targets have similar spectra, it is possible for the DNN to minimize Eq. (3.7) by being too conservative: when a feature could be attributed to either source 1 or source 2, the neural network attributes it to both. The conservative strategy is effective in training, but leads to reduced signal-to-interference ratio (SIR) in testing, as the network allows ambiguous



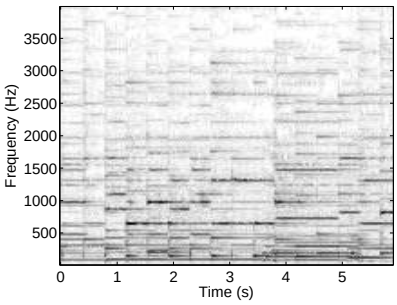
(a) Mixture



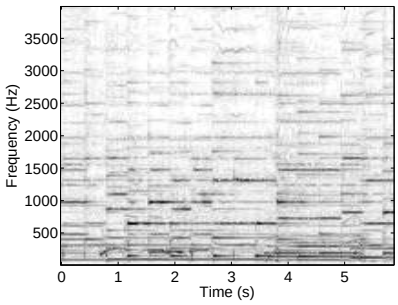
(b) Original singing



(c) Recovered singing



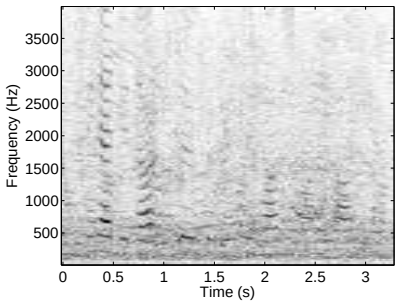
(d) Original music



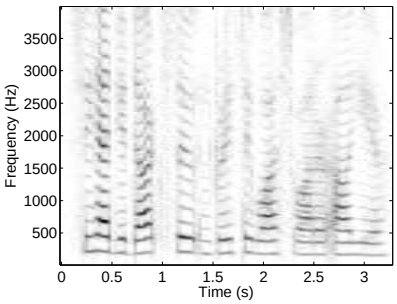
(e) Recovered music

Figure 3.4: A singing voice separation example using the MIR-1K dataset. (a) The mixture (singing voice and music accompaniment) magnitude spectrogram for the clip Yifen_2_07 in MIR-1K; (b) the ground truth spectrogram for the singing voice; (c) the separated signing voice spectrogram from our proposed model (DRNN-2 + discrim); (d) the ground truth spectrogram for the music accompaniment; (e) the separated music accompaniment spectrogram from our proposed model (DRNN-2 + discrim).

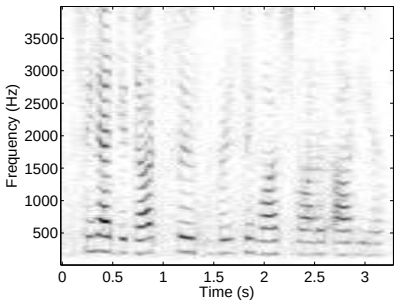
spectral features to *bleed* through partially from one source to the other. We address this issue by proposing a discriminative network training criterion for reducing the interference, possibly at the cost of increased artifacts. Suppose



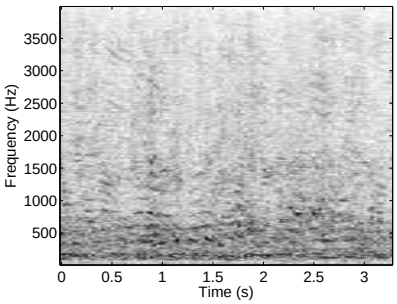
(a) Mixture



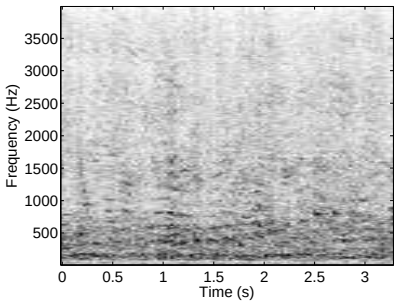
(b) Original speech



(c) Recovered speech



(d) Original noise



(e) Recovered noise

Figure 3.5: A speech denoising example using the TIMIT dataset. (a) The mixture (speech and babble noise) magnitude spectrogram for a test clip in TIMIT; (b) the ground truth spectrogram for the speech; (c) the separated speech spectrogram from our proposed model (DNN); (d) the ground truth spectrogram for the babble noise; (e) the separated babble noise spectrogram from our proposed model (DNN).

that we define

$$J_{\mathrm{DIS}} = -(1-\gamma)\ln p_{12}(\mathbf{y}) - \gamma D_{\mathrm{KL}}(p_{12}\|p_{21}) \tag{3.8}$$

where $0 \leq \gamma \leq 1$ is a regularization constant. $p_{12}(\mathbf{y})$ is the likelihood of the training data under the assumption that the neural net computes the MSE estimate of each feature vector (i.e., its conditional expected value given

knowledge of the mixture), and that all residual noise is Gaussian with unit covariance, thus

$$\ln p_{12}(\mathbf{y}) = -\frac{1}{2}\sum_{t=1}^{T}\left(\|\mathbf{y}_{\mathbf{1}_t} - \tilde{\mathbf{y}}_{\mathbf{1}_t}\|^2 + \|\mathbf{y}_{\mathbf{2}_t} - \tilde{\mathbf{y}}_{\mathbf{2}_t}\|^2\right) \tag{3.9}$$

The discriminative term, $D_{\mathrm{KL}}(p_{12}\|p_{21})$, is a point estimate of the KL divergence between the likelihood model $p_{12}(\mathbf{y})$ and the model $p_{21}(\mathbf{y})$, where the latter is computed by swapping affiliation of spectra to sources, thus

$$\begin{aligned} D_{\mathrm{KL}}(p_{12}\|p_{21}) = \quad & \frac{1}{2}\sum_{t=1}^{T}\left(\|\mathbf{y}_{\mathbf{1}_t} - \tilde{\mathbf{y}}_{\mathbf{2}_t}\|^2 + \|\mathbf{y}_{\mathbf{2}_t} - \tilde{\mathbf{y}}_{\mathbf{1}_t}\|^2 - \right. \\ & \left. \|\mathbf{y}_{\mathbf{1}_t} - \tilde{\mathbf{y}}_{\mathbf{1}_t}\|^2 - \|\mathbf{y}_{\mathbf{2}_t} - \tilde{\mathbf{y}}_{\mathbf{2}_t}\|^2\right) \end{aligned} \tag{3.10}$$

Combining Eqs. (3.8)–(3.10) gives a discriminative criterion with a simple and useful form:

$$\begin{aligned} J_{\mathrm{DIS}} = \quad & \frac{1}{2}\sum_{t=1}^{T}\left(\|\mathbf{y}_{\mathbf{1}_t} - \tilde{\mathbf{y}}_{\mathbf{1}_t}\|^2 + \|\mathbf{y}_{\mathbf{2}_t} - \tilde{\mathbf{y}}_{\mathbf{2}_t}\|^2 - \right. \\ & \left. \gamma\|\mathbf{y}_{\mathbf{1}_t} - \tilde{\mathbf{y}}_{\mathbf{2}_t}\|^2 - \gamma\|\mathbf{y}_{\mathbf{2}_t} - \tilde{\mathbf{y}}_{\mathbf{1}_t}\|^2\right) \end{aligned} \tag{3.11}$$

Although Eq. (3.7) directly optimizes the reconstruction objective, adding the extra term $-\gamma\|\mathbf{y}_{\mathbf{1}_t} - \tilde{\mathbf{y}}_{\mathbf{2}_t}\|^2 - \gamma\|\mathbf{y}_{\mathbf{2}_t} - \tilde{\mathbf{y}}_{\mathbf{1}_t}\|^2$ in Eq. (3.11) further penalizes the interference from the other source, and can be viewed as a regularizer of Eq. (3.7) during the training. From our experimental results, we generally achieve higher source to interference ratio while maintaining similar or higher source to distortion ratio and source to artifacts ratio.

3.4 Experiments

In this section, we evaluate the proposed models on three monaural source separation tasks: speech separation, singing voice separation, and speech denoising. We quantitatively evaluate the source separation performance using three metrics: Source to Interference Ratio (SIR), Source to Artifacts Ratio (SAR), and Source to Distortion Ratio (SDR), according to the BSS-EVAL metrics [63]. SDR is the ratio of the power of the input signal to the power

of the difference between input and reconstructed signals. SDR is therefore exactly the same as the classical measure "signal-to-noise ratio" (SNR), and SDR reflects the overall separation performance. In addition to SDR, SIR reports errors caused by failures to fully remove the interfering signal, and SAR reports errors caused by extraneous artifacts introduced during the source separation procedure. In the past decade, the source separation community has been seeking more precise information about source reconstruction performance; in particular, recent work [1, 64] and competitions (e.g., Signal Separation Evaluation Campaign (SiSEC), Music Information Retrieval Evaluation (MIREX)) now separately report SDR, SIR, and SAR for objectively comparing different approaches. Note that these measures are defined so that distortion = interference + artifacts. For the speech denoising task, we additionally compute the short-time objective intelligibility measure (STOI) which is a quantitative estimate of the intelligibility of the denoised speech [65]. Higher values of SDR, SAR, SIR, and STOI represent higher separation quality.

We use the abbreviations DRNN-k and sRNN to denote the DRNN with the recurrent connection at the k-th hidden layer, or at all hidden layers, respectively. Examples are shown in Figure 3.1. We select the architecture and hyperparameters (the γ parameter in Eq. (3.11), the mini-batch size, L-BFGS iterations, and the circular shift size of the training data) based on the development set performance.

We optimize our models by back-propagating the gradients with respect to the training objective in Eq. (3.11). We use the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm [66] to train the models from random initialization. Examples of the separation results are shown in Figures 3.3, 3.4, and 3.5. The sound examples and source codes of this work are available online.[1]

3.4.1 Speech Separation Setting

We evaluate the performance of the proposed approaches for a monaural speech separation task using the TSP corpus [44]. There are 1444 utterances, with average length 2.372 s, spoken by 24 speakers (half male and

[1] https://sites.google.com/site/deeplearningsourceseparation/

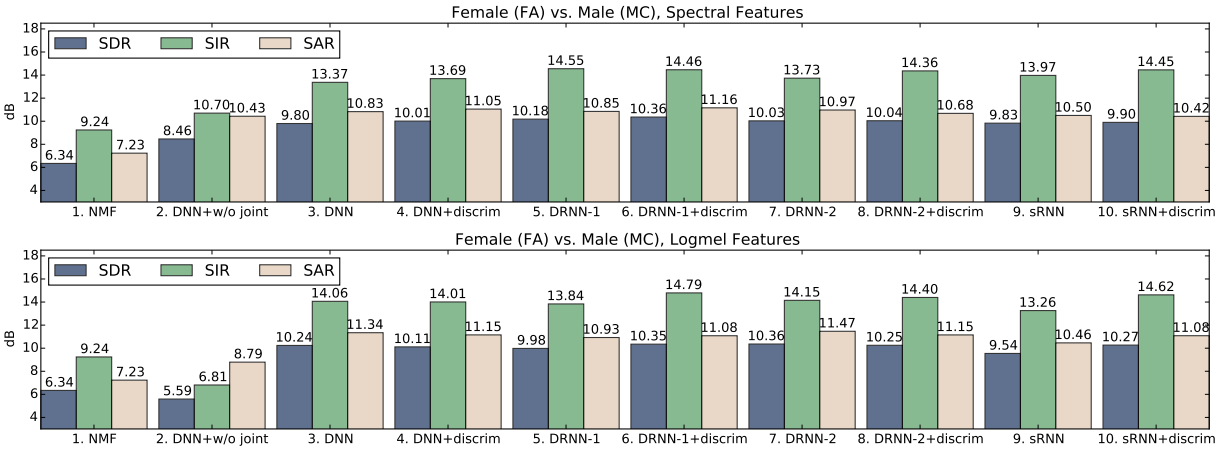


Figure 3.6: TSP speech separation results (Female vs. Male), where "w/o joint" indicates the network is not trained with the masking layer, and "discrim" indicates the training with the discriminative objective. Note that the NMF model uses spectral features.

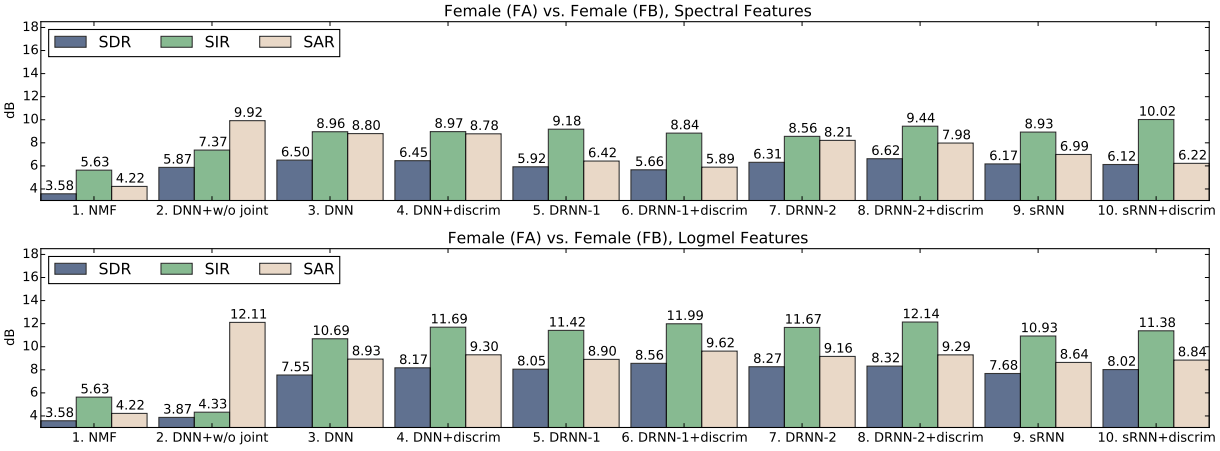


Figure 3.7: TSP speech separation results (Female vs. Female), where "w/o joint" indicates the network is not trained with the masking layer, and "discrim" indicates the training with the discriminative objective. Note that the NMF model uses spectral features.

half female). We choose four speakers, `FA` (female), `FB` (female), `MC` (male), and `MD` (male), from the TSP speech database. After concatenating together 60 sentences for each speaker, we use 80% of the signals for training, 10% for development, and 10% for testing. The signals are downsampled to 16 kHz. The neural networks are trained on three different mixing cases: FA versus MC, FA versus FB, and MC versus MD. Since FA and FB are female speakers while MC and MD are male, the latter two cases are expected to

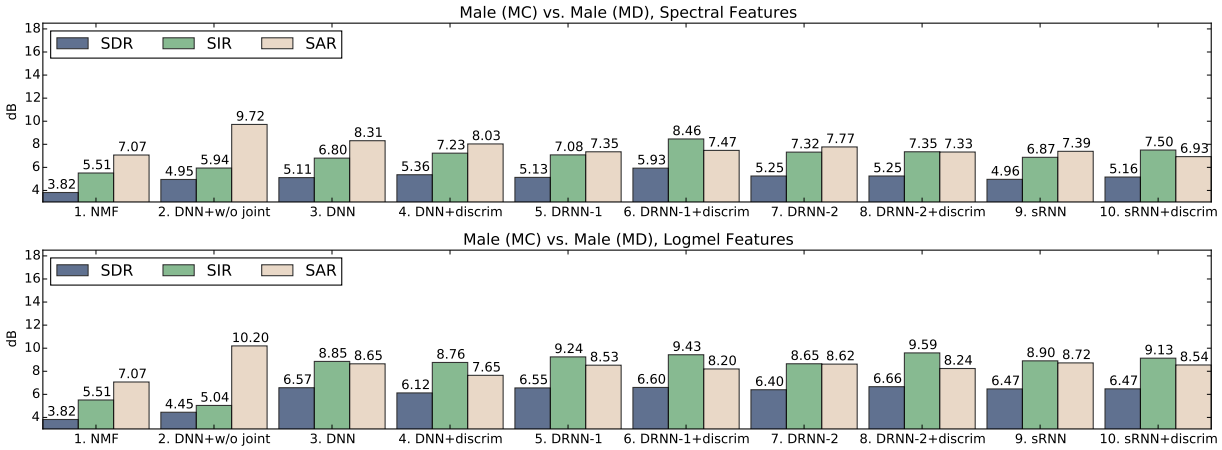


Figure 3.8: TSP speech separation results (Male vs. Male), where "w/o joint" indicates the network is not trained with the masking layer, and "discrim" indicates the training with the discriminative objective. Note that the NMF model uses spectral features.

be more difficult due to the similar frequency ranges from the same gender. After normalizing the signals to have 0 dB input SNR, the neural networks are trained to learn the mapping between an input mixture spectrum and the corresponding pair of clean spectra.

As for the NMF experiments, 10 to 100 speaker-specific basis vectors are trained from the training part of the signals. The optimal number of basis vectors is chosen based on the development set. We empirically found that using 20 basis vectors achieves the best performance on the development set in the three different mixing cases. The NMF separation is done by fixing the known speakers' basis vectors during the test procedure and learning the speaker-specific activation matrices.

We explore two different types of input features: spectral and log-mel filterbank features. The spectral representation is extracted using a 1024-point shot-time Fourier transform (STFT) with 50% overlap. In the speech recognition literature [67], the log-mel filterbank is found to provide lower word-error-rate compared to mel-frequency cepstral coefficients (MFCC) and log FFT bins. The 40-dimensional log-mel representation and the first- and second-order derivative features are used in the experiments. For the neural network training, in order to increase the variety of training samples, we circularly shift (in the time domain) the signals of one speaker and mix them with utterances from the other speaker.

3.4.2 Speech Separation Results

We use the standard NMF with the generalized KL-divergence metric as our baseline. We report the best NMF results among models with different basis vectors, as shown in the first column of Figures 3.6, 3.7, and 3.8. Note that NMF uses spectral features, and hence the results in the second row (log-mel features) of each figure are the same as the first row (spectral features).

The speech separation results of the cases, FA versus MC, FA versus FB, and MC versus MD, are shown in Figures 3.6, 3.7, and 3.8, respectively. We train models with two hidden layers of 300 hidden units using features with a context window size of one frame (one frame within a window), where the architecture and the hyperparameters are chosen based on the development set performance. We report the results of single frame spectra and log-mel features in the top and bottom rows of Figures 3.6, 3.7, and 3.8, respectively. To further understand the strength of the models, we compare the experimental results in several aspects. In the second and third columns of Figures 3.6, 3.7, and 3.8, we examine the effect of joint optimization of the masking layer and the DNN. Jointly optimizing the masking layer significantly outperforms the cases where the masking layer is applied separately (the second column). In the FA vs. FB case, DNN without joint optimization of the masking layer achieves high SAR, but results in low SDR and SIR. In the top and bottom rows of Figures 3.6, 3.7, and 3.8, we compare the results between spectral features and log-mel features. In the joint optimization case, (columns 3–10), log-mel features achieve higher SDRs, SIRs, and SARs compared to spectral features. On the other hand, spectral features achieve higher SDRs and SIRs in the case where DNN is not jointly trained with a masking layer, as shown in the second column of Figures 3.6, 3.7, and 3.8. In the FA vs. FB and MC vs. MD cases, the log-mel features outperform spectral features greatly.

Between columns 3, 5, 7, and 9, and columns 4, 6, 8, and 10 of Figures 3.6, 3.7, and 3.8, we make comparisons between various network architectures, including DNN, DRNN-1, DRNN-2, and sRNN. In many cases, recurrent neural network models (DRNN-1, DRNN-2, or sRNN) outperform DNN. Between columns 3 and 4, columns 5 and 6, columns 7 and 8, and columns 9 and 10 of Figures 3.6, 3.7, and 3.8, we compare the effectiveness of using the discriminative training criterion, i.e., $\gamma > 0$ in Eq. (3.11). In most cases,

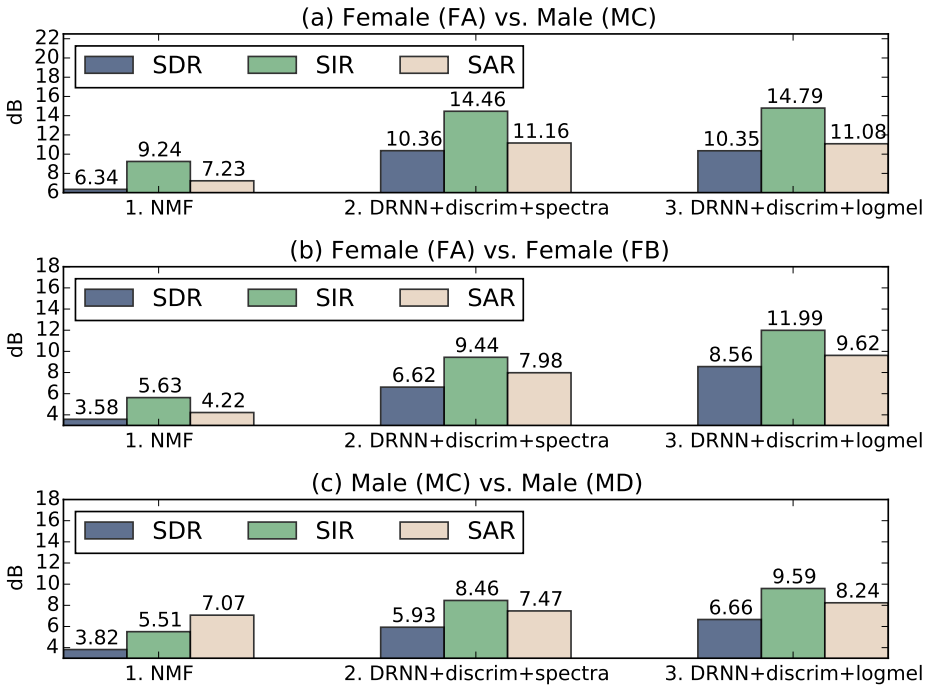


Figure 3.9: TSP speech separation result summary. We compare the results under three settings, (a) Female vs. Male, (b) Female vs. Female, and (c) Male vs. Male, using the NMF model, the best DRNN+discrim architecture with spectra features, and the best DRNN+discrim architecture with log-mel features.

SIRs are improved. The results match our expectation when we design the objective function. However, it also leads to some artifacts which result in slightly lower SARs in some cases. Empirically, the value γ is in the range of 0.01–0.1 in order to achieve SIR improvements and maintain reasonable SAR and SDR.

Finally, we compare the NMF results with our proposed models with the best architecture using spectral and log-mel features, as shown in Figure 3.9. NMF models learn activation matrices from different speakers and hence perform poorly in the same sex speech separation cases, FA vs. FB and MC vs. MD. Our proposed models greatly outperform NMF models for all three cases. Especially for the FA vs. FB case, our proposed model achieves around 5 dB SDR gain compared to the NMF model while maintaining higher SIR and SAR.

3.4.3 Singing Voice Separation Setting

We apply our models to a singing voice separation task, where one source is the singing voice and the other source is the background music. The goal is to separate singing voice from music recordings.

We evaluate our proposed system using the MIR-1K dataset [45].[2] A thousand song clips are encoded at a sampling rate of 16 KHz, with a duration from 4 to 13 seconds. The clips were extracted from 110 Chinese karaoke songs performed by both male and female amateurs. There are manual annotations of the pitch contours, lyrics, indices and types for unvoiced frames, and the indices of the vocal and non-vocal frames; none of the annotations were used in our experiments. Each clip contains the singing voice and the background music in different channels.

Following the evaluation framework in [34, 35], we use 175 clips sung by one male and one female singer ("abjones" and "amy") as the training and development set.[3] The remaining 825 clips of 17 singers are used for testing. For each clip, we mixed the singing voice and the background music with equal energy, i.e., 0 dB SNR.

To quantitatively evaluate the source separation results, we report the overall performance via Global NSDR (GNSDR), Global SIR (GSIR), and Global SAR (GSAR), which are the weighted means of the NSDRs, SIRs, SARs, respectively, over all test clips weighted by their length. Normalized SDR (NSDR) [68] is defined as:

$$\mathrm{NSDR}(\hat{\mathbf{v}}, \mathbf{v}, \mathbf{x}) = \mathrm{SDR}(\hat{\mathbf{v}}, \mathbf{v}) - \mathrm{SDR}(\mathbf{x}, \mathbf{v}) \tag{3.12}$$

where $\hat{\mathbf{v}}$ is the estimated singing voice, $\mathbf{v}$ is the original clean singing voice, and $\mathbf{x}$ is the mixture. NSDR is for estimating the improvement of the SDR between the preprocessed mixture $\mathbf{x}$ and the separated singing voice $\hat{\mathbf{v}}$.

For the neural network training, in order to increase the variety of training samples, we circularly shift (in the time domain) the signals of the singing voice and mix them with the background music. In the experiments, we use magnitude spectra as input features to the neural network. The spectral representation is extracted using a 1024-point STFT with 50% overlap.

[2] https://sites.google.com/site/unvoicedsoundseparation/mir-1k

[3] Four clips, abjones_5_08, abjones_5_09, amy_9_08, amy_9_09, are used as the development set for adjusting the hyperparameters.

Table 3.1: MIR-1K separation result comparison using deep neural networks with single source as a target and using two sources as targets (with and without joint optimization of the masking layers and the DNNs).

Model (num. of output sources, joint optimization)	GNSDR	GSIR	GSAR
DNN (1, no)	5.64	8.87	9.73
DNN (2, no)	6.44	9.08	11.26
DNN (2, yes)	6.93	10.99	10.15

Empirically, we found that using log-mel filterbank features or log power spectrum provide worse performance than using magnitude spectra in the singing voice separation task.

3.4.4 Singing Voice Separation Results

In this section, we compare various deep learning models from several aspects, including the effect of different output formats, the effect of different deep recurrent neural network structures, and the effect of discriminative training.

For simplicity, unless mentioned explicitly, we report the results using three hidden layers of 1000 hidden units deep neural networks with the mean squared error criterion, joint optimization of the masking layer, and 10 K samples as the circular shift step size using features with a context window size of three frames (three frames within a window).

Table 3.1 presents the results with different output layer formats. We compare using single source as a target (row 1) and using two sources as targets in the output layer (row 2 and row 3). We observe that modeling two sources simultaneously provides higher performance in GNSDR, GSIR, and GSAR. Comparing row 2 and row 3 in Table 3.1, we observe that jointly optimizing the masking layer and the DRNN further improves the results.

Table 3.2 presents the results of different deep recurrent neural network architectures (DNN, DRNN with different recurrent connections, and sRNN) with and without discriminative training. We can observe that discriminative training further improves GSIR while maintaining similar GNSDR and GSAR.

Finally, we compare our best results with other previous work under the same setting. Table 3.3 shows the results with unsupervised and supervised

Table 3.2: MIR-1K separation result comparison for the effect of discriminative training using different architectures. "discrim" denotes the models with discriminative training.

Model	GNSDR	GSIR	GSAR
DNN	6.93	10.99	10.15
DRNN-1	7.11	11.74	9.93
DRNN-2	7.27	11.98	9.99
DRNN-3	7.14	11.48	10.15
sRNN	7.09	11.72	9.88
DNN + discrim	7.09	12.11	9.67
DRNN-1 + discrim	7.21	12.76	9.56
DRNN-2 + discrim	7.45	13.08	9.68
DRNN-3 + discrim	7.09	11.69	10.00
sRNN + discrim	7.15	12.79	9.39

Table 3.3: MIR-1K separation result comparison between our models and previous proposed approaches. "discrim" denotes the models with discriminative training.

Unsupervised			
Model	GNSDR	GSIR	GSAR
RPCA [32]	3.15	4.43	11.09
RPCAh [36]	3.25	4.52	11.10
RPCAh + FASST [36]	3.84	6.22	9.19
Supervised			
Model	GNSDR	GSIR	GSAR
MLRR [35]	3.85	5.63	10.70
RNMF [34]	4.97	7.66	10.03
DRNN-2	**7.27**	**11.98**	**9.99**
DRNN-2 + discrim	**7.45**	**13.08**	**9.68**

settings. Our proposed models achieve 2.30–2.48 dB GNSDR gain, 4.32–5.42 dB GSIR gain with similar GSAR performance, compared with the RNMF model [34].

3.4.5 Speech Denoising Setting

We apply the proposed framework to a speech denoising task, where one source is the clean speech and the other source is the noise. The goal of

the task is to separate clean speech from noisy speech. In the experiments, we use magnitude spectra as input features to the neural network. The spectral representation is extracted using a 1024-point STFT with 50% overlap. Empirically, we found that log-mel filterbank features provide worse performance than magnitude spectra. Unless mentioned explicitly, we use two hidden layers of 1000 hidden units deep neural networks with the mean squared error criterion, joint optimization of the masking layer, and 10 K samples as the circular shift step size, using features with a context window size of one frame (one frame within a window). The model is trained and tested on 0 dB mixtures, without input normalization.

To understand the effect of degradation in the mismatch condition, we set up the experimental recipe as follows. We use a hundred utterances spanning ten different speakers from the TIMIT database. We also use a set of five noises: Airport, Train, Subway, Babble, and Drill. We generate a number of noisy speech recordings by selecting random subsets of noises and overlaying them with speech signals. We also specify the signal to noise ratio when constructing the noisy mixtures. After we complete the generation of the noisy signals, we split them into a training set and a test set.

3.4.6 Speech Denoising Results

In the following experiments, we examine the effect of the proposed methods under various scenarios. We first evaluate various architectures using 0 dB SNR inputs, as shown in Figure 3.10. We can observe that the recurrent neural network architectures (DRNN-1, DRNN-2, sRNN) achieve similar performance compared to the DNN model. Including the discriminative training objective improves SDR and SIR, but results in slightly degraded SAR and similar STOI values.

To further evaluate the robustness of the model, we examine our model under a variety of situations in which it is presented with unseen data, such as unseen SNRs, speakers, and noise types. These tests provide a way of understanding the performance of the proposed approach under mismatched conditions. In Figure 3.11, we show the robustness of this model under various SNRs. The model is trained on 0 dB SNR mixtures and it is evaluated on mixtures ranging from 20 dB SNR to -18 dB SNR. We compare the results

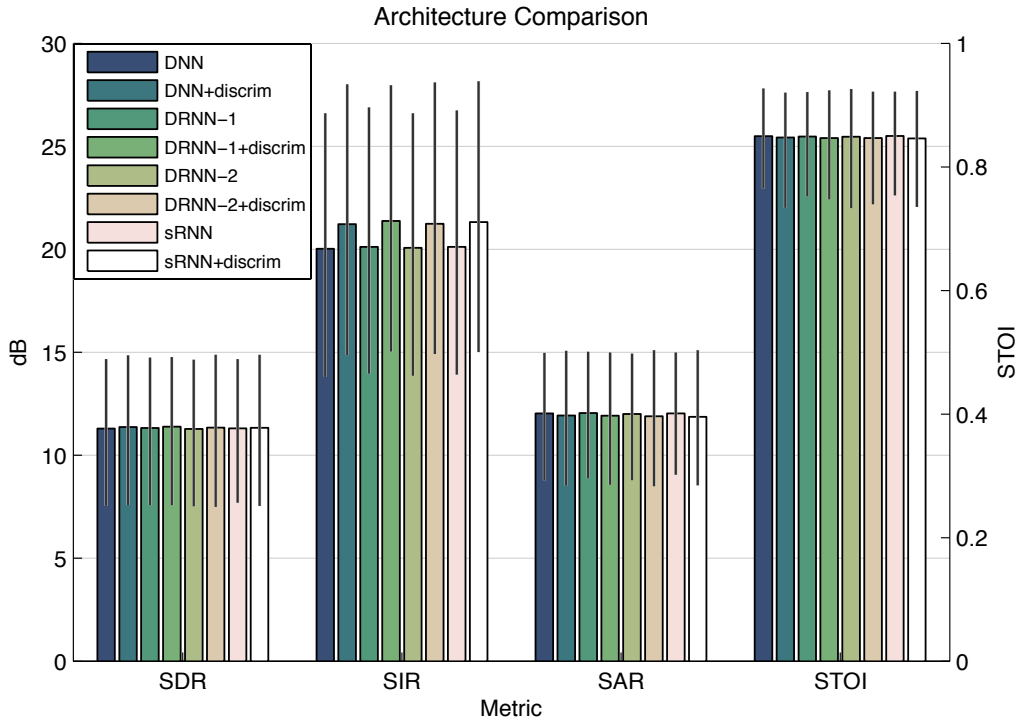


Figure 3.10: Speech denoising architecture comparison, where "+discrim" indicates the training with the discriminative objective, and the bars show average values and the vertical lines on the bars denote minimum and maximum observed values. Models are trained and tested on 0 dB SNR inputs. The average STOI score for unprocessed mixtures is 0.675.

between NMF, DNN without joint optimization of the masking layer, and DNN with joint optimization of the masking layer. In most cases, DNN with joint optimization achieves the best results, especially under low SNR inputs. For the 20 dB SNR case, NMF achieves the best performance. DNN without joint optimization achieves the highest SIR given high SNR inputs, though SDR, SAR, and STOI are lower than the DNN with joint optimization. Note that in our approach, joint optimization of the time-frequency masks and DNNs can be viewed as a way to directly incorporate the FFT-MASK targets [53] into the DNNs for both speech and noise, where authors in [53] found FFT-MASK has achieved better performance compared to other targets in speech denoising tasks.

Next, we evaluate the models under three different cases: (1) the testing noise is unseen in training, (2) the testing speaker is unseen in training, and (3) both the testing noise and testing speaker are unseen in training stage.

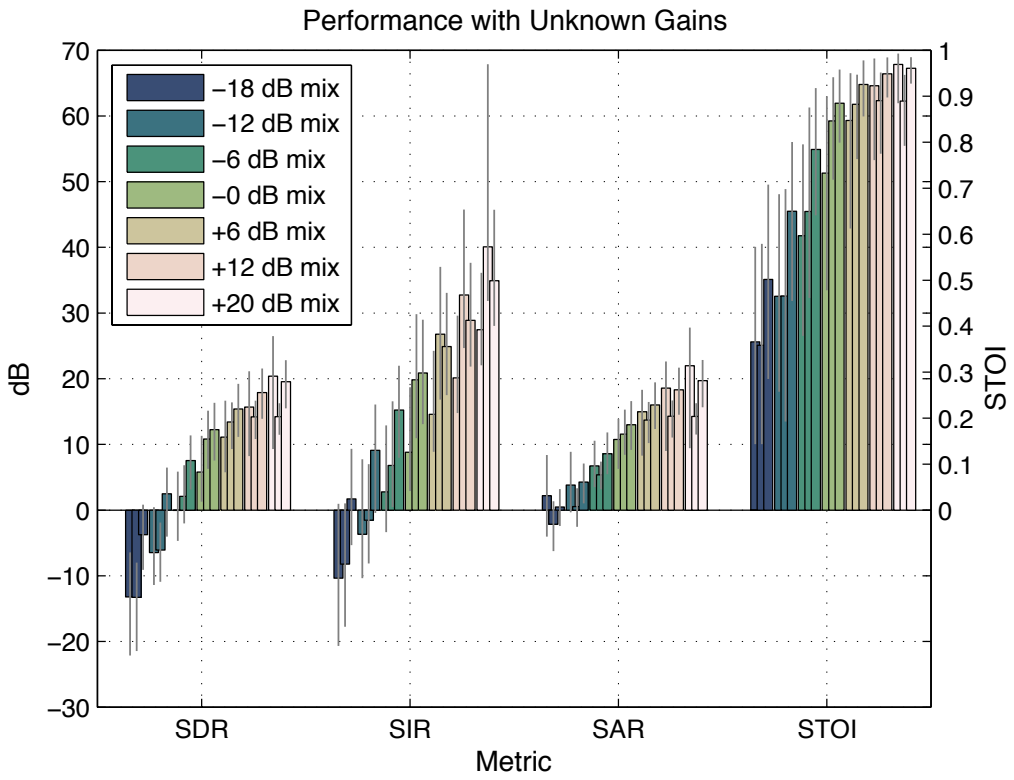
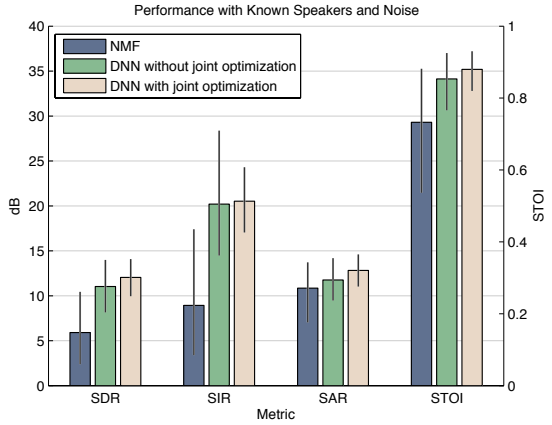


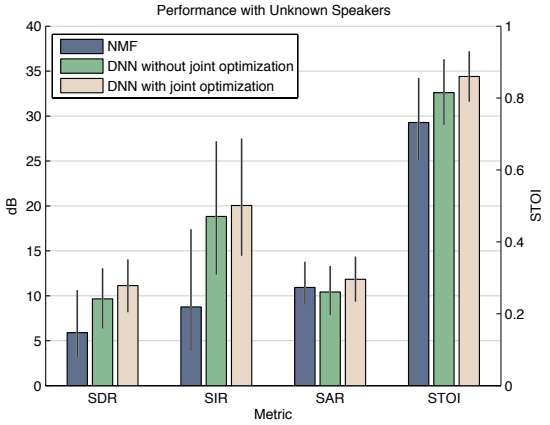
Figure 3.11: Speech denoising using multiple SNR inputs and testing on a model that is trained on 0 dB SNR, where the bars show average values and the vertical lines on the bars denote minimum and maximum observed values. The left/back, middle, right/front bars in each pair show the results of NMF, DNN without joint optimization of the masking layer [1], and DNN with joint optimization of the masking layer, respectively. The average STOI scores for unprocessed mixtures at -18 dB, -12 dB, -6 dB, 0 dB, 6 dB, 12 dB, and 20 dB SNR are 0.370, 0.450, 0.563, 0.693, 0.815, 0.903, and 0.968, respectively.

For the unseen noise case, we train the model on mixtures with Babble, Airport, Train and Subway noises, and evaluate it on mixtures that include a Drill noise (which is significantly different from the training noises in both spectral and temporal structure). For the unknown speaker case, we hold out some of the speakers from the training data. For the case where both the noise and speaker are unseen, we use the combination of the above.

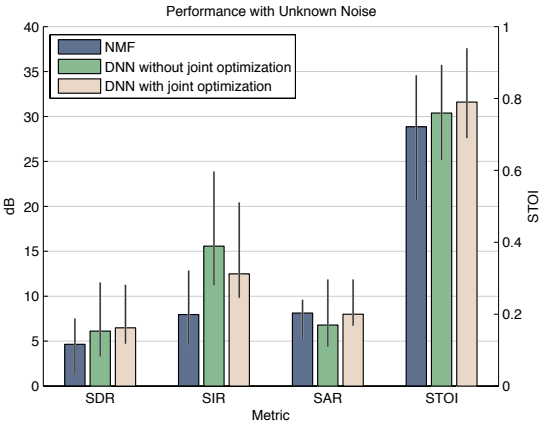
We compare our proposed approach with the NMF model and DNN without joint optimization of the masking layer [1]. The models are trained and tested on 0 dB SNR inputs, and these experimental results are shown in Figure 3.12. For the unknown speaker case, as shown in Figure 3.12 (b), we



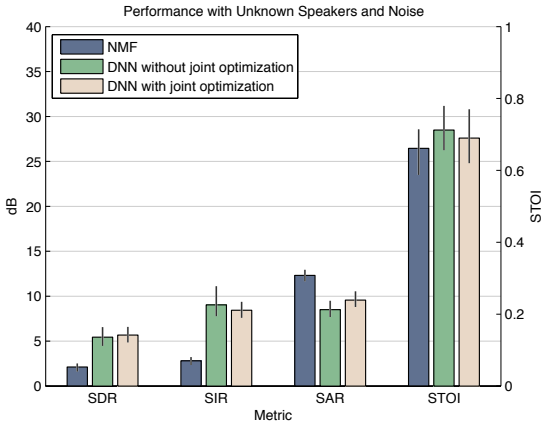
(a) Known speakers and noise



(b) Unknown speakers



(c) Unknown noise



(d) Unknown speakers and noise

Figure 3.12: Speech denoising experimental results comparison between NMF, DNN without joint optimization of the masking layer, and DNN with joint optimization of the masking layer, given 0 dB SNR inputs, when used on data that is not represented in training. The bars show average values and the vertical lines on the bars denote minimum and maximum observed values. We show the separation results of (a) known speakers and noise, (b) unseen speakers, (c) unseen noise, and (d) unseen speakers and noise. The average STOI scores for unprocessed mixtures for cases (a), (b), (c), and (d) are 0.698, 0.686, 0.705, and 0.628, respectively.

observe that there is only a mild degradation in performance for all models compared to the case where the speakers are known in Figure 3.12 (a). The results suggest that the approaches can be easily used in speaker variant situations. In Figure 3.12 (c), with the unseen noise, we observe a larger degradation in results, which is expected due to the drastically different nature of the noise type. For the case where both the noise and speakers are unknown, as shown in Figure 3.12 (d), all three models achieve the worst performance compare to the other cases. Overall, the proposed approach generalizes well across speakers and achieves higher source separation performance, especially in SDRs, compared to the baseline models under various conditions.

3.4.7 Discussion

Throughout the experiments in speech separation, singing voice separation, and speech denoising tasks, we have seen significant improvement over the baseline models under various settings, by the use of joint optimization of time-frequency masks with deep recurrent neural networks and the discriminative training objective. By jointly optimizing time-frequency masks with deep recurrent neural networks, the proposed end-to-end system outperforms baseline models (such as NMF, DNN models without joint optimization) in matched and mismatched conditions. Given audio signals are time series in nature, we explore various recurrent neural network architectures to capture temporal information and further enhance performance. Though there are extra memory and computational costs compared to feed-forward neural networks, DRNNs achieve extra gains, especially in the speech separation (0.5 dB SDR gain) and singing voice separation (0.34 dB GNSDR gain) tasks. Similar observations can be found in related work using LSTM models [49, 69], where the authors observe significant improvements using recurrent neural networks compared with DNN models. Our proposed discriminative objective can be viewed as a regularization term towards the original mean-squared error objective. By enforcing the similarity between targets and predictions of the same source and dissimilarity between targets and predictions of competing sources, we observe that interference is further reduced while maintaining similar or higher SDRs and SARs.

3.5 Summary

In this chapter, we explore various deep learning architectures, including deep neural networks and deep recurrent neural networks for monaural source separation problems. We enhance the performance by jointly optimizing a soft time-frequency mask layer with the networks in an end-to-end fashion and exploring a discriminative training criterion. We evaluate our proposed method for speech separation, singing voice separation, and speech denoising tasks. Overall, our proposed models achieve 2.30–4.98 dB SDR gain compared to the NMF baseline, while maintaining higher SIRs and SARs in the TSP speech separation task. In the MIR-1K singing voice separation task, our proposed models achieve 2.30–2.48 dB GNSDR gain and 4.32–5.42 dB GSIR gain, compared to the previously proposed methods, while maintaining similar GSARs. Moreover, our proposed method also outperforms NMF and DNN baselines in various mismatch conditions in the TIMIT speech denoising task. To further improve the performance, one direction is to further explore using LSTMs to model longer temporal information [70], which has shown great performance compared to conventional recurrent neural networks as LSTM has properties of avoiding vanishing gradient properties. In addition, our proposed models can also be applied to many other applications such as robust ASR.

Chapter 4

Deep Learning for Structured Inputs: Deep Structured Semantic Models

4.1 Introduction

Modern search engines retrieve web documents mainly by matching keywords in documents with those in search queries. However, lexical matching can be inaccurate due to the fact that a concept is often expressed using different vocabularies and language styles in documents and queries.

Latent semantic models such as latent semantic analysis (LSA) are able to map a query to its relevant documents at the semantic level where lexical matching often fails (e.g., [40, 71–74]). These latent semantic models address the language discrepancy between web documents and search queries by grouping different terms that occur in a similar context into the same semantic cluster. Thus, a query and a document, represented as two vectors in the lower-dimensional semantic space, can still have a high similarity score even if they do not share any term. Extending from LSA, probabilistic topic models such as probabilistic LSA (PLSA) and Latent Dirichlet Allocation (LDA) have also been proposed for semantic matching [40, 72]. However, these models are often trained in an unsupervised manner using an objective function that is only loosely coupled with the evaluation metric for the retrieval task. Thus the performance of these models on web search tasks is not as good as originally expected.

Recently, two lines of research have been conducted to extend the aforementioned latent semantic models, which will be briefly reviewed below.

First, clickthrough data, which consists of a list of queries and their clicked documents, is exploited for semantic modeling so as to bridge the language discrepancy between search queries and web documents [75, 76]. For example, Gao et al. [76] propose the use of Bi-Lingual Topic Models (BLTMs) and linear Discriminative Projection Models (DPMs) for query-document match-

ing at the semantic level. These models are trained on clickthrough data using objectives that tailor to the document ranking task. More specifically, BLTM is a generative model which requires that a query and its clicked documents not only share the same distribution over topics, but also contain similar factions of words assigned to each topic. In contrast, the DPM is learned using the S2Net algorithm [77] that follows the pairwise learning-to-rank paradigm outlined in [78]. After projecting term vectors of queries and documents into concept vectors in a low-dimensional semantic space, the concept vectors of the query and its clicked documents have a smaller distance than that of the query and its unclicked documents. Gao et al. [76] report that both BLTM and DPM outperform significantly the unsupervised latent semantic models, including LSA and PLSA, in the document ranking task. However, the training of BLTM, though using clickthrough data, is to maximize a log-likelihood criterion which is sub-optimal for the evaluation metric for document ranking. On the other hand, the training of DPM involves large-scale matrix multiplications. The sizes of these matrices often grow quickly with the vocabulary size, which could be of an order of millions in web search tasks. In order to make the training time tolerable, the vocabulary was pruned aggressively. Although a small vocabulary makes the models trainable, it leads to suboptimal performance.

In the second line of research, Salakhutdinov and Hinton extended the semantic modeling using deep auto-encoders [79]. They demonstrated that hierarchical semantic structure embedded in the query and the document can be extracted via deep learning. Superior performance to the conventional LSA is reported [79]. However, the deep learning approach they used still adopts an unsupervised learning method where the model parameters are optimized for the reconstruction of the documents rather than for differentiating the relevant documents from the irrelevant ones for a given query. As a result, the deep learning models do not significantly outperform the baseline retrieval models based on keyword matching. Moreover, the semantic hashing model also faces the scalability challenge regarding large-scale matrix multiplication. We will show in this chapter that the capability of learning semantic models with large vocabularies is crucial to obtain good results in real-world web search tasks.

In this chapter, extending from both research lines discussed above, we propose a series of Deep Structured Semantic Models (DSSM) for web search

tasks. More specifically, our best model uses a deep neural network (DNN) to rank a set of documents for a given query as follows. First, a nonlinear projection is performed to map the query and the documents to a common semantic space. Then, the relevance of each document given the query is calculated as the cosine similarity between their vectors in that semantic space. The neural network models are discriminatively trained using the clickthrough data such that the conditional likelihood of the clicked document given the query is maximized. Different from the previous latent semantic models that are learned in an unsupervised fashion, our models are optimized directly for web document ranking, and thus give superior performance, as we will show shortly. Furthermore, to deal with large vocabularies, we propose the so-called word hashing method, through which the high-dimensional term vectors of queries or documents are projected to low-dimensional letter-based n-gram vectors with little information loss. In our experiments, we show that, by adding this extra layer of representation in semantic models, word hashing enables us to learn discriminatively the semantic models with large vocabularies, which are essential for web search tasks. We evaluated the proposed DSSMs on a web document ranking task using a real-world dataset. The results show that our best model outperforms all the competing methods with a significant margin of 2.5-4.3% in NDCG@1.

In the rest of this chapter, Section 4.2 reviews related work. Section 4.3 describes our DSSM for web search tasks. Section 4.4 presents the experimental results.

4.2 Related Work

Our work is based on two recent extensions to the latent semantic models for IR. The first is the exploration of the clickthrough data for learning latent semantic models in a supervised fashion [76]. The second is the introduction of deep learning methods for semantic modeling [79].

4.2.1 Latent Semantic Models and the Use of Clickthrough Data

The use of latent semantic models for query-document matching is a long-standing research topic in the IR community. Popular models can be grouped into two categories, linear projection models and generative topic models, which we will review in turn. The most well-known linear projection model for IR is LSA [71]. By using the singular value decomposition (SVD) of a document-term matrix, a document (or a query) can be mapped to a low-dimensional concept vector $\hat{\mathbf{d}} = \mathbf{A}^{\mathrm{T}}\mathbf{d}$, where the $\mathbf{A}$ is the projection matrix. In document search, the relevance score between a query and a document, represented respectively by term vectors $\mathbf{q}$ and $\mathbf{d}$, is assumed to be proportional to their cosine similarity score of the corresponding concept vectors $\hat{\mathbf{q}}$ and $\hat{\mathbf{d}}$, according to the projection matrix $\mathbf{A}$

$$\mathrm{sim}_{\mathbf{A}}(\mathbf{q}, \mathbf{d}) = \frac{\hat{\mathbf{q}}^{\mathrm{T}}\hat{\mathbf{d}}}{\|\hat{\mathbf{q}}\|\|\hat{\mathbf{d}}\|} \tag{4.1}$$

In addition to latent semantic models, the translation models trained on clicked query-document pairs provide an alternative approach to semantic matching [75]. Unlike latent semantic models, the translation-based approach learns translation relationships directly between a term in a document and a term in a query. Recent studies show that given large amounts of clickthrough data for training, this approach can be very effective [75, 76]. We will also compare our approach with translation models experimentally as reported in Section 4.4.

4.2.2 Deep Learning

Recently, deep learning methods have been successfully applied to a variety of language and information retrieval applications [15, 79–86]. By exploiting deep architectures, deep learning techniques are able to discover from training data the hidden structures and features at different levels of abstractions useful for the tasks. In [79], Salakhutdinov and Hinton extended the LSA model by using a deep network (auto-encoder) to discover the hierarchical semantic structure embedded in the query and the document. They proposed a semantic hashing (SH) method which uses bottleneck features

learned from the deep auto-encoder for information retrieval. These deep models are learned in two stages. First, a stack of generative models (i.e., the restricted Boltzmann machine) are learned to map a term vector representation of a document layer-by-layer to a low-dimensional semantic concept vector. Second, the model parameters are fine-tuned so as to minimize the cross entropy error between the original term vector of the document and the reconstructed term vector. The intermediate layer activations are used as features (i.e., bottleneck) for document ranking. Their evaluation shows that the SH approach achieves a superior document retrieval performance to the LSA. However, SH suffers from two problems, and cannot outperform the standard lexical matching based retrieval model (e.g., cosine similarity using TF-IDF term weighting). The first problem is that the model parameters are optimized for the reconstruction of the document term vectors rather than for differentiating the relevant documents from the irrelevant ones for a given query. Second, in order to make the computational cost manageable, the term vectors of documents consist of only the most-frequent 2000 words. In Section 4.3, we show our solutions to these two problems.

4.3 Deep Structured Semantic Models for Web Search

4.3.1 DNN for Computing Semantic Features

The typical DNN architecture we have developed for mapping the raw text features into the features in a semantic space is shown in Figure 4.1. The input (raw text features) to the DNN is a high-dimensional term vector, e.g., raw counts of terms in a query or a document without normalization, and the output of the DNN is a concept vector in a low-dimensional semantic feature space. This DNN model is used for web document ranking as follows: (1) to map term vectors to their corresponding semantic concept vectors; (2) to compute the relevance score between a document and a query as cosine similarity of their corresponding semantic concept vectors, as shown in Eqs. (4.2), (4.3), and (4.4).

More formally, if we denote x as the input term vector, $\mathbf{y}$ as the output vector, $\mathbf{l}_i$, $i = 1, \ldots, N-1$, as the intermediate hidden layers, $\mathbf{W}_i$ as the i-th

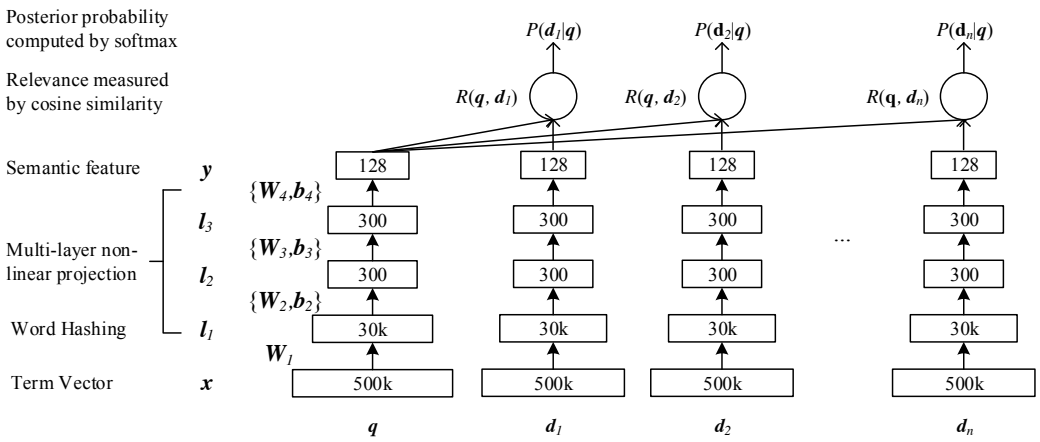


Figure 4.1: Illustration of the Deep Structured Semantic Models (DSSM). DSSM uses a DNN to map high-dimensional sparse text features into low-dimensional dense features in a semantic space. The first hidden layer, with 30 K units, accomplishes word hashing. The word-hashed features are then projected through multiple layers of nonlinear projections. The final layer's neural activities in this DNN form the feature in the semantic space.

weight matrix, and $\mathbf{b}_i$ as the i-th bias term, we have

$$\begin{aligned} \mathbf{l}_1 &= \mathbf{W}_1\mathbf{x} \\ \mathbf{l}_i &= f(\mathbf{W}_i\mathbf{l}_{i-1} + \mathbf{b}_i), \qquad i = 2, \ldots, N-1 \\ \mathbf{y} &= f(\mathbf{W}_N\mathbf{l}_{N-1} + \mathbf{b}_N) \end{aligned} \tag{4.2}$$

where we use the tanh as the activation function at the output layer and the hidden layers $\mathbf{l}_i$, $i = 2, , N-1$:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{4.3}$$

The semantic relevance score between a query $\mathbf{q}$ and a document $\mathbf{d}$ is then measured as:

$$R(\mathbf{q}, \mathbf{d}) = \cos(\mathbf{y}_q, \mathbf{y}_d) = \frac{\mathbf{y}_q^{\mathrm{T}}\mathbf{y}_d}{\|\mathbf{y}_q\|\|\mathbf{y}_d\|} \tag{4.4}$$

where $\mathbf{y}_q$ and $\mathbf{y}_d$ are the concept vectors of the query and the document, respectively. In web search, given the query, the documents are sorted by their semantic relevance scores. Conventionally, the size of the term vector, which can be viewed as the raw bag-of-words features in IR, is identical to that of the vocabulary that is used for indexing the web document collection.

The vocabulary size is usually very large in real-world web search tasks. Therefore, when using a term vector as the input, the size of the input layer of the neural network would be unmanageable for inference and model training. To address this problem, we have developed a method called word hashing for the first layer of the DNN, as indicated in the lower portion of Figure 4.1. This layer consists of only linear hidden units in which the weight matrix of a very large size is not learned. In Section 4.3.2, we describe the word hashing method in detail.

4.3.2 Word Hashing

The word hashing method described here aims to reduce the dimensionality of the bag-of-words term vectors. It is based on letter n-grams, and is a new method developed especially for our task. Given a word (e.g. good), we first add word starting and ending marks to the word (e.g. #good#). Then, we break the word into letter n-grams (e.g. letter trigrams: #go, goo, ood, od#). Finally, the word is represented using a vector of letter n-grams. One problem of this method is collision, i.e., two different words could have the same letter n-gram vector representation. Table 4.1 shows some statistics of word hashing on two vocabularies. Compared with the original size of the one-hot vector, word hashing allows us to represent a query or a document using a vector with much lower dimensionality. Take the 40K-word vocabulary as an example. Each word can be represented by a 10,306-dimensional vector using letter trigrams, giving a fourfold dimensionality reduction with few collisions. The reduction of dimensionality is even more significant when the technique is applied to a larger vocabulary. As shown in Table 4.1, each word in the 500K-word vocabulary can be represented by a 30,621-dimensional vector using letter trigrams, a reduction of 16-fold in dimensionality with a negligible collision rate of 0.0044% (22/500,000).

While the number of English words can be unlimited, the number of letter n-grams in English (or other similar languages) is often limited. Moreover, word hashing is able to map the morphological variations of the same word to the points that are close to each other in the letter n-gram space. More importantly, while a word unseen in the training set always causes difficulties in word-based representations, it is not the case where the letter n-gram-based

Table 4.1: Word hashing token size and collision numbers as a function of the vocabulary size and the type of letter n-grams.

	Letter Bigrams		Letter Trigrams	
Word Size	Token Size	Collision	Token Size	Collision
40 K	1107	18	10306	2
500 K	1607	1192	30621	22

representation is used. The only risk is the minor representation collision as quantified in Table 4.1. Thus, letter n-gram-based word hashing is robust to the out-of-vocabulary problem, allowing us to scale-up the DNN solution to the web search tasks where extremely large vocabularies are desirable. We will demonstrate the benefit of the technique in Section 4.4.

In our implementation, the letter n-gram-based word hashing can be viewed as a fixed (i.e., non-adaptive) linear transformation, through which a term vector in the input layer is projected to a letter n-gram vector in the next layer higher up, as shown in Figure 4.1. Since the letter n-gram vector is of a much lower dimensionality, DNN learning can be carried out effectively.

4.3.3 Learning the DSSM

The clickthrough logs consist of a list of queries and their clicked documents. We assume that a query is relevant, at least partially, to the documents that are clicked on for that query. Inspired by the discriminative training approaches in speech and language processing, we thus propose a supervised training method to learn our model parameters, i.e., the weight matrices $\mathbf{W}_i$ and bias vectors $\mathbf{b}_i$ in our neural network as the essential part of the DSSM, so as to maximize the conditional likelihood of the clicked documents given the queries.

First, we compute the posterior probability of a document given a query from the semantic relevance score between them through a softmax function

$$P(\mathbf{d}|\mathbf{q}) = \frac{\exp(\gamma R(\mathbf{q}, \mathbf{d}))}{\sum_{\mathbf{d}' \in \mathbf{D}} \exp(\gamma R(\mathbf{q}, \mathbf{d}'))} \tag{4.5}$$

where γ is a smoothing factor in the softmax function, which is set empirically on a held-out dataset in our experiment. $\mathbf{D}$ denotes the set of candidate documents to be ranked. Ideally, $\mathbf{D}$ should contain all possible

documents. In practice, for each (query, clicked-document) pair, denoted by $(\mathbf{q}, \mathbf{d}^+)$ where $\mathbf{q}$ is a query and $\mathbf{d}^+$ is the clicked document, we approximate $\mathbf{D}$ by including $\mathbf{d}^+$ and four randomly selected unclicked documents, denoted by $\mathbf{d}_j^-; j = 1, \ldots, 4$. In our pilot study, we do not observe any significant difference when different sampling strategies were used to select the unclicked documents.

In training, the model parameters are estimated to maximize the likelihood of the clicked documents given the queries across the training set. Equivalently, we need to minimize the following loss function

$$L(\mathbf{\Lambda}) = -\log \Pi_{(\mathbf{q},\mathbf{d}^+)} P(\mathbf{d}^+|\mathbf{q}) \tag{4.6}$$

where $\mathbf{\Lambda}$ denotes the parameter set of the neural networks $\{\mathbf{W}_i, \mathbf{b}_i\}$. Since $L(\mathbf{\Lambda})$ is differentiable w.r.t. to $\mathbf{\Lambda}$, the model is trained readily using gradient-based numerical optimization algorithms. The update rule is

$$\mathbf{\Lambda}_t = \mathbf{\Lambda}_{t-1} - \epsilon_t \frac{\partial L(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}}\bigg|_{\mathbf{\Lambda}=\mathbf{\Lambda}_{t-1}} \tag{4.7}$$

where ϵ_t is the learning rate at the t-th iteration, $\mathbf{\Lambda}_t$ and $\mathbf{\Lambda}_{t-1}$ are the models at the t-th and the $(t-1)$-th iteration, respectively.

In what follows, we derive the gradient of the loss function w.r.t. the parameters of the neural networks. Assuming that there are in total R (query, clicked-document) pairs, we denote $(\mathbf{q}_r, \mathbf{d}_r^+)$ as the r-th (query, clicked-document) pair. Then, if we denote

$$L_r(\mathbf{\Lambda}) = -\log P(\mathbf{d}_r^+|\mathbf{q}_r) \tag{4.8}$$

we have

$$\frac{\partial L(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = \sum_{r=1}^{R} \frac{\partial L_r(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} \tag{4.9}$$

In the following, we will show the derivation of $\frac{\partial L_r(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}}$. For a query $\mathbf{q}$ and a document $\mathbf{d}$, we denote $\mathbf{l}_{i,q}$ and $\mathbf{l}_{i,d}$ be the activation in the hidden layer i, and $\mathbf{y}_q$ and $\mathbf{y}_d$ be the output activation for $\mathbf{q}$ and $\mathbf{d}$, respectively. They are computed according to Eq. (4.2). We then derive $\frac{\partial L_r(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}}$ as follows.[1] For

[1]We present only the derivation for the weight matrices. The derivation for the bias vector is similar and is omitted.

simplification, the subscript of r will be omitted hereafter. First, the loss function in Eq. (4.8) can be written as:

$$L(\mathbf{\Lambda}) = \log(1 + \sum_j \exp(-\gamma \Delta_j)) \tag{4.10}$$

where $\Delta_j = R(\mathbf{q}, \mathbf{d}^+) - R(\mathbf{q}, \mathbf{d}_j^-)$. The gradient of the loss function w.r.t. the N-th weight matrix $\mathbf{W}_N$ is

$$\frac{\partial L(\mathbf{\Lambda})}{\partial \mathbf{W}_N} = \sum_j \alpha_j \frac{\partial \Delta_j}{\partial \mathbf{W}_N} \tag{4.11}$$

where

$$\frac{\partial \Delta_j}{\partial \mathbf{W}_N} = \frac{\partial R(\mathbf{q}, \mathbf{d}^+)}{\partial \mathbf{W}_N} - \frac{\partial R(\mathbf{q}, \mathbf{d}_j^-)}{\partial \mathbf{W}_N} \tag{4.12}$$

and

$$\alpha_j = \frac{-\gamma \exp(-\gamma \Delta_j)}{1 + \sum_{j'} \exp(-\gamma \Delta_{j'})} \tag{4.13}$$

To simplify the notation, let a, b, c be $\mathbf{y}_q^{\mathrm{T}} \mathbf{y}_d$, $1/\|\mathbf{y}_q\|$, and $1/\|\mathbf{y}_d\|$, respectively. With *tanh* as the activation function in our model, each term in the right-hand side of Eq. (4.12) can be calculated using the following formula:

$$\frac{\partial R(\mathbf{q}, \mathbf{d})}{\partial \mathbf{W}_N} = \frac{\partial}{\partial \mathbf{W}_N} \frac{\mathbf{y}_q^{\mathrm{T}} \mathbf{y}_d}{\|\mathbf{y}_q\| \|\mathbf{y}_d\|} = \boldsymbol{\delta}_{y_q}^{(q,d)} \mathbf{l}_{N-1,q}^{\mathrm{T}} + \boldsymbol{\delta}_{y_d}^{(q,d)} \mathbf{l}_{N-1,d}^{\mathrm{T}} \tag{4.14}$$

where $\boldsymbol{\delta}_{y_q}^{(q,d)}$ and $\boldsymbol{\delta}_{y_d}^{(q,d)}$ for a pair of $(\mathbf{q}, \mathbf{d})$ are computed as

$$\begin{aligned} \boldsymbol{\delta}_{y_q}^{(q,d)} &= (\mathbf{1} - \mathbf{y}_q) \odot (\mathbf{1} + \mathbf{y}_q) \odot (bc\mathbf{y}_d - acb^3 \mathbf{y}_q) \\ \boldsymbol{\delta}_{y_d}^{(q,d)} &= (\mathbf{1} - \mathbf{y}_d) \odot (\mathbf{1} + \mathbf{y}_d) \odot (bc\mathbf{y}_q - abc^3 \mathbf{y}_d) \end{aligned} \tag{4.15}$$

where the operator $\odot$ is the element-wise multiplication (Hadamard product). For hidden layers, we also need to calculate $\boldsymbol{\delta}$ for each Δ_j. For example, each $\boldsymbol{\delta}$ in the hidden layer i can be calculated through back-propagation as

$$\begin{aligned} \boldsymbol{\delta}_{i,q}^{(q,d)} &= (\mathbf{1} + \mathbf{l}_{i,q}) \odot (\mathbf{1} - \mathbf{l}_{i,q}) \odot \mathbf{W}_{i+1}^{\mathrm{T}} \boldsymbol{\delta}_{i+1,q}^{(q,d)} \\ \boldsymbol{\delta}_{i,d}^{(q,d)} &= (\mathbf{1} + \mathbf{l}_{i,d}) \odot (\mathbf{1} - \mathbf{l}_{i,d}) \odot \mathbf{W}_{i+1}^{\mathrm{T}} \boldsymbol{\delta}_{i+1,d}^{(q,d)} \end{aligned} \tag{4.16}$$

and eventually we have $\boldsymbol{\delta}_{N,q}^{(q,d)} = \boldsymbol{\delta}_{y_q}^{(q,d)}$ and $\boldsymbol{\delta}_{N,d}^{(q,d)} = \boldsymbol{\delta}_{y_d}^{(q,d)}$.

Correspondingly, the gradient of the loss function w.r.t. the intermediate

weight matrix, $\mathbf{W}_i$, $i = 2, \ldots, N-1$, can be computed as[2]

$$\frac{\partial L(\Lambda)}{\partial \mathbf{W}_i} = \sum_j \alpha_j \frac{\partial \Delta_j}{\partial \mathbf{W}_i} \tag{4.17}$$

where

$$\frac{\partial \Delta_j}{\partial \mathbf{W}_i} = \left(\boldsymbol{\delta}_{i,q}^{(q,d^+)} \mathbf{l}_{i-1,q}^{\mathrm{T}} + \boldsymbol{\delta}_{i,d^+}^{(q,d^+)} \mathbf{l}_{i-1,d^+}^{\mathrm{T}}\right) - \left(\boldsymbol{\delta}_{i,q}^{(q,d_j^-)} \mathbf{l}_{i-1,q}^{\mathrm{T}} + \boldsymbol{\delta}_{i,d_j^-}^{(q,d_j^-)} \mathbf{l}_{i-1,d_j^-}^{\mathrm{T}}\right) \tag{4.18}$$

4.3.4 Implementation Details

To determine the training parameters and to avoid over-fitting, we divided the clickthrough data into two sets that do not overlap, called training and validation datasets, respectively. In our experiments, the models are trained on the training set and the training parameters are optimized on the validation dataset. For the DNN experiments, we used the architecture with three hidden layers as shown in Figure 4.1. The first hidden layer is the word hashing layer containing about 30 K nodes (e.g., the size of the letter trigrams as shown in Table 4.1). The next two hidden layers have 300 hidden nodes each, and the output layer has 128 nodes. Word hashing is based on a fixed projection matrix. The similarity measure is based on the output layer with the dimensionality of 128. Following [87], we initialize the network weights with uniform distribution in the range between $-\sqrt{6/(\text{fanin} + \text{fanout})}$ and $\sqrt{6/(\text{fanin} + \text{fanout})}$ where fanin and fanout are the number of input and output units, respectively. Empirically, we have not observed better performance by doing layer-wise pre-training. In the training stage, we optimize the model using mini-batch based stochastic gradient descent (SGD). Each mini-batch consists of 1024 training samples. We observed that the DNN training usually converges within 20 epochs (passes) over the entire training data.

[2]Note that $\mathbf{W}_1$ is the matrix of word hashing. It is fixed and needs no training.

4.4 Experiments

We evaluated the DSSM, proposed in Section 4.3, on the web document ranking task using a real-world dataset. In this section, we first describe the dataset on which the models are evaluated. Then, we compare the performances of our best model against other state-of-the-art ranking models. We also investigate the break-down impact of the techniques proposed in Section 4.3.

4.4.1 Datasets and Evaluation Methodology

We have evaluated the retrieval models on a large-scale real world dataset, called the evaluation dataset henceforth. The evaluation dataset contains 16,510 English queries sampled from one-year query log files of a commercial search engine. On average, each query is associated with 15 web documents (URLs). Each query-title pair has a relevance label. The label is human generated and is on a 5-level relevance scale, 0 to 4, where level 4 means that the document is the most relevant to query and 0 means the document is not relevant to the corresponding query. All the queries and documents are preprocessed such that the text is white-space tokenized and lowercased, numbers are retained, and no stemming/inflection is performed.

All ranking models used in this study (i.e., DSSM, topic models, and linear projection models) contain many free hyperparameters that must be estimated empirically. In all experiments, we have used a twofold cross validation: A set of results on one-half of the data is obtained using the parameter settings optimized on the other half, and the global retrieval results are combined from the two sets.

The performance of all ranking models we have evaluated has been measured by mean Normalized Discounted Cumulative Gain (NDCG) [88], and we will report NDCG scores at truncation levels 1, 3, and 10 in this section. We have also performed a significance test using the paired t-test. Differences are considered statistically significant when the p-value is less than 0.05.

In our experiments, we assume that a query is parallel to the titles of the documents clicked on for that query. We extracted large amounts of the query-title pairs for model training from one-year query log files using a procedure similar to [89]. Some previous studies, e.g., [89,90], showed that the

Table 4.2: Comparative results with the previous state-of-the-art approaches and various settings of DSSM.

#	Models	NDCG@1	NDCG@3	NDCG@10
1	TF-IDF	0.319	0.382	0.462
2	BM25	0.308	0.373	0.455
3	WTM	0.332	0.400	0.478
4	LSA	0.298	0.372	0.455
5	PLSA	0.295	0.371	0.456
6	DAE	0.310	0.377	0.459
7	BLTM-PR	0.337	0.403	0.480
8	DPM	0.329	0.401	0.479
9	DNN	0.342	0.410	0.486
10	L-WH linear	0.357	0.422	0.495
11	L-WH nonlinear	0.357	0.421	0.494
12	**L-WH DNN**	**0.362**	**0.425**	**0.498**

query click field, when it is valid, is the most effective piece of information for web search, seconded by the title field. However, click information is unavailable for many URLs, especially new URLs and tail URLs, leaving their click fields invalid (i.e., the field is either empty or unreliable because of sparseness). In this chapter, we assume that each document contained in the evaluation dataset is either a new URL or a tail URL, thus has no click information (i.e., its click field is invalid). Our research goal is to investigate how to learn the latent semantic models from the popular URLs that have rich click information, and apply the models to improve the retrieval of those tail or new URLs. To this end, in our experiments only the title fields of the web documents are used for ranking. For training latent semantic models, we use a randomly sampled subset of approximately 100 million pairs whose documents are popular and have rich click information. We then test trained models in ranking the documents in the evaluation dataset containing no click information. The query-title pairs are pre-processed in the same way as the evaluation data to ensure uniformity.

4.4.2 Results

The main results of our experiments are summarized in Table 4.2, where we compared our best version of the DSSM (Row 12) with three sets of baseline

models. The first set of baselines includes a couple of widely used lexical matching methods such as TF-IDF (Row 1) and BM25 (Row 2). The second is a word translation model (WTM in Row 3) which is intended to directly address the query-document language discrepancy problem by learning a lexical mapping between query words and document words [75,76]. The third includes a set of state-of-the-art latent semantic models which are learned either on documents only in an unsupervised manner (LSA, PLSA, DAE as in Rows 4 to 6) or on clickthrough data in a supervised way (BLTM-PR, DPM, as in Rows 7 and 8). In order to make the results comparable, we re-implement these models following the descriptions in [76], e.g., models of LSA and DPM are trained using a 40K-word vocabulary due to the model complexity constraint, and the other models are trained using a 500K-word vocabulary. Details are elaborated in the following paragraphs.

TF-IDF (Row 1) is the baseline model, where both documents and queries are represented as term vectors with TF-IDF term weighting. The documents are ranked by the cosine similarity between the query and document vectors. We also use BM25 (Row 2) ranking model as one of our baselines. Both TF-IDF and BM25 are state-of-the-art document ranking models based on term matching. They have been widely used as baselines in related studies.

WTM (Rows 3) is our implementation of the word translation model described in [75], listed here for comparison. We see that WTM outperforms both baselines (TF-IDF and BM25) significantly, confirming the conclusion reached in [75]. LSA (Row 4) is our implementation of latent semantic analysis model. We used PCA instead of SVD to compute the linear projection matrix. Queries and titles are treated as separate documents, the pair information from the clickthrough data was not used in this model. **PLSA** (Rows 5) is our implementation of the model proposed in [40], and was trained on documents only (i.e., the title side of the query-title pairs). Different from [40], our version of PLSA was learned using MAP estimation as in [76]. **DAE** (Row 6) is our implementation of the deep auto-encoder based semantic hashing model proposed by Salakhutdinov and Hinton in [79]. Due to the model training complexity, the input term vector is based on a 40K-word vocabulary. The DAE architecture contains four hidden layers, each of which has 300 nodes, and a bottleneck layer in the middle which has 128 nodes. The model is trained on documents only in an unsupervised manner. In the fine-tuning stage, we used cross-entropy error as the training objective.

The central layer activations are used as features for the computation of cosine similarity between query and document. Our results are consistent with previous results reported in [79]. The DNN-based latent semantic model outperforms the linear projection model (e.g., LSA). However, both LSA and DAE are trained in an unsupervised fashion on document collection only, thus cannot outperform the state-of-the-art lexical matching ranking models.

BLTM-PR (Rows 7) is the best performer among different versions of the bilingual topic models described in [76]. BLTM with posterior regularization (BLTM-PR) is trained on query-title pairs using the EM algorithm with a constraint enforcing the paired query and title to have same fractions of terms assigned to each hidden topic. **DPM** (Row 8) is the linear discriminative projection model proposed in [76], where the projection matrix is discriminatively learned using the S2Net algorithm [77] on relevant and irrelevant pairs of queries and titles. Similar to that BLTM is an extension to PLSA, DPM can also be viewed as an extension of LSA, where the linear projection matrix is learned in a supervised manner using clickthrough data, optimized for document ranking. We see that using clickthrough data for model training leads to some significant improvement. Both BLTM-PR and DPM outperform the baseline models (TF-IDF and BM25).

Rows 9 to 12 present results of different settings of the DSSM. **DNN** (Row 9) is a DSSM without using word hashing. It uses the same structure as DAE (Row 6), but is trained in a supervised fashion on the clickthrough data. The input term vector is based on a 40K-word vocabulary, as used by DAE. **L-WH linear** (Row 10) is the model built using letter-trigram-based word hashing and supervised training. It differs from the **L-WH nonlinear model** (Row 11) in that we do not apply any nonlinear activation function, such as tanh, to its output layer. **L-WH DNN** (Row 12) is our best DNN-based semantic model, which uses three hidden layers, including the layer with the Letter-trigram-based Word Hashing (L-WH), and an output layer, and is discriminatively trained on query-title pairs, as described in Section 4.3. Although the letter n-gram-based word hashing method can be applied to arbitrarily large vocabularies, in order to perform a fair comparison with other competing methods, the model uses a 500K-word vocabulary.

The results in Table 4.2 show that the deep structured semantic model is the best performer, beating other methods by a statistically significant margin in NDCG and demonstrating the empirical effectiveness of using DNNs for

semantic matching. From the results in Table 4.2, it is also clear that supervised learning on clickthrough data, coupled with an IR-centric optimization criterion tailoring to ranking, is essential for obtaining superior document ranking performance. For example, both DNN and DAE (Row 9 and 6) use the same 40K-word vocabulary and adopt the same deep architecture. The former outperforms the latter by 3.2 points in NDCG@1.

Word hashing allows us to use very large vocabularies for modeling. For instance, the models in Rows 12, which use a 500K-word vocabulary (with word hashing), significantly outperform the model in Row 9, which uses a 40K-word vocabulary, although the former has slightly fewer free parameters than the later since the word hashing layer containing about only 30 K nodes.

We also evaluated the impact of using a deep architecture versus a shallow one in modeling semantic information embedded in a query and a document. Results in Table 4.2 show that DAE (Row 3) is better than LSA (Row 2), while both LSA and DAE are unsupervised models. We also have observed similar results when comparing the shallow vs. deep architecture in the case of supervised models. Comparing models in Rows 11 and 12 respectively, we observe that increasing the number of nonlinear layers from one to three raises the NDCG scores by 0.4-0.5 point which are statistically significant, while there is no significant difference between linear and nonlinear models if both are one-layer shallow models (Row 10 vs. Row 11).

4.4.3 Analysis on Document Ranking Errors

In the test data, among 16,412 unique queries, we compare each query's NDCG@1 values using TF-IDF and our best model, letter-trigram-based word hashing with supervised DNN (L-WH DNN). There are in total 1,985 queries on which L-WH DNN performs better than TF-IDF (the sum of NDCG@1 differences is 1332.3). On the other hand, TF-IDF outperforms L-WH DNN on 1077 queries (the sum of NDCG@1 differences is 630.61). For both cases, we sample several concrete examples. They are shown in Tables 4.3 and 4.4, respectively. We observe in Table 4.3 that the NDCG improvement is largely to the better match between queries and titles in the semantic level than in the lexical level.

To make our method more intuitive, we have also visualized the learned

Table 4.3: Examples showing our deep semantic model performs better than TF-IDF.

L-WH DNN wins over TF-IDF		
	Query	Title
1	bfpo	postcodes in the united kingdom wikipedia the free encyclopedia
2	univ of penn	university of pennsylvania wikipedia the free encyclopedia
3	citibank	citi com
4	ccra	canada revenue agency website
5	search galleries	photography community including forums reviews and galleries from photo net
6	met art	metropolitan museum of art wikipedia the free encyclopedia
7	new york brides	long island bride and groom wedding magazine website
8	motocycle loans	auto financing is easy with the capital one blank check
9	boat	new and used yarts for sale yachartworld com
10	bbc games	bbc sport

Table 4.4: Examples showing our deep semantic model performs worse than TF-IDF.

L-WH DNN loses to TF-IDF		
	Query	Title
1	hey arnold	hey arnold the movie
2	internet by dell	dell hyperconnect mobile internet solutions dell
3	www mcdonalds com	mcdonald s
4	m t	m t bank
5	board of directors	board of directors west s encyclopedia of american law full article from answers com
6	puppet skits	skits
7	montreal canada attractions	go montreal tourist information
8	how to address a cover letter	how to write a cover letter
9	bbc television	bbc academy
10	rock com	rock music information from answers com

hidden representations of the words in the queries and documents. We do so by treating each word as a unique document and passing it as an input to the trained DNN. At each output node, we group all the words with high activation levels and cluster them accordingly. Table 4.5 shows some

Table 4.5: Examples of the clustered words on five different output nodes of the trained DNN. The clustering criterion is high activation levels at the output nodes of the DNN.

automotive	chevrolet	youtube	bear	systems
wheels	fuel	videos	hunting	protect
cars	motorcycle	dvd	texas	platform
auto	toyota	downloads	colorado	efficiency
car	chevy	movie	hunter	oems
vehicle	motorcycles	cd	tucson	systems32

example clusters, each corresponding to an output node of the DNN model. It is interesting to see that words with the same or related semantic meanings do stay in the same cluster.

4.5 Summary

We presented and evaluated a series of new latent semantic models, notably those with deep architectures which we call the DSSM. The main contribution lies in our significant extension of the previous latent semantic models (e.g., LSA) in three key aspects. First, we make use of the clickthrough data to optimize the parameters of all versions of the models by directly targeting the goal of document ranking. Second, inspired by the deep learning framework recently shown to be highly successful in speech recognition [17,82,83,91,92], we extend the linear semantic models to their nonlinear counterparts using multiple hidden-representation layers. The deep architectures adopted have further enhanced the modeling capacity so that more sophisticated semantic structures in queries and documents can be captured and represented. Third, we use a letter n-gram-based word hashing technique that proves instrumental in scaling up the training of the deep models so that very large vocabularies can be used in realistic web search. In our experiments, we show that the new techniques pertaining to each of the above three aspects lead to significant performance improvement on the document ranking task. A combination of all three sets of new techniques has led to a new state-of-the-art semantic model that beats all the previously developed competing models with a significant margin.

Chapter 5

Joint Shallow and Deep Learning: Random Features for Kernel Deep Convex Networks

5.1 Introduction

Deep learning has achieved many state-of-the-art results in speech and language processing in recent years [17, 91]. By exploiting deep architectures, deep learning techniques are able to learn different levels of abstraction and further discriminate among data. While deep learning techniques such as deep neural networks have shown remarkable results in recognition and classification tasks, training deep learning models has proved to be computationally difficult [17, 91].

Another architecture, Deep Convex Network (DCN), was proposed to address the scalability issue [93, 94]. Instead of using a large number of hidden units in DCN, Kernel Deep Convex Network (K-DCN) was further proposed to use a kernel trick so that the number of hidden units in each DCN layer is unbounded [95]. However, K-DCN with a Gaussian kernel suffers some limitations in memory and computation, when there are a large number of training and testing samples.

In this chapter, we propose a method for efficiently training and testing K-DCN using the Bochner Fourier-space sampling approximation of a Gaussian kernel in each of the K-DCN modules. By projecting original features to a higher-dimensional space explicitly, we can achieve similar performance as K-DCN but with better computational efficiency in time and memory. We demonstrate the effectiveness of our approach on ATIS slot filling and TIMIT phone classification tasks.

The remainder of this chapter is organized as follows: Section 5.2 introduces the kernel deep convex network (K-DCN). Section 5.3 discusses the limitation of the original K-DCN, and possible solutions in the literature which only addressed the computation and memory problem in training but

not in evaluation. Section 5.4 presents random features and the application to solve the computation and memory problem in both training and evaluation for each of the K-DCN modules. The experimental setups, along with results, are described in Section 5.5.

5.2 Kernel Deep Convex Network

Kernel Deep Convex Network (K-DCN) was proposed in [95], which is a kernel version of the deep convex network (DCN) [93, 94]. In the DCN or K-DCN framework, the outputs from all previous modules are concatenated with the original input data as a new input for the current module. Figure 5.1 shows an example of a three-layer K-DCN architecture.

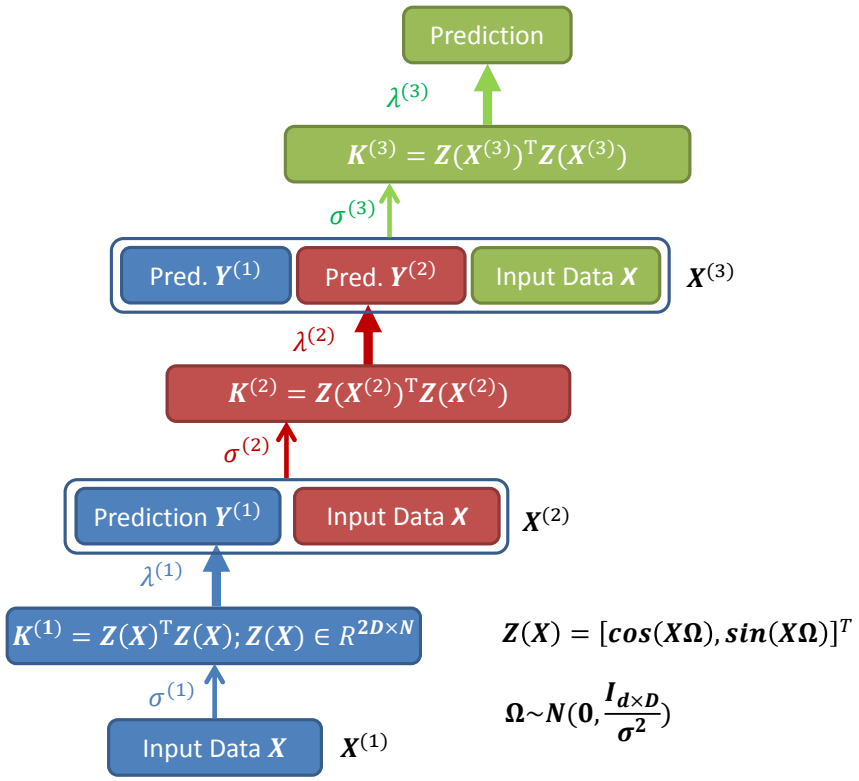


Figure 5.1: Architecture of a three-layer K-DCN with random Fourier features, where $\mathbf{\Omega}$ is a random matrix with values sampled from $\mathcal{N}(0, \mathbf{I}_{d \times D}/\sigma^2)$, $\mathbf{Z}(\mathbf{X})$ is a random projection of input $\mathbf{X}$, and parameters σ and λ are the standard deviation for the Gaussian random variable and the regularization parameter for kernel ridge regression, respectively.

The single module of a K-DCN is a kernel ridge regression, which can be expressed as:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = \mathbf{k}(\mathbf{x})^{\mathrm{T}} \alpha \tag{5.1}$$

where a sample $\mathbf{x}$ is evaluated with respect to all the training samples $\{\mathbf{x}_i\}_{i=1}^{N}$, vector $\mathbf{k}(\mathbf{x})$ is with element $k_n(\mathbf{x}) = k(\mathbf{x_n}, \mathbf{x})$, and α is the regression coefficient. Using the training data, the kernel ridge regression coefficient has a closed-form solution of the following form:

$$\alpha = (\lambda \mathbf{I} + \mathbf{K})^{-1} \mathbf{Y} \tag{5.2}$$

where λ is the regularization parameter, $\mathbf{K} \in \mathbb{R}^{N \times N}$ is a kernel matrix with elements $\mathbf{K}_{mn} = k(\mathbf{x}_m, \mathbf{x}_n)$, $\{\mathbf{x}_i\}_{i=1}^{N}$ are from the training set, and $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^{\mathrm{T}} \in \mathbb{R}^{N \times M}$ are the label vectors for training, where M is the number of classes and $\mathbf{y}_i$, $1 \leq i \leq N$, is a vector of size M with the only nonzero entry one as the class label [96].

5.3 Large-Scale Kernel Machines

5.3.1 Limitation in Large-Scale Cases

In the kernel regression framework, solving Eq. (5.2) suffers difficulty when there is a large number of training data. Suppose there are N training samples. It takes $O(N^2)$ storage for the kernel matrix and $O(N^3)$ time for computing the matrix inversion.

In the evaluation stage, as shown in Eq. (5.1), each sample is evaluated with respect to all the training samples. Suppose $\mathbf{x}, \{\mathbf{x}_i\}_{i=1}^{N} \in \mathbb{R}^d$. To evaluate a sample, it takes $O(Nd)$ operations to compute $k(\mathbf{x}, \mathbf{x}_i)$ with each training vector $\mathbf{x}_i$, and the training vectors must be retained in memory. For a large training dataset, these testing costs are significant.

5.3.2 Related Work

To handle the limitation in large-scale data cases, several approaches have been proposed. The approaches can be categorized into two categories: (i)

selecting a subset of training data and (ii) approximating the kernel matrix. (i) Burges and Schölkopf proposed a reduced set method by finding a smaller set of vectors approximating the support vector decision surface and thereby improve classification speed [97]. Baudat and Anouar further proposed a feature vector selection method by finding a subset $\mathcal{S}$ of the training data, forming a basis in the feature space, where the cardinality of the subset $|\mathcal{S}| \ll N$. Equation (5.1) can be approximated as:

$$f(\mathbf{x}) \approx \sum_{i \in \mathcal{S}} \beta_i \phi(\mathbf{x}) \phi(\mathbf{x}_i) = \sum_{i \in \mathcal{S}} \beta_i k(\mathbf{x}, \mathbf{x}_i) \tag{5.3}$$

Although computation and storage are lower with a smaller set of training samples, this approach requires $O(NL^2)$ computation cost to find a subset of size L sampled among N vectors. Moreover, selecting a subset of training data loses information by throwing away training samples, and is inefficient in finding a good subset among a large-scale dataset [98].
(ii) To reduce the storage and computation cost on a kernel matrix, the kernel matrix can be approximated by throwing away individual entries [99], by throwing away entire columns [100, 101], or by a linear kernel [7]. Nyström methods create a subset by sampling columns of the original kernel matrix [100–102]. With Nyström Woodbury approximation, Eq. (5.2) can be computed efficiently, as shown in Eq. (5.4).

$$\begin{aligned} &(\lambda \mathbf{I} + \mathbf{K})^{-1} \mathbf{Y} \\ &\approx (\lambda \mathbf{I} + \mathbf{C} \mathbf{W}_{\mathbf{k}}^{+} \mathbf{C}^{\mathrm{T}})^{-1} \mathbf{Y} \\ &= \frac{1}{\lambda} \Big(\mathbf{Y} - \mathbf{C} [\lambda \mathbf{I}_{\mathbf{k}} + \mathbf{W}_{\mathbf{k}}^{+} \mathbf{C}^{\mathrm{T}} \mathbf{C}]^{-1} \mathbf{W}_{\mathbf{k}}^{+} \mathbf{C}^{\mathrm{T}} \mathbf{Y} \Big) \end{aligned} \tag{5.4}$$

where $\mathbf{C} \in \mathbb{R}^{m \times N}$ is formed by selecting m columns of $\mathbf{K}$, $\mathbf{W}$ is the intersection of these m columns with the corresponding m rows of $\mathbf{K}$, and $\mathbf{W_k}^{+}$ is the pseudo inverse of the k-rank approximation of $\mathbf{W}$. However, in the evaluation stage, the limitation mentioned in Section 5.3.1 still exists. A linear random projection method was proposed in [99] to speed up kernel matrix evaluation, but it still takes $O(N \log d)$ to evaluate a sample. The random features method approach maps input data to a random feature space and approximates a kernel function by a linear kernel [7]. The detailed theorem and application to K-DCN will be presented in Section 5.4.

5.4 Random Features

The kernel trick of a kernel function $k(\mathbf{x}, \mathbf{y})$, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ is to compute an inner product between implicitly mapped features in a high-dimensional space, i.e., $k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$. Instead of using the implicit feature mapping in the kernel trick, Rahimi and Recht proposed a random feature method for approximating kernel evaluation [7]. The idea is to explicitly map the data to a Euclidean inner product space using a randomized feature map $\mathbf{z} : \mathbb{R}^d \to \mathbb{R}^D$ such that the kernel evaluation can be approximated by the inner product between the transformed pair:

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \approx \mathbf{z}(\mathbf{x})^{\mathrm{T}} \mathbf{z}(\mathbf{y}) \tag{5.5}$$

Theorem 1. (Bochner's theorem [103]) *A kernel $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} - \mathbf{y})$ on $\mathbb{R}^d$ is positive definite if and only if $k(\delta)$ is the Fourier transform of a non-negative measure.*

Bochner's theorem guarantees that the Fourier transform of a shift-invariant and properly scaled kernel is a probability density. Defining $z_\omega(\mathbf{x}) = e^{j\omega^{\mathrm{T}}\mathbf{x}}$, we get

$$k(\mathbf{x} - \mathbf{y}) = \int_{\mathbb{R}^d} p(\omega) e^{-j\omega^{\mathrm{T}}(\mathbf{x}-\mathbf{y})} d\omega = \mathbb{E}_\omega[z_\omega(\mathbf{x}) z_\omega(\mathbf{y})^*] \tag{5.6}$$

where $z_\omega(\mathbf{x}) z_\omega(\mathbf{y})^*$ is an unbiased estimate of $k(\mathbf{x}, \mathbf{y})$ when ω is sampled from $p(\omega)$. For example, $p(\omega) = (2\pi)^{-\frac{D}{2}} e^{-\frac{\|\omega\|_2^2}{2}}$ is the Fourier transform of a Gaussian kernel $k(\Delta) = e^{-\frac{\|\Delta\|_2^2}{2}}$.

To reduce the variance of the estimate, we can concatenate D randomly chosen z_ω into a column vector $\mathbf{z}$ and normalize each component by $\sqrt{D}$. Therefore, the inner product $\mathbf{z}(\mathbf{x})^{\mathrm{T}} \mathbf{z}(\mathbf{y}) = \frac{1}{D} \sum_{j=1}^{D} z_{\omega_j}(\mathbf{x}) z_{\omega_j}(\mathbf{y})^*$ is a lower variance approximation to the kernel function $k(\mathbf{x}, \mathbf{y})$. The approximation converges to the kernel function exponentially fast in D [7]. The quality of the approximation is determined by the mapping dimension D [104].

To obtain real-valued random features, we can replace $z_\omega(\mathbf{x})$ by the mapping $z_\omega(\mathbf{x}) = [\cos(\omega^{\mathrm{T}}\mathbf{x}), \sin(\omega^{\mathrm{T}}\mathbf{x})]$, which also satisfies the condition

$$\mathbb{E}_\omega[z_\omega(\mathbf{x}) z_\omega(\mathbf{y})^*] = k(\mathbf{x}, \mathbf{y}).$$

The vector

$$\mathbf{z}(\mathbf{x}) = \frac{1}{\sqrt{D}}[\cos(\omega_1^{\mathrm{T}}\mathbf{x}), \ldots, \cos(\omega_D^{\mathrm{T}}\mathbf{x}), \sin(\omega_1^{\mathrm{T}}\mathbf{x}), \ldots, \sin(\omega_D^{\mathrm{T}}\mathbf{x})]^{\mathrm{T}}$$

is a $2D$-dimensional *random Fourier feature* of the input feature $\mathbf{x}$, where $\omega_1, \ldots, \omega_D \in \mathbb{R}^d$ are drawn from $p(\omega)$. For a Gaussian kernel, ω_i, $i = 1, \ldots, D$, are drawn from a normal distribution $\mathcal{N}(0, \mathbf{I_d}/\sigma^2)$.

5.4.1 Random Features for Kernel Ridge Regression

By using the random Fourier feature approximation, we can resolve the limitation mentioned in Section 5.3.1. In the training stage, first, define $\mathbf{X} \in \mathbb{R}^{N \times d}$ with row i equal to $\mathbf{x}_i$, and $\mathbf{Z}(\mathbf{X}) = [\cos(\mathbf{X\Omega}), \sin(\mathbf{X\Omega})]^{\mathrm{T}} \in \mathbb{R}^{2D \times N}$ with column i equal to $\mathbf{z}(\mathbf{x}_i)$, where $\mathbf{\Omega}$ is a random matrix with values sampled from $\mathcal{N}(0, \mathbf{I}_{d \times D}/\sigma^2)$. The kernel matrix $\mathbf{K}$ can be approximated by $\mathbf{Z}(\mathbf{X})^{\mathrm{T}}\mathbf{Z}(\mathbf{X}) \in \mathbb{R}^{N \times N}$. We can write Eq. (5.1) in the matrix form as follows:

$$\mathbf{Y} = \mathbf{K}\alpha = \mathbf{Z}(\mathbf{X})^{\mathrm{T}}\mathbf{Z}(\mathbf{X})\alpha \tag{5.7}$$

Instead of computing and storing $\mathbf{Z}(\mathbf{X})^{\mathrm{T}}\mathbf{Z}(\mathbf{X})$ in the memory, we can use a trick to first compute the Eq. (5.8):

$$\mathbf{Z}(\mathbf{X})\mathbf{Y} = \mathbf{Z}(\mathbf{X})\mathbf{Z}(\mathbf{X})^{\mathrm{T}}\mathbf{Z}(\mathbf{X})\alpha \tag{5.8}$$

Then, defining $\mathbf{W} = \mathbf{Z}(\mathbf{X})\alpha$, we can solve for $\mathbf{W}$ as follows:

$$\mathbf{W} = (\lambda\mathbf{I} + \mathbf{Z}(\mathbf{X})\mathbf{Z}(\mathbf{X})^{\mathrm{T}})^{-1}\mathbf{Z}(\mathbf{X})\mathbf{Y} \tag{5.9}$$

In this case, we compute and store the matrix $\mathbf{Z}(\mathbf{X})\mathbf{Z}(\mathbf{X})^{\mathrm{T}} \in \mathbb{R}^{2D \times 2D}$, $\mathbf{Z}(\mathbf{X})\mathbf{Y} \in \mathbb{R}^{2D \times M}$ only.
In the evaluation stage, Eq. (5.1) can be written as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}, \mathbf{x_i}) = \sum_{i=1}^{N} \alpha_i \mathbf{z}(\mathbf{x_i})^{\mathrm{T}}\mathbf{z}(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{z}(\mathbf{x}) \tag{5.10}$$

where $\mathbf{w}^{\mathrm{T}} = \sum_{i=1}^{N} \alpha_i \mathbf{z}(\mathbf{x_i})^{\mathrm{T}}$ is solved in the training stage. The function $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{z}(\mathbf{x})$ requires only $O(D + d)$ operations and storage. Table 5.1

Table 5.1: Comparison between kernel ridge regression and random feature kernel regression using a Gaussian kernel.

	Kernel Ridge Regression	Random Feature Kernel Regression
Kernel	$\mathbf{K}_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/2\sigma^2)$	$\mathbf{K}_{ij} = \mathbf{z}(\mathbf{x}_i)^{\mathrm{T}}\mathbf{z}(\mathbf{x}_j)$
Training Formula	$\alpha = (\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{Y}$	$\mathbf{W} = (\lambda\mathbf{I} + \mathbf{Z}(\mathbf{X})\mathbf{Z}(\mathbf{X})^{\mathrm{T}})^{-1}\mathbf{Z}(\mathbf{X})\mathbf{Y}$
Evaluation Formula	$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}_i)$	$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{z}(\mathbf{x})$

summarizes the comparison between kernel ridge regression and random feature kernel regression using a Gaussian kernel.

5.5 Experiments

In this section, we examine the effectiveness of K-DCN with random Fourier features, denoted as *K-DCNRF*, on ATIS slot-filling and TIMIT phone classification tasks.

5.5.1 ATIS Experiments

We conduct slot-filling experiments on the ATIS dataset following similar settings as described in [105]. In the task, each word will be tagged by a slot ID, and there is a total of 127 slot IDs. An example of a sentence and its slot ID sequence is shown in Table 5.2, following the in/out/begin (IOB) representation, e.g., Boston and New York are the departure and arrival cities and today is the date.

Table 5.2: An example of a sentence and its slot ID sequence.

sentence	show	flights	from	Boston	to	New	York	Today
Slot ID	O	O	O	B-dept	O	B-arr	l-arr	B-date

The training set consists of 4978 sentences and the test set consists of 893 sentences. In the experiment, we project each word to a 50-dimensional dense vector by looking-up a embedding mapping table, which is trained through

unsupervised learning on Wikipedia text corpus [80]. Then we concatenate the embedding vectors in a context window to form a contextual vector as the input for K-DCN/K-DCNRF. As in classical classification tasks, the output of K-DCN/K-DCNRF is a 127-dimensional vector, each element corresponds to one slot ID. K-DCN/K-DCNRF is then trained on the training set. The slot ID prediction error rate of K-DCN/K-DCNRF with various window sizes are reported in Table 5.3. In contrast, a logistic regression baseline that uses n-gram features (n = 1~5) derived from a five-word window obtains a slot error rate of 4.38%. We also present a strong baseline using linear CRF. The result is obtained using only lexical features, with the default parameters of the CRF++ toolkit, following [106]. This gives a slot ID error rate of 3.42%. In Table 5.3, we show that, by projecting original features to a high-dimensional space explicitly, K-DCNRF can achieve similar (or even slightly better) performance to K-DCN. Moreover, in the five-word window case, K-DCNRF outperforms two baseline results. Figure 5.2 shows the experimental results on the training and testing set at different layers.

Table 5.3: ATIS results with different window sizes.

Window	Method	Layer (proj. dim)	Test Error (%)
1	K-DCNRF	36 (2000)	12.77
3	K-DCN	10	3.83
3	K-DCNRF	34 (16000)	**3.81**
5	logistic regression	-	4.38
5	CRF	-	3.42
5	K-DCNRF	76 (20000)	**2.78**
7	K-DCNRF	88 (30000)	2.74

5.5.2 TIMIT Experiments

We examine our proposed method on TIMIT phone classification tasks. We extract MFCC features and add delta and delta-delta information, with a window size of 25 ms and step size of 10 ms. To include contextual information, we concatenate a context of five frames on each side, with 11 frames in total. Each feature vector contains $39 \times 11 = 429$ elements. In the training set, we use all data from 462 speakers, which contains about

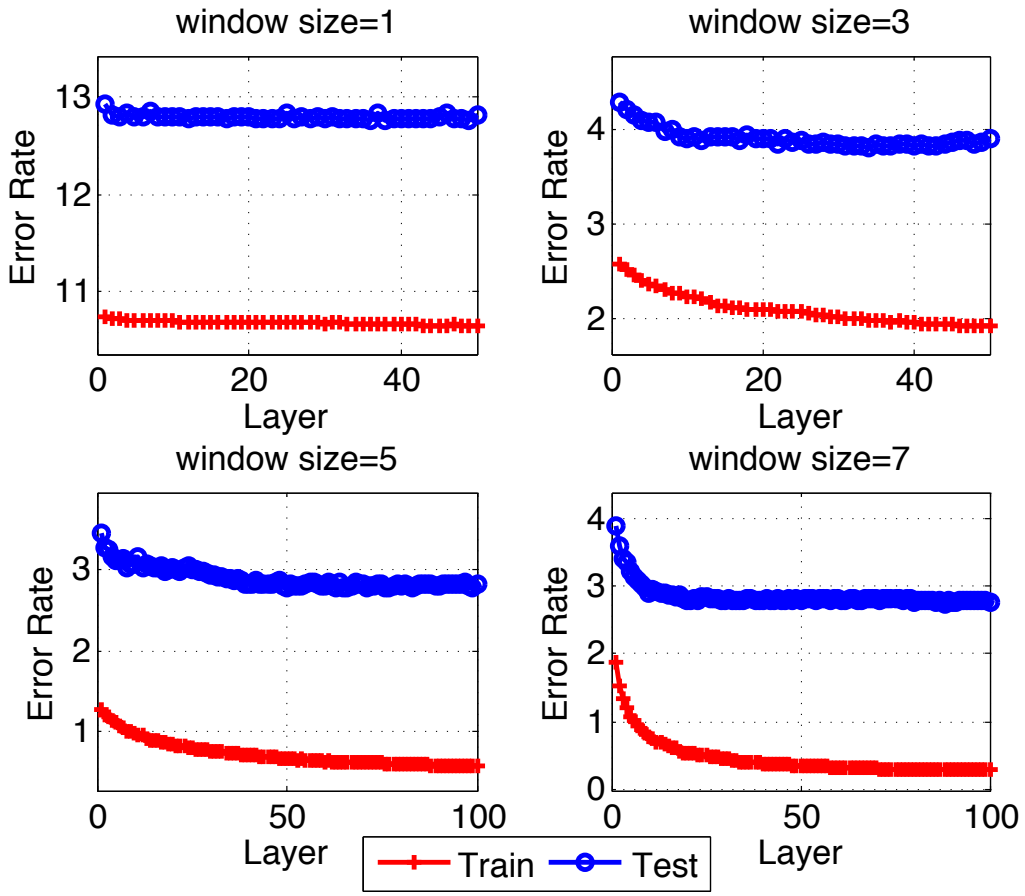


Figure 5.2: Experimental results on the ATIS dataset, where the abscissa axis is the layer number and ordinate axis is the error rate.

1.12 million samples. We use the standard TIMIT development set of 50 speakers, with 22,488 samples, for cross validation on parameters λ and σ. We test our method on the TIMIT core test set, consisting of 192 utterances with 57,920 samples. In the experiment, we use 183 senone labels as targets for K-DCNRF. The 183 target labels correspond to 61 phone units defined in TIMIT and can be further mapped to 39 phone classes. The frame-level state classification error is shown in Table 5.4. K-DCNRF outperforms its predecessor, DCN [93, 94], and shallow models [107]. Table 5.5 shows the detailed results of K-DCN with 44000 projection dimensions on the TIMIT core test set.

Table 5.4: Frame-level classification error rates of states.

Method	Layer (hidden units/proj. dim)	Frame-level State Err.(%) (183 classes)
SVM	-	60.3
OMP [107]	-	48.9
DCN ([93])	6 (6000)	44.24
DCN ([93])	6 (7000)	44.04
DSN ([94])	8 (6000)	43.86
K-DCNRF	4 (44000)	**42.87**

Table 5.5: Detailed K-DCNRF frame-level classification error (projection dimension = 44000).

Layer	Frame-level Phone Err.(%) (39 classes)	Frame-level Phone Err.(%) (61 classes)	Frame-level State Err.(%) (183 classes)
1	37.49	43.72	54.97
2	28.74	34.39	45.11
3	26.77	32.29	42.97
4	26.60	32.20	42.87
5	26.58	32.24	43.13

5.6 Summary

This chapter described a method for efficiently training and testing kernel deep convex networks (K-DCN) using the Bochner Fourier-space sampling approximation of an RBF kernel. The computational savings afforded by this approach, enabled the application of K-DCN to solve classification tasks with on the order of a million or more training vectors. Training sets this large were impractical using the previous approaches. Evaluating on the task of frame-level state classification on the TIMIT data with about 1.12 million training samples, we used the proposed approach to reduce the error rate by 29% compared to an SVM. Since the Bochner's approximation allows significant reduction in the computational complexity of a K-DCN, this approach will facilitate future work aiming to apply the K-DCN to large vocabulary speech recognition.

Chapter 6

Shallow Learning Matches Deep Learning

6.1 Introduction

The recent dramatic success of Deep Neural Networks (DNNs) in speech recognition [17] highlights the statistical benefits of marrying highly nonlinear and near-nonparametric models with large datasets, with efficient optimization algorithms running in distributed computing environments. Deep learning models project input data through several layers of nonlinearity and learn different levels of abstraction. The composition of multiple layers of nonlinear functions can approximate a rich set of naturally occurring input-output dependencies. At the same time, the combinatorial difficulty of performing exhaustive model selection in the discrete space of DNN architectures and the potential to get trapped in local minima are well recognized as valid concerns. In this chapter, we revisit kernel methods, considered "shallow" in the DNN sense, for large-scale nonparametric learning. We ask whether, despite their shallow architecture and convex formulations, recent advances in scaling-up kernel methods via randomized algorithms allow them to match DNN performance.

Let $\mathcal{X} \subset \mathbb{R}^d$ be in input domain of acoustic features. With a kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, there is an associated reproducing kernel Hilbert space of functions H_k, with inner product $\langle \cdot, \cdot \rangle_{H_k}$ and norm $\| \cdot \|_{H_k}$, in which model estimation can be performed by minimizing a regularized objective function,

$$f^\star = \text{argmin}_{f \in H_k} \frac{1}{n} \sum_{i=1}^{n} V(\mathbf{y}_i, f(\mathbf{x}_i)) + \lambda \|f\|_{H_k}^2$$

where $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ denotes a training set, λ is a regularization parameter and $V(\cdot, \cdot)$ denotes a loss function. In this chapter, we work with the squared loss $V(\mathbf{y}, \mathbf{t}) = (\mathbf{y} - \mathbf{t})^2$ function so that the model estimation above is for kernel

ridge regression, although all our techniques generalize to other loss functions. We also mainly work with the Gaussian kernel, though the randomization techniques considered in this chapter apply to a broader family of kernels. According to the classical representer theorem [27], the minimizer for the above problem has the form,

$$f^*(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \tag{6.1}$$

Plugging this formula back into the objective function with squared loss yields a dense linear system for the coefficients α:

$$(\mathbf{K} + \lambda \mathbf{I})\alpha = \mathbf{y} \tag{6.2}$$

where $\mathbf{y}$ is the vector of labels and $\mathbf{K}$ is the $n \times n$ Gram matrix given by $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. It is the $O(n^2)$ storage requirements of the Gram matrix, the $O(n^3)$ computational expense of solving a large dense linear system during training, and the need to evaluate in $O(nd)$ time the sum in (6.1) during testing (assuming that evaluation of the kernel takes $O(d)$ time) that make kernel methods in this form rather unappealing for large datasets.

To handle the limitations in large-scale tasks, several approaches have been proposed. One of the popular directions is to approximate the kernel matrix by linearization, such as the Nyström method [100, 101] and random Fourier features based methods [7, 8]. Although the Nyström method has shown better performance theoretically [108], it involves more expensive computations and larger memory requirements for large-scale tasks. Hence, our focus in this chapter is on improving the efficiency of random Fourier features based methods.

A kernel function $k(\mathbf{x}, \mathbf{y})$, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ can be associated with a high-dimensional inner product feature space H such that $k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_H$. One such feature space is H_k itself, while another can be obtained through the Mercer's theorem. For the Gaussian kernel, the dimensionality of these feature spaces is infinite and hence it is preferred to perform computations implicitly using the kernel and its Gram matrix. However, this is the source of increased computational complexity with respect to the number of data points.

Instead of using the implicit feature mapping in the kernel trick, Rahimi and Recht proposed a random feature method for approximating kernel eval-

uation [7]. The idea is to explicitly map the data to a Euclidean inner product space using a randomized feature map $\mathbf{z} : \mathbb{R}^d \to \mathbb{R}^D$ such that the kernel evaluation can be approximated by the Euclidean inner product between the transformed pair:

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \approx \mathbf{z}(\mathbf{x})^{\mathrm{T}} \mathbf{z}(\mathbf{y}) \tag{6.3}$$

This feature map is justified via Bochner's theorem which associates each continuous shift-invariant kernel on $\mathbb{R}^d$ with a unique probability density, from which a Monte-Carlo sampling procedure defines a random feature map. In this construction, the Gaussian kernel can be expressed as the expectation of $z_\omega(\mathbf{x}) z_\omega(\mathbf{y})$, where $z_\omega(\mathbf{x}) = \cos(\omega^{\mathrm{T}} \mathbf{x} + \mathbf{b})$, $\omega \in \mathbb{R}^d$ is drawn from a Gaussian distribution, $\mathbf{b}$ is drawn from a uniform distribution on $[0, \pi)$, and the cosine function is applied entry-wise. To obtain a lower-variance estimate, we can concatenate D randomly chosen z_ω into a column vector $\mathbf{z}$ and normalize each component by $1/\sqrt{D}$. Therefore, the inner product $\mathbf{z}(\mathbf{x})^{\mathrm{T}} \mathbf{z}(\mathbf{y}) = \frac{1}{D} \sum_{j=1}^{D} z_{\omega_j}(\mathbf{x}) z_{\omega_j}(\mathbf{y})$ is a lower variance approximation of the kernel function $k(\mathbf{x}, \mathbf{y})$. With this approximation, instead of solving (6.2), one can instead solve a $D \times D$ standard regularized linear regression problem,

$$(\mathbf{Z}^{\mathrm{T}} \mathbf{Z} + \lambda \mathbf{I}) \mathbf{w} = \mathbf{Z}^{\mathrm{T}} \mathbf{y}$$

where $\mathbf{Z}$ is a feature space data matrix whose rows are $\mathbf{z}(\mathbf{x}_i), i = 1 \ldots n$. Predictions can be made with $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \mathbf{z}(\mathbf{x})$. The storage cost of materializing $\mathbf{Z}$ is $O(nD)$; while the training and testing complexity is $O(nD^2)$ and $O(dD)$ respectively. These are much more appealing due to linear scaling with respect to number of training points, provided D is small enough. More recent techniques [8] offer similarly performing random Fourier feature maps, with faster mapping, which translates into reducing the time needed for generating $\mathbf{Z}$ and faster predictions.

In practice, when the number of training examples increase and the dimension of original features is large, the minimum random feature space dimensionality $D^\star$ needed for competitive performance tends to be very large, and that poses a scalability problem. As an example, the TIMIT dataset needs $D = 400,000$ to get performance on par with state-of-the-art methods. In such settings, materializing the corresponding feature space matrix $\mathbf{Z}$ requires several terabytes of storage, leading to a large, dense least-squares

regression problem, which likely needs an out-of-core solver to solve exactly.

This chapter is motivated by the question of how to make randomized feature map techniques for estimating nonlinear kernel models more scalable. Toward this objective, we propose two complimentary techniques:

- We describe a parallel least squares solver that is specialized for kernel ridge regression with random Fourier feature approximations. Our solver does not materialize $\mathbf{Z}$ explicitly, but rather computes blocks of the associated covariance matrix $\mathbf{Z}^{\mathrm{T}}\mathbf{Z}$ in one pass, followed by on-the-fly calculations in a distributed block coordinate descent procedure. As a side benefit of using an iterative solver, we find that with early-stopping, an explicit regularizer is not needed. In practice, we therefore set $\lambda = 0$. This implies that our approach only requires a single kernel parameter (the bandwidth of the Gaussian kernel). Here, we demonstrate its value for speech recognition problems.
- We develop an ensemble learning approach where multiple low-dimensional random Fourier models are combined together. While the initial models focus on the entire dataset, the subsequent models focus on high-error areas of the input domain. This enables more efficient data fitting.

Our results indicate that on TIMIT, the generalization performance of deep learning non-convex models can be matched by "shallow" kernel machines with convex optimization using the proposed schemes. Informally, when we say "match" we mean that the best generalization performance obtained on TIMIT with our scalable kernel ridge regression approaches is very similar or better than achieved with DNN (with MSE loss), with comparable number of parameters. We do not report a training time comparison due to differences in the underlying compute environment: our distributed solvers for kernel-methods were trained on an IBM BlueGene/Q Nodecard (32 nodes x 16 cores per node) while DNNs were trained on a GPU.

The remainder of this chapter is organized as follows. Section 6.2 reviews some related work. Section 6.3 introduces the proposed algorithms: the ensemble kernel machines and the scalable solver. The experimental setups, along with results, are described in Section 6.4. Section 6.5 concludes this chapter.

6.2 Related Work

Cho and Saul [109] proposed an arc-cosine kernel function, which mimics the computation in a multilayer neural network. Cheng and Kingsbury further applied the arc-cosine kernel to TIMIT speech recognition task [110]. The sub-sampling strategy in computing the kernel functions with millions of training data, and the usage of MFCC-LDA features lead to a worse result compared with deep neural networks. Huang et al. proposed a kernel deep convex network [95, 111]. The architecture consists of kernel ridge regression with random Fourier approximation in each layer. The model is convex in each layer, but is not convex overall. Furthermore, the dimensionality of random Fourier approximation is relatively small such that the approximation error of the kernel machine is high.

6.3 Enhancing the Scalability of Random Fourier Features

In this section, we consider the multivariate kernel ridge regression setting with a k-dimensional output space. We denote $\mathbf{Y} = [\mathbf{y}_1 \dots \mathbf{y}_k]$ where $\mathbf{Y}_{ij} = 1$ if the i-th training data has label j and $\mathbf{Y}_{is} = -1, s \neq j$ otherwise. Correspondingly, the matrix of coefficients is represented by $\mathbf{W} = [\mathbf{w}_1 \dots \mathbf{w}_k]$.

6.3.1 Ensemble of Kernel Machines

As mentioned in Section 6.1, when the number of training examples increases and the original feature dimension is high, the dimensionality $D^\star$ also needs to be high for competitive generalization performance approaching that of exact kernel methods. When the dimensionality is high, the memory usage and computation of the feature map is also costly.

We can tradeoff speed and accuracy with feature maps with dimensionality $D < D^\star$. One may view such a random features model as a weak learner, whose performance can be improved by generating an ensemble of multiple weak learners. Hence, we propose to incorporate the idea of ensemble methods [112] to aggregate the models from different weak learners trained with $D < D^\star$ random features, as shown in Algorithm 1. Initially

Algorithm 1: A Proposed Ensemble Algorithm

1: Input: training data $\mathbf{X}$ and target $\mathbf{Y}$
2: Initialize distribution: $\mathbf{d}_1(i) \leftarrow 1/n$, $i = 1, \ldots, n$
3: **for** each learner $t = 1, \ldots, T$ **do**
4: Sample N samples from $\mathbf{X}$ according to distribution $\mathbf{d}_t$, generating $\mathbf{X}_t$.
5: Generate a random Fourier feature map, mapping $\mathbf{X}_t$ to $\mathbf{Z}_t$.
6: Solve $\min_{\mathbf{W}_t} \|\mathbf{Z}_t\mathbf{W}_t - \mathbf{Y}_t\|_F^2$
7: Prediction $\hat{\mathbf{y}}_t \leftarrow \mathbf{Z}_t\mathbf{W}_t$.
8: Compute the weighted training error: $e_t \leftarrow \sum_{i=1}^{N} \mathbf{d}_t(i)(\hat{\mathbf{y}}_{t_i} \neq \mathbf{y}_i)$
9: Define $\alpha_t = \frac{1}{2}(\frac{1-e_t}{e_t})$
10: Update $\mathbf{d}_{t+1}(i)$:

$$\mathbf{d}_{t+1}(i) \leftarrow \frac{\mathbf{d}_t(i)}{S_t} \times \begin{cases} e^{-\alpha_t} & \text{if } \hat{\mathbf{y}}_{t_i} = \mathbf{y}_i \\ e^{\alpha_t} & \text{if } \hat{\mathbf{y}}_{t_i} \neq \mathbf{y}_i \end{cases}$$

S_t is a normalization factor.
11: **end for**
12: Output: for each $t = 1, \ldots, T$: $\mathbf{W}_t$, α_t, feature map

all training samples are assigned with uniform weights. For each learner t, we first sample the training samples according to the weights. Given the sampled training data $\mathbf{Z}_t$ with labels $\mathbf{Y}_t$, we solve the least-squares problem $\|\mathbf{Z}_t\mathbf{W}_t - \mathbf{Y}_t\|_F^2$. Then, each training sample is reweighted based on its classification result. Difficult samples, that is, those with higher classification error, become more likely to be sampled for training the $(t+1)$-th learner. For samples which were classified incorrectly and are classified correctly in learner t, the weights will be reduced. The final decision is made based on the prediction from learners $t = 1, \ldots, T$, weighted by the classification error of each learner. This ensemble mechanism may be seen as a non-uniform sampling approach where the capacity of the overall model is distributed as per the hardness of classification.

6.3.2 Parallel Scalable Solver

We now discuss a more direct approach to handle very large random feature spaces, which in turn leads to solving very large least-squares problems of

the form

$$\min_{\mathbf{W}} \|\mathbf{ZW} - \mathbf{Y}\|_F^2 \tag{6.4}$$

Large number of random features poses a serious scalability problem, if the least squares solver is implemented naively. For example, consider the TIMIT dataset with $100,000$ random features. Just storing the matrix $\mathbf{Z}$ in memory in double-precision requires over 1 terabyte of data. Once in memory, one has to solve the dense regression problem, which will take too long for a direct method. If terabyte order memory is not available, one traditional approach is to resort to slower out-of-core solution with costly disk I/O per iteration.

The key for a scalable solver is observing that the matrix $\mathbf{Z}$ is implicitly defined by the input matrix $\mathbf{X}$. To take advantage of this we use a block coordinate descent algorithm. At each iteration block coordinate descent fixes the part of $\mathbf{W}$ that corresponds to coordinates (columns of $\mathbf{Z}$) that are outside the current block, and optimizes the part of $\mathbf{W}$ that corresponds to coordinates inside the current block (i.e., a set of rows of $\mathbf{W}$). This leads to solving the following least-squares problem in each iteration

$$\min_{\Delta \mathbf{W}_b} \|\mathbf{Z}_b \Delta \mathbf{W}_b - \mathbf{R}\|_2^F \tag{6.5}$$

where $\mathbf{Z}_b$ is the part of $\mathbf{Z}$ that corresponds to the coordinates in block b (a set of columns of $\mathbf{Z}$), $\Delta \mathbf{W}_b$ is the update to $\mathbf{W}_b$, the part of $\mathbf{W}$ that corresponds to the coordinates in block b (a set of rows of $\mathbf{W}$), and $\mathbf{R} = \mathbf{ZW} - \mathbf{Y}$ is the current residual vector. The update is then $\mathbf{W}_b \leftarrow \mathbf{W}_b + \Delta \mathbf{W}_b$.

As we see each iteration requires only a part of $\mathbf{Z}$, and there is no need to form the entire matrix. This is much more economical in memory use; our solver simply forms the relevant $\mathbf{Z}_b$ in each iteration, and solves (6.5).

Let s_b denote the size of block b. To avoid paying $O(ns_b^2)$ in each iteration, we form the Gram matrix $\mathbf{Z}_b^{\mathrm{T}} \mathbf{Z}_b$ and factor it only during the first epoch. Subsequent epochs require only $O(ns_b)$ per iteration in addition to the time required to form $\mathbf{Z}_b$.

We use parallel processing to allow us to work on large datasets. We designed our solver as a distributed-memory solver. We split the input matrix $\mathbf{X}$ and right-hand-size $\mathbf{Y}$ row-wise, with each processor receiving a subset of the rows of $\mathbf{X}$ and $\mathbf{Y}$. The solver also updates a copy of $\mathbf{R}$, which is distributed row-wise with exactly the same split as $\mathbf{X}$. All processors keep a

synchronized copy of $\mathbf{W}$. The row-wise split enables us to take advantage of the fact that mapping a row of $\mathbf{X}$ to a row of $\mathbf{Z}$ can be done independently of other rows of $\mathbf{X}$, so the distribution of $\mathbf{X}$ implicitly defines a distribution of $\mathbf{Z}$ and $\mathbf{Z}_b$. This enables the block coordinate descent to be performed with little synchronization and communication overhead. See Algorithm 2 for a pseudo-code description.

Algorithm 2: Scalable Distributed-Memory Solver

Input: training data $\mathbf{X}$ and target $\mathbf{Y}$
Distribute $\mathbf{X}$ and $\mathbf{Y}$ row-wise across machines
Initialize: $\mathbf{W} \leftarrow \mathbf{0}$, $\mathbf{R} \leftarrow \mathbf{Y}$.
Generate a random Fourier feature map (maps $\mathbf{X}$ to $\mathbf{Z}$).
while weights have not converged **do**
  for each block b **do**
    ▷ $\mathbf{Z}_b$ denotes the columns of $\mathbf{Z}$ corresponding to block b. $\mathbf{W}_b$ denotes the rows of $\mathbf{W}$ corresponding to block b.
    Compute $\mathbf{Z}_b$
    If (first epoch): compute and factor $\mathbf{Z}_b^{\mathrm{T}}\mathbf{Z}_b$
    Compute $\mathbf{Z}_b^{\mathrm{T}}\mathbf{R}$
    $\Delta\mathbf{W}_b \leftarrow (\mathbf{Z}_b^{\mathrm{T}}\mathbf{Z}_b)^{-1}\mathbf{Z}_b^{\mathrm{T}}\mathbf{R}$
    $\mathbf{W}_b \leftarrow \mathbf{W}_b + \Delta\mathbf{W}_b$
    $\mathbf{R} \leftarrow \mathbf{R} - \mathbf{Z}_b\Delta\mathbf{W}_b$
  end for
end while
Output: $\mathbf{W}$, feature map

6.4 Experiments

In this section, we examine the effectiveness of our approaches on the TIMIT corpus for phone classification and recognition tasks.

6.4.1 Corpus

Phone classification and recognition experiments were evaluated on the TIMIT corpus. The training set contains 3696 SI and SX sentences from 462 speakers. A separate development set of 50 speakers was used for hyperparameter tuning. Experiments are evaluated on the core test set, which consists of 192

utterances, with 8 utterances from each of the 24 speakers. The SA sentences are excluded from tests.

In all experiments, we use 40-dimensional feature space maximum likelihood linear regression (fMLLR) features [113] and then concatenate the neighboring 5 frames (11 frames in total) as the input feature for the experiments, which is reported as the state-of-the-art feature for deep neural network by Mohamed et al. [114]. In this chapter, we focus on training kernel machines with the mean square error (MSE) objective and compare the results with DNNs using MSE and cross entropy (CE) objectives. To have a fair comparison, we select the best architecture (number of hidden units and number of layers) for DNNs based on the development set. We report the number of hidden units and the number of hidden layers for the selected DNNs in Tables 6.1, 6.2, and 6.3.

6.4.2 Classification Task

We first look at the frame-level classification results on the TIMIT dataset. We use 147 (49 phonemes $\times$ 3 states) context independent states as targets. The training data consists of 2 million frames of fMLLR features, extracted using a 5 ms step size.

As mentioned in Section 6.1, the training and evaluation of an exact kernel ridge regression model is computationally constrained by the number of training samples. Hence, in order to evaluate performance with an exact kernel method, given computational limitations, we use a subset of 100,000 training samples (2 million frames). Table 6.1 shows the results. We can observe that the exact kernel achieves the best results by fitting the training data well. The error of random Fourier (RF) models decreases as the number of random features increases. By constructing an ensemble of several RF models, the kernel machines are able to get closer to the full kernel performance. The performance of kernel machines is significantly better than DNN with MSE objectives.

Table 6.2 shows the frame-level classification results on the full training set. Here, the exact kernel method becomes computationally infeasible. By increasing the number of random features of RF models using the scalable solver or combining several relatively low-dimensional RF models, the per-

Table 6.1: Classification results (147 classes) on 100 K samples, where D is the number of random features.

Model (D - T learners)	Training Error (%)	Test Error (%)
Full RBF	0.07	38.61
MSE-RF (5K - 1)	33.32	44.08
MSE-RF (5K - 30)	3.69	39.59
MSE-RF (10K - 1)	23.70	42.08
MSE-RF (10K - 30)	0.20	39.33
Model (hidden units - layers)	Training Error (%)	Test Error (%)
MSE-DNN (1K - 1)	49.71	50.13
MSE-DNN (1K - 3)	36.79	40.12
MSE-DNN (1K - 5)	41.55	43.23

Table 6.2: Classification results (147 classes) on the full training set, where D is the number of random features.

Model (D - T learners)	Training Error (%)	Test Error (%)
MSE-RF (60K - 1)	27.16	35.95
MSE-RF (60K - 7)	14.56	**33.5**
Scal. MSE-RF (200K - 1)	18.43	34.08
Scal. MSE-RF (400K - 1)	11.95	**33.67**
Scal. MSE-RF (600K - 1)	9.15	33.69
Model (hidden units - layers)	Training Error (%)	Test Error (%)
MSE-DNN (4K - 2)	21.57	**33.53**
CE-DNN (1K - 3)	22.05	**32.99**

formance of MSE-RF is similar to the best results by MSE-DNN. The DNN with CE objective, minimizing the classification objectives directly, achieves the best results.

Figures 6.1 and 6.2 show the training and test set classification result comparison between kernel machines and DNN with MSE criterion using 100 K training samples and the full training set, respectively. We compare the classification results using models with different number of parameters, where the parameters of a DNN model denote the parameters of the weight matrices and bias vectors, and the parameters of the kernel machines denote the parameters of the weight matrix $\mathbf{W}$ in Eq. (6.4). From Figures 6.1 and 6.2, we can observe that, as the number of parameters increases, DNN and kernel machines achieve close test errors. For the training error, DNN

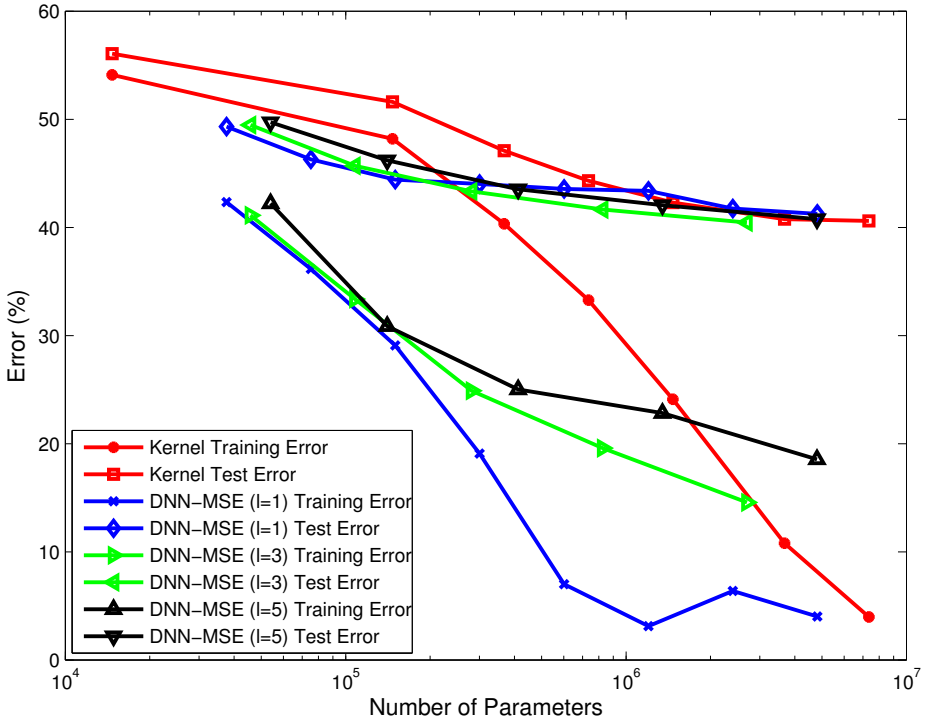


Figure 6.1: Classification result comparison with 100 K training samples, where $l = k$ represents k hidden layers in the DNN model.

models achieve higher training error when the number of layers increases (e.g., $l = 5$). On the other hand, kernel machines achieves lower training error compared to DNN models. Note that the different training and testing behaviors might be also related to different hyperparameters tuning strategies; that is, the learning rate of the DNN models is tuned adaptively during training based on the development set performance [115], whereas the hyperparameters of kernel machines are fixed during training and selected based on the development set performance.

6.4.3 Recognition Task

To convert the output of kernel machines for the hidden Markov model (HMM) Viterbi decoding, we follow the hybrid approach in [110]. We trained a weighted softmax function with the CE objective for phone posterior $p(q = i|\mathbf{x})$.

$$p(q = i \mid \mathbf{x}) = \frac{\exp(\sum_j a_{ij} p_{ij} + b_i)}{\sum_k \exp(\sum_j a_{kj} p_{kj} + b_k)} \tag{6.6}$$

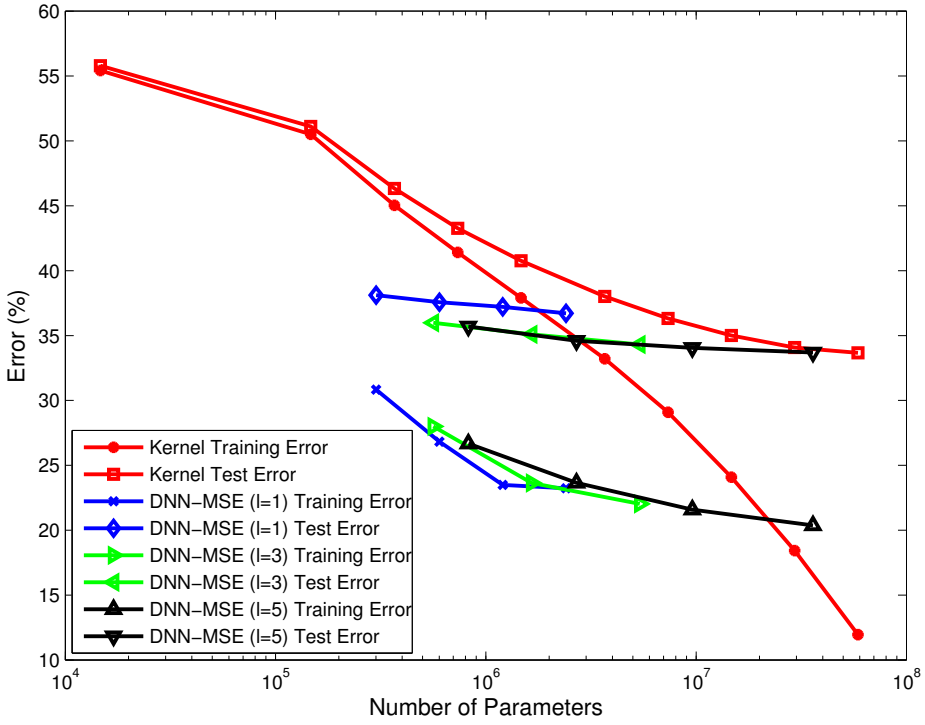


Figure 6.2: Classification result comparison with the full training set, where $l = k$ represents k hidden layers in the DNN model.

where a_{ij} is a trainable weight for kernel output p_{ij} and b_i is a trainable bias for phone i. The weights are trained using stochastic gradient descent. By Bayes rule, $p(\mathbf{x}|q = i) \propto p(q = i|\mathbf{x})/p(q = i)$, the posteriors can be transformed to HMM state emission probabilities. The priors $p(q = i)$ are estimated from the training data. Then, Viterbi decoding is used for phone recognition. Table 6.3 shows the recognition results on TIMIT. The MSE-RF achieves better performance compared to MSE-DNN and slightly worse performance than CE-DNN. Note that the DNNs chosen are the best performing models in terms of the number of hidden units and number of layers on the development set.

6.5 Summary

In this chapter, we explore the comparison between deep learning models and shallow kernel machines. By using an ensemble approach and a complimentary scalable solver, we show that the model expressibility of kernel

Table 6.3: Recognition results comparison, where D is the number of random features, CI Error is the context independent state classification error rate, and PER is the phone error rate.

Model (hidden units - layers)	Test CI Error (%)	Test PER (%)
MSE-DNN (4K - 2)	33.53	22.3
MSE-DNN (2K - 2)	34.12	**22.2**
CE-DNN (1K - 3)	32.99	21.7
CE-DNN (4K - 3)	33.34	**20.5**
Model (D - T learners)	Test CI Error (%)	Test PER (%)
Scal. MSE-RF (400K - 1)	33.67	**21.3**

machines can match deep neural network models, for very similar generalization performance. Furthermore, kernel machines have the advantages of having a convex optimization based formulation and simpler model selection.

Chapter 7

Conclusions and Future Work

In this dissertation, we study deep learning and kernel machines for audio and natural language processing. A background study (Chapter 2) finds that deep learning models and kernel machines both have high expressive power to model complex dependencies in real-world tasks. In the first part of this thesis, Chapter 3 and 4, we study deep learning for problems beyond classification by considering problems with structured relationships between and among inputs and outputs. Because of the practical difficulties in training deep learning models (selecting the right architecture, potential to get trapped in local minima, etc.), we explore kernel machines in the second part of this thesis, Chapter 5 and 6. We explore efficiently scaling up kernel machines with random Fourier approximation. Given the computational limitations for large number of random Fourier features, we investigate integrating kernel machines with deep architectures, an ensemble learning framework with kernel machines, and a parallel scalable least-squared solver. In this chapter, we summarize the contributions of this dissertation, discuss its impact and limitations, and future research work.

7.1 Summary of Contributions

The main contributions of this dissertation are as follows:

Deep Learning for Structured Outputs: Joint Optimization of Time-frequency Masks and Deep Recurrent Neural Networks

We investigate deep learning models for problems with structured relationships among outputs. Specifically, in monaural source separation tasks, we exploit the competing relationship among output predictions and the constraint that the sum of the output predictions is the original mixtures. We

propose a general framework jointly optimizing the time-frequency masking functions and deep learning models for monaural source separation problems. Given we jointly model two sources as outputs, we propose a discriminative training objective to further minimize the interference between the two sources. Our proposed approach can be generally applied to problems with two (or multiple) targets. We evaluate the proposed model for speech separation, singing voice separation, and speech denoising tasks. Our proposed model achieves state-of-the-art performance in various conditions (speech separation for the same gender speech and different gender speech, singing voice and background music, speech denoising for seen and unseen environments).

Deep Learning for Structured Inputs: Deep Structured Semantic Models

We investigate deep learning models for problems with structured relationships among inputs. We propose the deep structured semantic models which represent latent semantic meaning of queries and documents for information retrieval tasks. We model queries and documents as inputs to the deep learning models and measure the relevance through the similarity at the output layer. The main contributions of our approach are in three key aspects. First, we directly optimize the document ranking objectives by the use of clickthrough data. We propose a discriminative objective function to exploit the similarity and dissimilarity between queries and documents. Second, we extend previous linear semantic models (such as latent semantic analysis (LSA)) to nonlinear models using multiple hidden representations. The deep architectures further enhance the model capacity to capture and represent more sophisticated semantic structures within queries and documents. Third, we propose a letter n-gram-based technique, word hashing, in scaling up the training of deep learning models with very large vocabularies, which is essential in the web search task. In our experiments, we show that the new techniques pertaining to each of the above three aspects lead to significant performance improvement on the document ranking task. The combination of all three sets of new techniques has led to a new state-of-the-art semantic model that beats all the previously developed competing models by a significant margin.

Jointly Shallow and Deep Learning: Random Features for Kernel Deep Convex Network

To resolve the computational limitations in kernel machines, we propose random Fourier approximations with kernel deep convex networks (K-DCNs). Given the optimization concerns of deep neural networks, we explore the use of kernel machines in deep convex networks. K-DCN is constructed by stacking multiple kernels on top of each other. Within each module, the kernel optimization is convex. For the use of kernel ridge regression in each module, we have a closed-form solution for the regression coefficients. Due to the computational limitations, we further explore using Bochner Fourier-space sampling approximation of an RBF kernel. The computational savings afforded by this approach, enable the application of K-DCN to solve classification tasks when there are more than a million training vectors. Training sets this large were impractical using the previous approaches. We evaluate our proposed model on spoken language understanding and phone classification tasks. Our proposed models outperform previous approaches including conventional deep neural networks.

Shallow Kernel Machines Match Deep Neural Networks

We investigate pushing the limit of kernel machines for large-scale applications. The previous proposed random Fourier kernel deep convex networks is convex within each module, but is not convex globally. We develop two approaches to efficiently scale-up kernel machines without relying on deep architectures. First, assuming a kernel machine approximated by a low-dimensional random Fourier projections is as a weak learner, by combining multiple weak learners together in an ensemble, we can create a stronger learner. We develop an ensemble learning framework of kernel machines with random Fourier approximations. Second, given the property of random projections and the least squares solver, we explore a disjoint approach by proposing a scalable parallel least squares solver for kernel ridge regression with random Fourier feature approximations. The solver enables us to efficiently solve a kernel ridge regression with a large number random Fourier projections in a parallel environment. We evaluate our proposed methods for phone classification and recognition tasks on the TIMIT corpus. In ad-

dition, we compare the differences between kernel machines and deep neural networks with different numbers of parameters and architectures. We show that we are able to match deep neural network performance by using kernel machines.

Optimization Error

In the statistical learning framework, as summarized in Chapter 2, there are approximation error, estimation error, and optimization error. Under some assumptions, previous works have characterized neural networks and kernel machines in terms of approximation and estimation error (or bias-variance tradeoff). The work presented in this dissertation can be regarded as reducing the third type of error (optimization error) for real-world tasks by developing training objectives that optimize better metrics, and enhancing the scalability and efficiency for large-scale tasks.

In deep learning problems, given prior knowledge about the problems, we consider structured relationships between and among inputs and outputs, which directly capture the intrinsic relationships among data. In the monaural source separation tasks, we jointly optimize the time-frequency mask with deep learning models so that the proposed model directly reconstructs the output predictions without applying a mask function separately. The discriminative training objective captures the competing relationships among outputs and further enhances the source to interference ratio for the problem. In the information retrieval task, we directly optimize the document ranking objectives using clickthrough data. Given the clicked and unclicked pairs in the data, we further enhance the models for discriminating between similar and dissimilar query-document pairs. The proposed word hashing technique further enables the models to overcome the problem of scalability in the large vocabularies case.

We study enhancing the scalability and efficiency of large-scale kernel machines. Especially, we resolve the computation limitations (memory, computation time) for a large number of random Fourier features. Given the computational limitations, we are only able to compute smaller dimensional random Fourier projections within a single machine. We have developed, using the deep convex networks, ensemble learning methods to enhance the expressibility of a low-dimensional random Fourier approximated kernel. Moreover, we

also explore a scalable parallel least-squares solver to efficiently compute a large number of random Fourier features using multiple machines in parallel.

7.2 Future Work

Building upon the previous work in deep learning and large-scale kernel machines, this dissertation also open doors for future research opportunities in related fields. Some potential research directions are summarized as follows:

Deep Learning for Structured Relationships among Data

Our proposed approaches in monaural source separation and deep structured semantic models can be viewed as a general framework exploiting the intrinsic relationships among data and directly modeling output objectives in the end-to-end way. By incorporating intermediate processing steps during training and optimizing the target objectives directly, models can learn the intrinsic relationship among data and can be optimized towards desired objectives.

The proposed model, jointly optimizing masks with deep recurrent neural networks, exploits the relationships among input and outputs, incorporates the masking operation during training, and thereby optimizes directly for the masked results. The model can be directly applied to modeling multiple sources such as different music instruments for source separation problems by incorporating the relationships among data. The proposed framework can also be applied to optimize speech senones for robust automatic speech recognition problems. Beyond our current proposed approach modeling output targets at every single time step, it is possible to further extend the model objectives for modeling a sequence of structured outputs, which will be useful to ensure the smoothness of time series signals such as audio or speech. Moreover, as we explored during our experiments, it is important to improve the generalization ability to unseen or mismatch environments.

The proposed deep structured semantic model exploits the similarity and dissimilarity between clicked and unclicked query-document pairs, respectively, and discriminatively optimizes objectives directly for information retrieval tasks. The proposed framework can be viewed as a general approach to model the semantic similarity between two different entities. The frame-

Table 7.1: The extensions and future directions of deep structured semantic models.

Tasks	Source	Target
Web search [118, 119]	query	web documents
Question answering [120]	question	answer
Machine translation [121]	sentence in language A	translations in language B
Automatic Image captioning [122]	image	caption
Recommendation [123]	key phrase and context	recommendation documents
Summarization	document	summary
Knowledge-base construction	entity	entity

work can be naturally extended to different extensions and potential applications. For example, in the web search task, the model optimizes the semantic similarity between query and clicked documents; Similarly, in the question-answering tasks, the model can be applied to optimize the semantic similarity between a question and an answer. A summary of potential extensions and applications is shown in Table 7.1, which are also mentioned in [116, 117].

Large-scale Kernel Machines

To resolve computational limitations for large datasets, our proposed models utilize random Fourier features with deep convex networks, ensemble models, and a parallel scalable solver. To further enhance the model scalability and efficiency for large-scale applications, a promising direction is the use of Quasi-Monte Carlo techniques to improve the efficiency of explicit feature map approximations following the lines in [124]. Another interesting direction is to improve the learning of kernel machines in the end-to-end sense. For example, we explore using random Fourier features with kernel ridge regression whose training objective is the mean-squared error criterion. It is important to further explore the direct optimization of kernel machines for the desired objectives. Optimizing the cross-entropy objective could be a better metric in the classification task. In addition, the input features to kernel machines could also be learned from the data. While our work utilizes features such as feature space maximum likelihood linear regression (fMLLR) or Mel-frequency Cepstral Coefficients (MFCC), a recent work achieves comparable results with deep neural networks using kernel machines with feature learning [125]. Finally, it is essential to explore efficient optimization algo-

rithms for large-scale kernel machines. One of the key elements for deep learning models is the efficient utilization of powerful Graphics Processing Units (GPUs) for large datasets. To further push the limits of kernel machines, there is room to further enhance the computation efficiency for large datasets algorithmically along the lines proposed in [126, 127].

References

[1] D. Liu, P. Smaragdis, and M. Kim, "Experiments on deep learning for speech denoising," in *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2014, pp. 2685–2689.

[2] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *Algorithmic Learning Theory*, ser. Lecture Notes in Computer Science, 2011, vol. 6925, pp. 18–36.

[3] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[4] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Schölkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.

[5] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[7] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2007, pp. 1177–1184.

[8] Q. Le, T. Sarlós, and A. Smola, "Fastfood – Approximating kernel expansions in loglinear time," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[9] V. N. Vapnik, *Statistical Learning Theory*. Wiley New York, 1998, vol. 2.

[10] O. Bousquet, S. Boucheron, and G. Lugosi, "Introduction to statistical learning theory," in *Advanced Lectures on Machine Learning*, ser. Lecture Notes in Computer Science. Springer, 2004, vol. 3176, pp. 169–207.

[11] O. Bousquet and L. Bottou, "The tradeoffs of large scale learning," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2007, pp. 161–168.

[12] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[13] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, pp. 807–814.

[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[15] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[16] L. Deng and D. Yu, *Deep Learning: Methods and Applications*. Now Publishers, 2014.

[17] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[18] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[19] A. R. Barron, "Approximation and estimation bounds for artificial neural networks," *Machine Learning*, vol. 14, no. 1, pp. 115–133, 1994.

[20] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation*, vol. 1, no. 1, pp. 151–160, 1989.

[21] J. Hastad, "Almost optimal lower bounds for small depth circuits," in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, 1986, pp. 6–20.

[22] O. Delalleau and Y. Bengio, "Shallow vs. deep sum-product networks," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2011, pp. 666–674.

[23] F. Anselmi, J. Z. Leibo, L. Rosasco, J. Mutch, A. Tacchetti, and T. Poggio, "Unsupervised learning of invariant representations in hierarchical architectures," *arXiv preprint arXiv:1311.4158*, 2013.

[24] M. Anthony and P. L. Bartlett, *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 2009.

[25] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[26] A. Berlinet and C. Thomas-Agnan, *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer, 2004.

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

[28] Y. Zhang, J. Duchi, and M. Wainwright, "Divide and conquer kernel ridge regression," in *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*, 2013, pp. 592–617.

[29] T. Zhang, "Learning bounds for kernel regression using effective data dimensionality," *Neural Computation*, vol. 17, no. 9, pp. 2077–2098, 2005.

[30] M. D. Buhmann, *Radial Basis Functions*. Cambridge University Press, 2003.

[31] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

[32] P.-S. Huang, S. D. Chen, P. Smaragdis, and M. Hasegawa-Johnson, "Singing-voice separation from monaural recordings using robust principal component analysis," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 57–60.

[33] A. L. Maas, Q. V. Le, T. M. O'Neil, O. Vinyals, P. Nguyen, and A. Y. Ng, "Recurrent neural networks for noise reduction in robust ASR," in *Proceedings of the 13th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2012, pp. 22–25.

[34] P. Sprechmann, A. Bronstein, and G. Sapiro, "Real-time online singing voice separation from monaural recordings using robust low-rank modeling," in *Proceedings of the 13th International Society for Music Information Retrieval (ISMIR)*, 2012.

[35] Y.-H. Yang, "Low-rank representation of both singing voice and music accompaniment via learned dictionaries," in *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, 2013.

[36] Y.-H. Yang, "On sparse and low-rank matrix decomposition for singing voice separation," in *Proceedings of the 20th ACM International Conference on Multimedia*, 2012, pp. 757–760.

[37] S. Boll, "Suppression of acoustic noise in speech using spectral subtraction," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 27, no. 2, pp. 113–120, 1979.

[38] Y. Ephraim and D. Malah, "Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, no. 6, pp. 1109–1121, 1984.

[39] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[40] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1999, pp. 50–57.

[41] P. Smaragdis, B. Raj, and M. Shashanka, "A probabilistic latent variable model for acoustic modeling," in *Proceedings of the Advances in Models for Acoustic Processing, Neural Information Processing Systems Workshop*, vol. 148, 2006.

[42] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Deep learning for monaural speech separation," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 1562–1566.

[43] P.-S. Huang and M. Kim and M. Hasegawa-Johnson and P. Smaragdis, "Singing-voice separation from monaural recordings using deep recurrent neural networks," in *Proceedings of the 15th International Society for Music Information Retrieval (ISMIR)*, 2014.

[44] P. Kabal, "TSP speech database," McGill University, Montreal, Quebec, Tech. Rep., 2002.

[45] C.-L. Hsu and J.-S. Jang, "On the improvement of singing voice separation for monaural recordings using the MIR-1K dataset," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 2, pp. 310–319, 2010.

[46] J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren, and V. Zue, "TIMIT: Acoustic-phonetic continuous speech corpus," Linguistic Data Consortium, 1993.

[47] Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, "A regression approach to speech enhancement based on deep neural networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 1, pp. 7–19, 2015.

[48] D. Wang, "Time-frequency masking for speech separation and its potential for hearing aid design," *Trends in Amplification*, vol. 12, pp. 332–353, 2008.

[49] F. Weninger, F. Eyben, and B. Schuller, "Single-channel speech separation with memory-enhanced recurrent neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 3709–3713.

[50] S. Nie, H. Zhang, X. Zhang, and W. Liu, "Deep stacking networks with time series for speech separation," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 6667–6671.

[51] A. Narayanan and D. Wang, "Ideal ratio mask estimation using deep neural networks for robust speech recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 7092–7096.

[52] Y. Wang and D. Wang, "Towards scaling up classification-based speech separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 7, pp. 1381–1390, 2013.

[53] Y. Wang, A. Narayanan, and D. Wang, "On training targets for supervised speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 12, pp. 1849–1858, 2014.

[54] Y. Tu, J. Du, Y. Xu, L. Dai, and C.-H. Lee, "Deep neural network based speech separation for robust speech recognition," in *Proceedings of the International Symposium on Chinese Spoken Language Processing*, 2014, pp. 532–536.

[55] E. Grais, M. Sen, and H. Erdogan, "Deep neural networks for single channel source separation," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 3734–3738.

[56] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 190–198.

[57] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in *Proceedings of the International Conference on Learning Representations*, 2014.

[58] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 15, 2011, pp. 315–323.

[59] M. C. Mozer, "A focused back-propagation algorithm for temporal pattern recognition," *Complex Systems*, vol. 3, no. 4, pp. 349–381, 1989.

[60] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[61] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Backpropagation: Theory, Architectures, and Applications*, 1995, pp. 433–486.

[62] O. Yilmaz and S. Rickard, "Blind separation of speech mixtures via time-frequency masking," *IEEE Transactions on Signal Processing*, vol. 52, no. 7, pp. 1830–1847, 2004.

[63] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1462–1469, 2006.

[64] J. Bruna, P. Sprechmann, and Y. Lecun, "Source separation with scattering non-negative matrix factorization," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[65] C. Taal, R. Hendriks, R. Heusdens, and J. Jensen, "An algorithm for intelligibility prediction of time-frequency weighted noisy speech," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2125–2136, 2011.

[66] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[67] J. Li, D. Yu, J.-T. Huang, and Y. Gong, "Improving wideband speech recognition using mixed-bandwidth training data in CD-DNN-HMM," in *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*, 2012, pp. 131–136.

[68] A. Ozerov, P. Philippe, F. Bimbot, and R. Gribonval, "Adaptation of Bayesian models for single-channel source separation and its application to voice/music separation in popular songs," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 5, pp. 1564–1578, 2007.

[69] F. Weninger, J. R. Hershey, J. Le Roux, and B. Schuller, "Discriminatively trained recurrent neural networks for single-channel speech separation," in *Proceedings of the IEEE Global Conference on Signal and Information Processing (GlobalSIP) Symposium on Machine Learning Applications in Speech Processing*, 2014, pp. 577–581.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[72] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[73] S. T. Dumais, T. A. Letsche, M. L. Littman, and T. K. Landauer, "Automatic cross-language retrieval using latent semantic indexing," in *Proceedings of the AAAI Spring Symposium on Cross-Language Text and Speech Retrieval*, vol. 15, 1997.

[74] J. C. Platt, K. Toutanova, and W.-T. Yih, "Translingual document representations from discriminative projections," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010, pp. 251–261.

[75] J. Gao, X. He, and J.-Y. Nie, "Clickthrough-based translation models for web search: From word models to phrase models," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, 2010, pp. 1139–1148.

[76] J. Gao, K. Toutanova, and W.-T. Yih, "Clickthrough-based latent semantic models for web search," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2011, pp. 675–684.

[77] W.-T. Yih, K. Toutanova, J. C. Platt, and C. Meek, "Learning discriminative projections for text similarity measures," in *Proceedings of the 15th Conference on Computational Natural Language Learning*, 2011, pp. 247–256.

[78] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005, pp. 89–96.

[79] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.

[80] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[81] L. Deng, X. He, and J. Gao, "Deep stacking networks for information retrieval," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 3153–3157.

[82] L. P. Heck, Y. Konig, M. K. Sönmez, and M. Weintraub, "Robustness to telephone handset distortion in speaker recognition by discriminative feature design," *Speech Communication*, vol. 31, no. 2, pp. 181–192, 2000.

[83] Y. Konig, L. Heck, M. Weintraub, and M. K. Sonmez, "Nonlinear discriminant feature extraction for robust text-independent speaker recognition," in *Proceedings of the RLA2C, ESCA Workshop on Speaker Recognition and its Commercial and Forensic Applications*, 1998, pp. 72–75.

[84] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding," in *Proceedings of the 14th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2013.

[85] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng, "Semantic compositionality through recursive matrix-vector spaces," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012, pp. 1201–1211.

[86] G. Tur, L. Deng, D. Hakkani-Tur, and X. He, "Towards deeper understanding: Deep convex networks for semantic utterance classification," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2012, pp. 5045–5048.

[87] G. Montavon, G. B. Orr, and K.-R. Müller, Eds., *Neural Networks: Tricks of the Trade*. Springer, 2012, vol. 7700.

[88] K. Järvelin and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents," in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2000, pp. 41–48.

[89] J. Gao, W. Yuan, X. Li, K. Deng, and J.-Y. Nie, "Smoothing clickthrough data for web search ranking," in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009, pp. 355–362.

[90] K. M. Svore and C. J. Burges, "A machine learning approach for improved bm25 retrieval," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 2009, pp. 1811–1814.

[91] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[92] B. Hutchinson, L. Deng, and D. Yu, "Tensor deep stacking networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1944–1957, 2013.

[93] L. Deng and D. Yu, "Deep convex network: A scalable architecture for speech pattern classification," in *Proceedings of the 12th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2011.

[94] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2012, pp. 2133–2136.

[95] L. Deng, G. Tur, X. He, and D. Hakkani-Tur, "Use of kernel deep convex networks and end-to-end learning for spoken language understanding," in *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*, 2012, pp. 210–215.

[96] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[97] C. J. Burges and B. Schölkopf, "Improving the accuracy and speed of support vector machines," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 1997, pp. 375–381.

[98] G. Baudat and F. Anouar, "Feature vector selection and projection using kernels," *Neurocomputing*, vol. 55, pp. 21–38, 2003.

[99] D. Achlioptas, F. Mcsherry, and B. Schölkopf, "Sampling techniques for kernel methods," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2001, pp. 335–342.

[100] P. Drineas and M. W. Mahoney, "On the Nyström method for approximating a gram matrix for improved kernel-based learning," *Journal of Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[101] S. Kumar, M. Mohri, and A. Talwalkar, "Ensemble Nyström method," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1060–1068.

[102] C. Cortes, M. Mohri, and A. Talwalkar, "On the impact of kernel approximation on learning accuracy," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 113–120.

[103] W. Rudin, *Fourier Analysis on Groups*. John Wiley & Sons, 1990.

[104] A. Singh, N. Ahuja, and P. Moulin, "Online learning with kernels: Overcoming the growing sum problem," in *Proceedings of the IEEE Workshop on Machine Learning for Signal Processing (MLSP)*, 2012.

[105] G. Tür, D. Hakkani-Tür, and L. Heck, "What's left to be understood in ATIS," in *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*, 2010.

[106] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding," in *Proceedings of the 8th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2007.

[107] O. Vinyals and L. Deng, "Are sparse representations rich enough for acoustic modeling?" in *Proceedings of the 13th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2012.

[108] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, "Nyström method vs random fourier features: A theoretical and empirical comparison," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 476–484.

[109] Y. Cho and L. K. Saul, "Kernel methods for deep learning," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 342–350.

[110] C.-C. Cheng and B. Kingsbury, "Arccosine kernels: Acoustic modeling with infinite neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011, pp. 5200–5203.

[111] P.-S. Huang, L. Deng, M. Hasegawa-Johnson, and X. He, "Random features for kernel deep convex network," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013, pp. 3143–3147.

[112] H. Xia and S. C. Hoi, "Mkboost: A framework of multiple kernel boosting," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 7, pp. 1574–1586, 2013.

[113] M. Gales, "Maximum likelihood linear transformations for HMM-based speech recognition," *Computer Speech and Language*, vol. 12, no. 2, pp. 75–98, 1998.

[114] A. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. Hinton, and M. Picheny, "Deep belief networks using discriminative features for phone recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 5060–5063.

[115] Y. Miao, "Kaldi+PDNN: building DNN-based ASR systems with Kaldi and PDNN," *arXiv preprint arXiv:1401.6984*, 2014.

[116] X. He, J. Gao, and L. Deng, "Deep learning for natural language processing and related applications (Tutorial at ICASSP)," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014.

[117] X. He, J. Gao, and L. Deng, "Deep learning for natural language processing: Theory and practice (Tutorial)," in *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management (CIKM)*, 2014.

[118] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM)*, 2013, pp. 2333–2338.

[119] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management (CIKM)*, 2014, pp. 101–110.

[120] W.-T. Yih, X. He, and C. Meek, "Semantic parsing for single-relation question answering," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2014.

[121] J. Gao, X. He, W.-T. Yih, and L. Deng, "Learning continuous phrase representations for translation modeling," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2014.

[122] H. Fang, S. Gupta, F. N. Iandola, R. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, C. L. Zitnick, and G. Zweig, "From captions to visual concepts and back," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[123] J. Gao, P. Pantel, M. Gamon, X. He, L. Deng, and Y. Shen, "Modeling interestingness with deep neural networks," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[124] J. Yang, V. Sindhwani, H. Avron, and M. Mahoney, "Quasi-Monte Carlo feature maps for shift-invariant kernels," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014, pp. 485–493.

[125] Z. Lu, A. May, K. Liu, A. B. Garakani, D. Guo, A. Bellet, L. Fan, M. Collins, B. Kingsbury, M. Picheny, and F. Sha, "How to scale up kernel methods to be as good as deep neural nets," *CoRR*, vol. abs/1411.4000, 2014.

[126] V. Sindhwani and H. Avron, "High-performance kernel machines with implicit distributed optimization and randomization," in *Technometrics*, 2015.

[127] C.-J. Hsieh, S. Si, and I. S. Dhillon, "Fast prediction for large-scale kernel machines," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 3689–3697.