Graph-based Semi-Supervised Learning in Acoustic Modeling for Automatic Speech Recognition

Yuzong Liu

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2016

Reading Committee:

Katrin Kirchhoff, Chair

Jenq-Neng Hwang

Gina-Anne Levow

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Graph-based Semi-Supervised Learning in Acoustic Modeling for Automatic Speech Recognition

Yuzong Liu

Chair of the Supervisory Committee:
Research Professor Katrin Kirchhoff
Electrical Engineering

Acoustic models require a large amount of training data. However, lots of labor is required to annotate the training data for automatic speech recognition. More importantly, the performance of the acoustic model could degenerate during test time, where the conditions of test data differ from the training data in speaker characteristics, channel and recording environment. To compensate for the deviation between training and test conditions, we investigate a graph-based semi-supervised learning approach to acoustic modeling in automatic speech recognition.

Graph-based semi-supervised learning (SSL) is a widely used semi-supervised learning method in which the labeled data and unlabeled data are jointly represented as a weighted graph, and the information is propagated from the labeled data to the unlabeled data. The key assumption that graph-based SSL makes is that data samples lie on a low dimensional manifold, where samples that are close to each other are expected to have the same class label. More importantly, by exploiting the relationship between training and test samples, graph-based SSL implicitly adapts to the test data.

In this thesis, we address several key challenges in applying graph-based SSL to acoustic modeling. We first investigate and compare several state-of-the-art graph-based SSL algorithms on a benchmark dataset. In addition, we propose novel graph construction methods

that allow graph-based SSL to handle variable-length input features. We next investigate the efficacy of graph-based SSL in context of a fully-fledged DNN-based ASR system. We compare two different integration frameworks for graph-based learning. First, we propose a lattice-based late integration framework that combines graph-based SSL with the DNN-based acoustic modeling and evaluate the framework on continuous word recognition tasks. Second, we propose an early integration framework using neural graph embeddings and compare two different neural graph embedding features that capture the information of the manifold at different levels. The embedding features are used as input to a DNN system and are shown to outperform the conventional acoustic feature inputs on several medium-to-large vocabulary conversational speech recognition tasks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ASR: Automatic speech recognition

CTC: Connectionist temporal classification

DNN: Deep neural network

FMLLR: Feature-based maximum likelihood linear regression

GBL: Graph-based learning

GMM: Gaussian mixture model

RNN: Recurrent neural network

LSTM: Long short-term memory

LVCSR: Large vocabulary conversational speech recognition

MLLR: Maximum likelihood linear regression

MLLT: Maximum likelihood linear transformation

SENONE: Context-dependent HMM state

SGD: Stochastic gradient descent

SSL: Semi-supervised learning

SVB: Small vocabulary Switchboard-I dataset

TANH: Hyperbolic tangent

ACKNOWLEDGMENTS

It has been great journey so far as I write this dissertation. First and foremost, I would like to thank my advisor Katrin Kirchhoff for her guidance in the past few years. Starting back to September in 2011 as a fresh PhD student without any experience and knowledge in speech recognition, she has been guiding me all the way here. It is a privilege to have a supportive, patient advisor with thorough knowledge in the ASR and NLP fields.

I am grateful for my committee members, Professor Jenq-Neng Hwang, Professor Gina-Anne Levow and Professor Meliha Yetisgen for their feedback and suggestion on my thesis proposal and my dissertation. I would also like to thank Professor Jeff Bilmes for the collaboration on the IARPA and NSF projects. He has been an amazingly sharp researcher and it has been a great experience learning from him. I am also grateful to Brenda Larson for being an awesome graduate program advisor.

I would like to thank my internship mentors in the past. It has been a great pleasure working with Björn Hoffmeister and Sri Garimella at Amazon. It has been a rewarding experience to work on challenging problems with many experts in the Amazon speech team. I would also like to thank Shawn Chang at Microsoft, for being a great mentor and friend.

I would like to thank my officemates at UW, of which I had many: from Alex Marin, Bin Zhang, Brian Hutchinson, Wei Wu, I learned from their senior student wisdom and knowledge. I would like to thank Professor Mari Ostendorf and her students, Yi Luan, Aaron Jaech, Vicky Zayats, Hao Fang, Hao Cheng, Ji He, Tran Trang and Nicole Nichols; and the Melodi officemates - Kai Wei, Rishabh Iyer, John Halloran, Chandrashekhar Lavania, Shengjie Wang, Wenruo Bai, Brian Dolhansky, for the time we shared together at UW. I would like to thank my friends at UW: Yishen Wang, Tong Zhang, Xiang Chen, Chuanjia

Xing, De Meng, Scott Wisdom and many more people I have met here. I would like to thank Aaron Jaech and Scott Wisdom for proofreading the initial draft of this dissertation.

I would also like to thank my friends: Xiaochen Tang, Xingguang Qu, Fei Zhou, Nan Jiang, Zhao Tan, some of which have been friends for more than 10 years. I have always cherished the time that we spent together since college, the trips we made and all the encouraging conversations we shared.

I would like to thank my "extended" family here in Seattle: Haoyuan Zhao, Xuan Qin, Evan Zhao, Claire Young, Greg Young and little William and Leila. I am blessed to be part of your family here, and you are probably the biggest reason why Seattle feels home to me.

Finally, I would like to thank my family - my grandfather Benci Yan and grandmother Wangdi Sun, who had been raising me and teaching me in my early childhood. I would like to thank my mother Huanmin Yan and father Zhiyong Liu for their unconditional love and support. In particular, I would love to thank my mother for the sacrifice she made to support our family. Finally, I would love to thank my great fiancée: Lida Lin, who had been supporting me throughout all the ups and downs in the past. I look forward to the great years that lie ahead of us. This dissertation is dedicated to all of you.

DEDICATION

Dedicated to

My grandparents: Benci Yan, Wangdi Sun

My parents: Zhiyong Liu, Huanmin Yan

and my fiancée: Lida Lin

for your love and support.

Chapter 1

INTRODUCTION

1.1 Motivation

The emergence of big data has had a huge impact on machine learning research. With the abundance of training data, researchers can train more reliable models for different learning tasks such as speech recognition, text classification, and object recognition. However, in the research area of automatic speech recognition (ASR), lots of labor is required for manual transcription of speech audio, which is time-consuming, expensive, and error-prone. Even expert human transcribers tend to make mistakes at the phonetic level. The labor is even more demanding when it comes to a low-resource language. While labeled samples are expensive to obtain in speech recognition, we have an unprecedented amount of unlabeled data in many practical applications. For example, we can collect a huge amount of audio data from multiple sources such as daily life conversation, digital assistant, or the World Wide Web.

More importantly, there is a consistent need to perform adaptation on new test data for speech recognition. Speech recognition systems usually suffer from performance degradation on unseen test data because acoustic models cannot generalize well to unseen test data when there is a mismatch between training and test conditions. The mismatch between the training and test data comes from different factors, such as environment variations, speaker variations and channel variations. For example, an acoustic model trained on native English speakers may generalize well to other native English speakers during test time, but could yield worse recognition results on an accented English speaker.

Semi-supervised learning (SSL) is a natural solution to both of these problems, as it leverages both labeled and unlabeled data to achieve a better performance than supervised

learning. Among various semi-supervised learning techniques, graph-based semi-supervised learning [158] has drawn increasing attention for different reasons: 1) the graph is a unified representation for many different data types such as images, speech sounds, and texts; 2) most graph-based SSL methods can be characterized as optimizing an energy function, and this objective is usually convex which guarantees global optima.

In graph-based SSL, labeled data and unlabeled data are represented as nodes on a weighted graph where the weights of edges reflect the similarity between nodes. The information of the labeled nodes is "propagated" via the graph to infer the labels of unlabeled data. Graph-based SSL is distinct from any other SSL methods as it utilizes *not only similarities between training and test data, but also exploits similarities between different test data*. Graph-based SSL methods are based on the smoothness assumption on manifolds: the data points are assumed to have a lower-dimensional manifold embedding and the label assignment must respect the inherent clustering properties expressed by the manifold. This means the highly similar data points are encouraged to receive the same labels and are closer on the manifold, whereas dissimilar data points are more likely to receive different labels and are further apart on the manifold.

As shown in previous work [64, 65, 29], speech sounds live on a lower-dimensional manifold of the high dimensional space of all possible sounds. Motivated by this evidence, we investigate graph-based SSL methods for acoustic modeling in a fully-fledged ASR system.

1.2 Contributions of the Thesis

To integrate graph-based SSL into acoustic modeling, we need to address several main challenges. In this thesis, we make the following contributions in addressing these challenges:

1. **What is the best graph-based SSL algorithm for acoustic modeling?**

   In the past, different graph-based SSL algorithms have been proposed, some of which have been applied to several very small-scale speech classification tasks. It is hard to assess which algorithm is most preferable, as there was no comparison using the

same experimental setup and same benchmark datasets. In this thesis, we propose an extension to a graph-based semi-supervised learning algorithm based on entropy regularization [130]. We explicitly use prior regularization in the learning objective, and we show that using the prior regularization significantly improves the phonetic classification tasks compared to other state-of-the-art graph-based SSL algorithms on TIMIT benchmark dataset.

2. **How to construct a good graph for graph-based SSL algorithm?**

   As pointed out in a seminal work [159], the importance of the graph quality far exceeds the learning algorithm and constructing a good graph is 'more art than science'. Even with a 'perfect' graph-based SSL algorithm, inference on a poorly constructed graph can lead to incorrect predictions. The graph quality hinges on various aspects, which include the definition of a similarity measure for the graph weights and a robust feature representation of the speech signals. In this thesis, we study different feature representations that are particularly effective in DNN-based acoustic modeling. We compared and evaluated several feature representations in the context of DNN-based ASR systems. Experimental results show significant improvement in frame-level accuracy, and a consistent word error rate (WER) reduction compared to standard acoustic features. In addition, we propose a novel graph construction method that allows graph-based SSL to handle variable length input vectors. This is especially important because the speech units (i.e. phoneme, syllables, word) consist of variable length by nature. The proposed methods are also applicable to other generic machine learning tasks where the input vectors can be variable-length (such as natural language processing, computational biology, and computer vision).

3. **How to integrate graph-based SSL into a fully-fledged ASR system?**

   In previous work, graph-based SSL methods have shown promising results on simple acoustic modeling tasks such as TIMIT phoneme classification. However, there was no

framework for, or experimental results on, integrating GBL into a fully-fledged ASR system. In addition, all the previous works used Gaussian mixture models (GMMs) as the baseline acoustic models. Since GMMs have largely been replaced by deep neural networks, it is worth exploring whether graph-based SSL can further boost the performance of DNN-based acoustic models. More importantly, we need to investigate how graph-based SSL can improve the word error rate on a realistic conversational speech dataset. In this thesis, we propose a lattice-based framework which systematically integrates the graph-based SSL method into a fully-fledged DNN-based ASR system. We show that the proposed framework results in significant improvements in frame-level classification accuracy as well as consistent reductions in word error rates in continuous word recognition tasks.

4. **How to scale up graph-based SSL methods to large vocabulary conversational speech recognition (LVCSR) systems?**

   To apply graph-based SSL in LVCSR systems, we need to address several characteristics of graph-based SSL which hamper the application of graph-based SSL methods to acoustic modeling for LVCSR tasks. When working on acoustic modeling for LVCSR tasks, the amount of training samples (frames) is usually in the millions or billions. For example, the popular LVCSR benchmark dataset Switchboard has 120 million frames in total, which is equivalent to 309 hours of speech. As a result, the graph consists of more than 120 million nodes, which can be computationally prohibitive to construct. Even with the graph, learning and inference on such large graphs can be extremely time-consuming. This makes standard graph-based SSL impractical for real-time applications, since for each test set a new graph has to be constructed. In addition, when optimizing the graph-based SSL component, we do not take other factors such as language modeling, pronunciations, and the decoder, into consideration. It is often observed that improving one component alone does not help reduce the WER. In this thesis, we propose a novel neural graph embedding method for acoustic modeling and

adaptation. Instead of constructing a similarity graph over a large amount of samples, we encode the information expressed by the graph into a continuous feature space. The original acoustic features, augmented with the learned embedding features, are used as inputs to the DNN model. We show several schemes of encoding the graph information at different level: 1) encoding local neighborhood information; 2) encoding global manifold information. We show statistically significant improvements on LVCSR tasks using the different state-of-the-art deep architectures for ASR systems. More importantly, the proposed approach scales well to very large datasets and achieves real-time performance compared to the lattice-based integration framework.

1.3 Outlines

The rest of the thesis is organized as follows:

In Chapter 2, we review the background of the thesis. We first describe the fundamentals of automatic speech recognition. Then we describe the background on semi-supervised learning and various SSL methods in practice. In particular, we give an overview of graph-based semi-supervised learning and its motivation, as well as the mathematical formulations of different graph-based SSL methods.

In Chapter 3, we first describe several widely used techniques of semi-supervised learning in acoustic modeling. We propose an extension to measure propagation [130], and refer to it as the 'prior-regularized measure propagation (pMP)'. We will describe its formulation as well as its optimization.

In Chapter 4, we study how different graph-based SSL algorithms perform on phoneme classification. We compare the pMP algorithm to several state-of-the-art graph-based SSL methods on a benchmark frame-level phonetic classification task. Then we show how to extend the graph-based SSL to variable-length input vectors and show experimental results on segment-level phonetic classification task. To address the computational cost in segment-level classification, we propose a novel feature selection algorithm based on submodular function optimization.

In Chapter 5, we show how graph-based SSL algorithm can be applied to a fully-fledged DNN-based ASR system. We propose a novel framework based on lattice rescoring, and describe several graph construction techniques that improve the quality of the graph, and consequently, the inference result on the graph. We show experimental results on phone recognition as well as several small-to-medium vocabulary word recognition tasks.

In Chapter 6, we describe a novel neural graph embedding method that learns a compact representation from a similarity graph. The original acoustic features, augmented with the graph embedding features, are used as inputs to the DNN model. We show that the proposed method not only reduces the computational cost by orders of magnitude, but also achieves significant improvements in WER compared to the standard graph-based SSL paradigm on LVCSR tasks using different state-of-the-art ASR systems.

In Chapter 7, we conclude the thesis by summarizing the main contributions of the thesis. We also point out several potential future directions for research on graph-based learning in acoustic modeling.

Chapter 2
BACKGROUND

In this chapter, we describe the background of this thesis: automatic speech recognition (ASR) and semi-supervised learning (SSL). In Section 2.1, we introduce the fundamentals of ASR, which includes different components in an ASR system and different acoustic modeling techniques. In Section 2.2, we introduce the concepts of semi-supervised learning and describe different semi-supervised learning techniques as well as their motivations and assumptions. In Section 2.3, we focus on graph-based semi-supervised learning, a popular SSL method based on manifold assumption. We describe the manifold assumption, mathematical formulation, and graph construction techniques of graph-based SSL algorithms.

2.1 An Overview of Automatic Speech Recognition

The goal of automatic speech recognition (ASR) is to take an input speech signal and convert it to a word sequence. ASR technology has been widely used in various applications, including voice search, virtual assistant, telephony and dictation.

From a statistical point of view, the fundamental problem of speech recognition can be described as follows [67]: given a sequence of speech observations (frames) $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\}$, the goal of ASR is to predict the most likely word sequence $W^* = \{w_1, w_2, \cdots, w_m\}$ by :

$$W^* = \underset{W}{\operatorname{argmax}}\, p(W|X) \tag{2.1}$$

where W denotes a word sequence. With Bayes' Rule, the equation can be expanded as follows:

$$P(W|X) = \frac{p(X|W)P(W)}{p(X)} \tag{2.2}$$

$$\propto p(X|W)P(W) \tag{2.3}$$

Here, the speech signal is first processed by front-end preprocessing such as Mel-frequency cepstral coefficients (MFCCs) and perceptual linear predictions (PLPs) extraction and is represented as a sequence of acoustic feature vectors $\mathbf{x}_t \in \mathbb{R}^d$. The term $p(X|W)$ in Equation 2.2 is the **acoustic likelihood** of an acoustic feature sequence X being generated by a word sequence W, and is computed by an **acoustic model**; the term $p(W)$ is the prior distribution of a word sequence W and is usually given by a **language model**. A commonly used language model is the N-gram language model. In a N-gram language model, the probability of a word w_i in W is conditioned on its previous $N-1$ words. Thus, we can compute the prior of a word sequence W as follows:

$$p(W) = p(w_1, w_2, \cdots, w_n) \tag{2.4}$$

$$= \prod_{i=1}^{n} p(w_i|w_{i-1} \cdots w_{i-N+1}) \tag{2.5}$$

More recently, neural network based language models have shown significant improvements over the conventional N-gram language model [9, 98], and have been widely used in different natural language processing tasks.

To compute the acoustic likelihood $p(X|W)$, we usually decompose the word sequences W into smaller linguistic units Y such as phonemes, syllables, or context-dependent phones. Thus, the original term $p(X|W)$ can be computed as follows:

$$p(X|W) = \sum_{Y} p(X, Y|W) \tag{2.6}$$

$$= \sum_{Y} p(X|Y)p(Y|W) \tag{2.7}$$

The acoustic model is used to calculate $p(X|Y)$ and the **pronunciation model** is used to calculate $p(Y|W)$. The pronunciation model is usually determined by a **dictionary**, or **lexicon**. The acoustic likelihood is computed by a hidden Markov model (HMM) based system and will be described in the following section. To predict the most likelihood word sequence W^*, the **decoder** searches through all possible word sequences and returns the most likely paths. Figure 2.1 shows a schematic depiction of an ASR system.

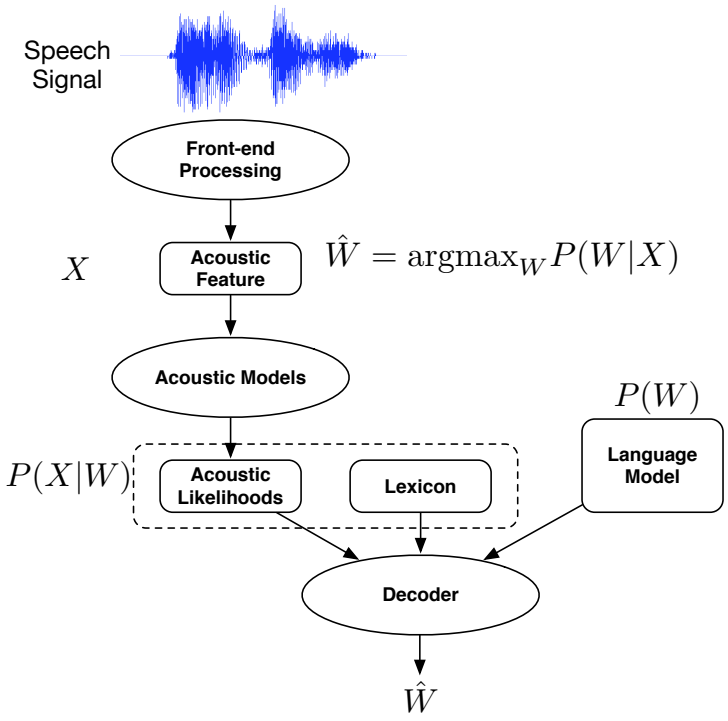


Figure 2.1: An illustration of automatic speech recognition system.

2.1.1 Hidden Markov Models in ASR

The key component in an ASR system is the hidden Markov model (HMM). The hidden Markov model dates back to the 1950s and 1960s, and was first used in speech recognition in the early 1970s. In HMM-based speech recognition systems, the acoustic feature sequence (or observation sequence) $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\}$ is assumed to be generated by a hidden state sequence $Y = \{y_1, y_2, \cdots, y_T\}$, where y_t comes from a finite set of states $\mathcal{S}$. There are two key assumptions in a HMM. First, each hidden state y_t is *only* conditioned on its previous state y_{t-1}, i.e. $p(y_t|y_{t-1}, \cdots, y_1) = p(y_t|y_{t-1})$. This is often referred to as the **Markov assumption**. Second, each observation $\mathbf{x}_t$ is assumed to be generated *only* by hidden state y_t. Thus, an acoustic observation $\mathbf{x}_t$ is conditionally independent of all other observations given the state that generated it, i.e. $p(\mathbf{x}_t|y_t, y_{t-1}, \cdots, y_1, \mathbf{x}_{t-1}, \cdots, \mathbf{x}_1) = p(\mathbf{x}_t|y_t)$. This is

often referred to as the **conditional independence assumption**.

An HMM $\mathcal{M}$ is parameterized by the following components [1]:

- $\mathcal{S} = \{s_1, \cdots, s_N\}$: a finite number of states. In speech recognition, these states are small phonetic units such as phonemes, monophone states, or more often context-dependent phone states [78] (which is often referred to as **senones**);

- $\mathbf{A} = \{a_{11}, a_{12}, \cdots, a_{n1}, a_{nn}\}$: the transition probability matrix $\mathbf{A}$, where each element a_{ij} is the transition probability from state i to state j and $\sum_j a_{ij} = 1$;

- $B = b_j(\mathbf{x}_t)$: acoustic likelihood or emission probability, which measures the probability of an observation $\mathbf{x}_t$ being generated from a state j.

As explained in a seminal work [115], there are three fundamental problems related to HMMs:

1. Compute the likelihood of an observation sequence $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\}$ in an HMM;

2. Discover the best hidden state sequence $Y^* = \{y_1, y_2, \cdots, y_T\}$ that can best explain the observation sequence X;

3. Determine the best parameters in an HMM.

To compute the likelihood of an observation sequence $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\}$ given an HMM $\mathcal{M}$, the **forward algorithm** is used to compute the likelihoods:

$$p(X; \mathcal{M}) = \sum_{Y} \prod_{t=1}^{T} b_{y_t}(\mathbf{x}_t) a_{y_t y_{t+1}} \tag{2.8}$$

where Y is the set of all possible hidden state sequences associated with that observation sequence.

---

[1] In many textbooks, the observation sequence is represented as O and the hidden state sequence is represented as Q.

Finding the hidden state sequence $Y = \{y_1, y_2, \cdots, y_T\}$ that can best explain the observation sequence X is often referred to as the decoding problem in an HMM. The **Viterbi algorithm** [47] with backtrace is used to compute the best path.

Finding the best parameters in an HMM given the training data is often referred to as the training problem in an HMM. The standard training algorithm for HMM is the **Baum-Welch algorithm** [7], a special version of the EM algorithm.

2.1.2 Acoustic Modeling in ASR

In an HMM-based ASR system, we need to compute the acoustic likelihood of an observation $\mathbf{x}_t$ being generated from a state j. A commonly used acoustic model to compute the likelihood is the **Gaussian mixture model (GMM)**. In a GMM-based acoustic model, it is assumed that each observation $\mathbf{x}_t$ is generated by a mixture of Gaussians. The output likelihood for an observation $\mathbf{x}_t \in \mathbb{R}^d$ being generated by state j is given as follows:

$$b_j(\mathbf{x}_t) = \sum_{m=1}^{M} w_m \mathcal{N}(\mathbf{x}_t; \mu_{jm}, \mathbf{\Sigma}_{jm}) \tag{2.9}$$

where

$$\mathcal{N}(\mathbf{x}_t; \mu_{jm}, \mathbf{\Sigma}_{jm}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_{jm}|}} \exp(-\frac{1}{2}(\mathbf{x}_t - \mu_{jm})^\intercal \Sigma^{-1}(\mathbf{x}_t - \mu_{jm})) \tag{2.10}$$

Here M is the number of components in the mixture model, w_m is a positive weight for the m-th component and $\sum_{m=1}^{M} w_i = 1$, μ_m and $\mathbf{\Sigma}_m$ are the mean vector and covariance matrix of the m-th component, respectively.

The GMM-based acoustic model has played a major role in speech recognition since the 1970s. Combined with the context-dependent phone states [78], the GMM-HMM system has achieved considerable success in the past decades. GMM-based acoustic models are also attractive because of the well-defined algorithms for speaker adaptation. Maximum likelihood linear transformation (MLLR) [79] and feature-based MLLR (fMLLR) [33] are well known adaptation techniques for GMM-based models.

Although the GMM is a generative model, previous studies showed that discriminative

training of the GMM-based model can yield far superior performance. Examples of discriminative GMM training include maximum mutual information (MMI) estimation/ boosted MMI (bMMI) [112], minimum classification error (MCE) [70], minimum phone/word error (MPE/MWE) [113] and minimum Bayes risk (MBR) [38].

Deep Neural Networks

DNN-based acoustic models have achieved drastic improvements on ASR in recent years. It is reported by different research groups that the DNN-based acoustic model outperforms the GMM-based system in different large vocabulary conversational speech recognition (LVCSR) by a large margin [53, 23, 121]. Neural networks are inspired by biological neural networks. The networks consist of interconnected nodes that mimic the behavior of neurons, and the information is propagated through the network. Each neuron takes in a linear combination of the inputs, followed by a nonlinear activation function. Figure 2.2 shows a depiction of a simple neuron. Given an input $\mathbf{x} \in \mathbb{R}^d$, the neuron produces an activation output as follows:

$$h_{\mathbf{w},b} = f(\mathbf{w}^\intercal \mathbf{x} + b) = f(\sum_{i=1}^{d} w_i x_i + b) \tag{2.11}$$

where x_i is the i-th dimension of $\mathbf{x}$, f is a nonlinear function, w_i is the weight from x_i to the neuron and b is a bias term. In acoustic modeling, the nonlinear activation function f usually takes one of the following forms:

1. Sigmoid function: $f(z) = \frac{1}{1+\exp(-z)}$;

2. Hyperbolic tangent (tanh) function: $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$;

3. Rectified linear unit (ReLU) function: $f(z) = \max(0, z)$.

A deep neural network consists of multiple layers with an input layer, an output layer and multiple hidden layers, where each layer consists of multiple neurons. Figure 2.3 shows an example of a 4-layer feedforward deep neural network. A feedforward neural network is one

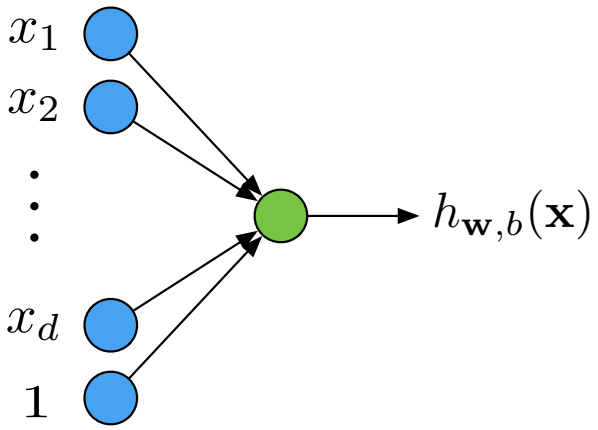


Figure 2.2: An example of a single unit (neuron) in a deep neural network.

neural network where the input information is passed through the network layer by layer, with no feedback or loops in the network. An alternative to a feedforward neural network is the recurrent neural network, which will be described in the following paragraph. For acoustic modeling, the input layer consists of acoustic features, such as MFCCs, or fMLLR features. Usually, the acoustic feature vector is concatenated with the vectors for its preceding and succeeding frames. The output layer corresponds to target classes. In the context of acoustic modeling, they are usually phonemes, or more often senones. The network is parameterized by a weight matrix $\mathbf{W}^l$ for each layer l. Each weight w_{ji}^l is associated with a unit i in the l-th layer, and another unit j in the $(l+1)$-th layer. The output layer consists of K nodes, where K is the number of classes. Each unit in the output layer produces a posterior probability $p(s_k|\mathbf{x}_t)$ for class s_k. These probabilities are obtained by transforming inputs to the neuron with a **softmax** function, given as follows:

$$p(s_k|\mathbf{x}_t) = g(a_k) = \frac{\exp(a_k)}{\sum_{k'=1}^{K} \exp(a_{k'})} \tag{2.12}$$

where s_k is the class label, a_k is the linear combination of inputs from the previous layer to the k-th node in the output layer, K is the total number of classes in the output layer and g is the softmax function. The posterior $p(s_k|\mathbf{x}_t)$ cannot be directly used in the HMM.

To obtain the likelihoods, the posterior is divided by the class prior $p(s_k)$, and we have the following likelihood:

$$p(\mathbf{x}_t|s_k) \propto p(s_k|\mathbf{x}_t)/p(s_k) \tag{2.13}$$

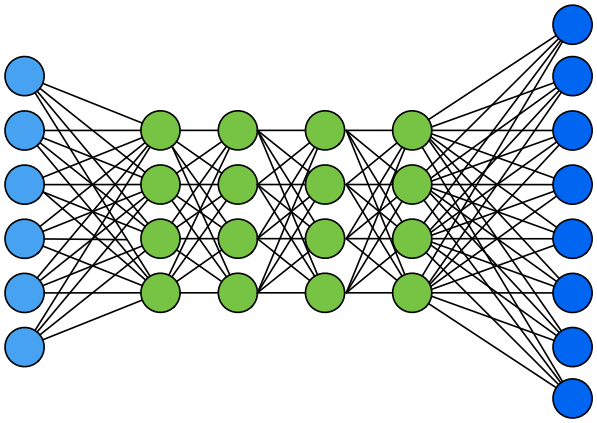


Figure 2.3: An example of a 4-layer feedforward deep neural network.

To train the weights in a DNN, we need to define an objective function that measures the training error of the network. In acoustic modeling, a popular choice of the error function is the cross-entropy between predicted and target outputs, given as follows:

$$CE = -\sum_{t=1}^{T} \log p(y_t|\mathbf{x}_t) \tag{2.14}$$

where y_t is target label at time t (derived from forced alignment), and $p(y_t|\mathbf{x}_t)$ is the posterior obtained from the DNN output layer. To train the DNN, the weights are optimized by minimizing the cross entropy objective function using gradient descent. The gradients for weights in a DNN can be calculated using chain rules with back-propagation. In current DNN training, the networks are usually trained with stochastic gradient descent (SGD) using mini-batches. Training a DNN can often result in local optima because the objective is not convex. As a result, the initialization of the weights in the network is critical. The training of DNNs can be generalized categorized into two approaches:

1. Layer-wise pre-training using Restricted Boltzmann Machines (RBM) [54];

2. Greedy layer-wise supervised training [10].

Recurrent Neural Networks and Long Short Term Memory (LSTM)

A major drawback of feedforward DNNs is that the DNN does not remember the previous state in the network. A recurrent neural network (RNN) overcomes this issue by allowing feedbacks (loops) in the network, thus allowing information to persist. A simple RNN is shown in Figure 2.4, where $\mathbf{x}_t$ is the input at time t, $\mathbf{y}_t$ is the output, and $\mathbf{h}_t$ is the hidden layer output. Note that there is a loop that passes information from the previous step to the next. Thus, at time t, the nodes in the hidden layer take in both current input $\mathbf{x}_t$, and its previous state $\mathbf{h}_{t-1}$. With this feedback loop, RNN can use previous states of the network to make predictions. Similar to the DNN formulation, a simple RNN can be formulated as follows:

$$\mathbf{h}_t = f(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{2.15}$$

where $\mathbf{W}_{hx}$ is a weight matrix between input and hidden layer, $\mathbf{W}_{hh}$ is a weight matrix between current state of the hidden layer and its previous state, $\mathbf{b}_h$ is a bias term and f is element-wise non-linear activation function. The output layer of the RNN uses an element-wise softmax function g:

$$\mathbf{y}_t = g(\mathbf{W}_{yh}\mathbf{h}_t + \mathbf{b}_y) \tag{2.16}$$

where $\mathbf{W}_{yh}$ is a weight matrix between hidden layer and output layer, and $\mathbf{b}_y$ is a bias term. The training of the RNN network can be done by first unfolding the network multiple times, and then applying the standard back-propagation. This method is often referred to as **back-propagation through time (BPTT)** [145].

In a standard RNN, the output at time t depends on inputs and hidden layer outputs from time 0 to $t-1$ and does not use information after time t. To utilize information from the entire sequence, a **bidirectional RNN** (BRNN) is used to process input sequences in both

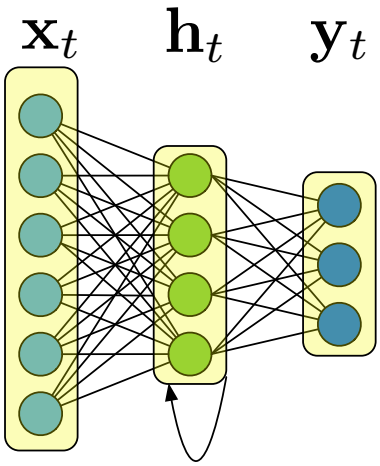


Figure 2.4: An example of a simple recurrent neural network.

forward and backward directions with two separate sub-layers. A BRNN usually outperforms a standard uni-directional RNN.

Training a RNN can be difficult because of the vanishing gradient (or exploding gradient in rare occasions) [55]. Long short term memory networks (LSTMs) [56] were introduced to overcome this problem. LSTM is an improved RNN, with a special memory block in the recurrent hidden layer. Each memory block consists of multiple cells, which replace conventional nodes in the DNNs. Each cell contains a node with a self-loop which is used to store temporal information. Each cell also consists of multiple gates that control the flow of information. A **forget gate** is used to decide what information to discard from the cell. An **input gate** is used to decide what new information to inject to the cell. An **output gate** is used to decide what information to emit. Combining bidirectional RNNs with LSTMs, the model leverages long-range context and achieves considerable improvements over a DNN-based acoustic model [42, 119].

Figure 2.5 shows a depiction of a long short-term memory (LSTM) network. In short,

the LSTM consists of the following functions:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{W}_{xc}\mathbf{x}_t + \mathbf{b}_c) \quad (2.17)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \quad (2.18)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f) \quad (2.19)$$

$$\mathbf{x}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o) \quad (2.20)$$

$$\mathbf{h}_t = \mathbf{x}_t \odot \tanh(\mathbf{c}_t) \quad (2.21)$$

In these equations, σ is a element-wise sigmoid function, *tanh* is a element-wise hyperbolic tangent function, $\odot$ is a element-wise multiplication symbol. $\mathbf{i}_t, \mathbf{f}_t, \mathbf{x}_t$ are the value of the input, forget and output gate at time t. $\mathbf{c}_t$ stores the cell information at time t. $\mathbf{W}_{**}$ are the weight matrices between gates and cells, where $\mathbf{W}_{c*}$ is diagonal. A detailed description of LSTM network can be found in [56, 43, 42]. The LSTM network can be stacked into multiple layers, resulting in a deep LSTM (DLSTM) network. The information in memory cells is updated according to Equation 2.17. When $\mathbf{f}_t = 0$, the cell completely forgets its previous state; when $\mathbf{f}_t = 1$, the previous state remains in the cell. Similarly, when $\mathbf{i}_t = 1$, new information is injected into the cell; when $\mathbf{i}_t = 0$, no new information is injected. The values of $\mathbf{i}_t, \mathbf{f}_t$ are obtained from two sigmoid functions, defined in Equation 2.18 and Equation 2.19. To get the output information from a cell, Equation 2.21 is used to obtain the hidden layer output. When $\mathbf{x}_t = 0$, the hidden layer is set to 0 and does not emit any output; when $\mathbf{x}_t = 1$, the output information is completely emitted.

2.1.3 Lexicon and Language Model

A language model is a critical component in an ASR system. It is used to compute the the prior probability of word sequences that can be recognized by an ASR system. Formally, a language model predicts the probability of a word sequence $P(W)$, where $W = \{w_1, \cdots, w_T\}$ and w_i is the word in the i-th position of W. To compute this quantity, the probability is

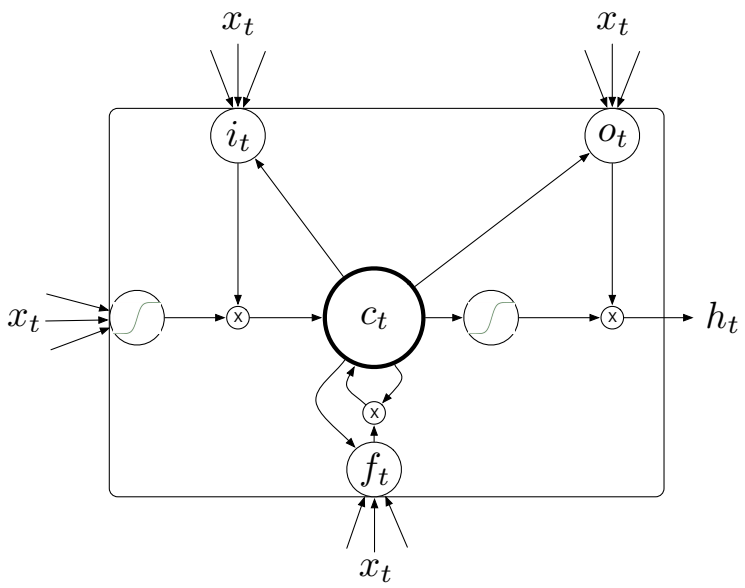


Figure 2.5: An example of a long short-term memory network.

decomposed using the chain rule as follows:

$$p(w_1, \cdots, w_T) = p(w_1) \prod_{i=2}^{T} p(w_i | w_1, \cdots, w_{i-1}) \tag{2.22}$$

In practice, instead of the conditional probability based on the history of words, we usually make an assumption that the conditional probability of a word w_t only depends on its previous $N-1$ words $w_{t-1}, \cdots, w_{t-(N-1)}$. Thus, conditional probability $p(w_t | w_1, \cdots, w_{t-1})$ can be approximated as follows:

$$p(w_t | w_1, \cdots, w_{t-(N-1)}) = p(w_t | w_{t-(N-1)}, \cdots, w_{t-1}) \tag{2.23}$$

This is the most common language model, and it is often referred to as the **N-gram** language model. When $N = 2$, it is referred to as the bigram language model; when $N = 3$, it is referred to as the trigram language model.

The pronunciation model is often represented as a dictionary, which consists of a list of pronunciations for each word. The pronunciations are represented as a phone sequences. In

most LVCSR systems, most words have one pronunciation; sometimes, they may consist of multiple pronunciations.

2.1.4 Search and Decoding

The decoding problem of an ASR system can be formulated as an optimal Bayes classifier, where we want to predict the most likely word sequence W^* as follows:

$$W^* = \text{argmax}_{W \in \mathcal{L}} p(X|W)P(W) \tag{2.24}$$

where $\mathcal{L}$ is a set of candidate output word sequences, $p(X|W)$ is the acoustic likelihoods and $P(W)$ is the language model score. In practice, we usually add a language model scaling factor (LMSF) and rewrite the above equation in log probability as follows:

$$W^* = \text{argmax}_{W \in \mathcal{L}} \log p(X|W) + LMSF \times \log P(W) \tag{2.25}$$

The decoder of an ASR system uses the Viterbi algorithm to decode the input sentence. To find the most probable path, the decoder has to search through all possible word sequences defined by a search graph. The size of the graph usually grows with a more complex model such as a higher-order language model. In a LVCSR system, the size of the graph can be huge. For this purpose, a **multipass decoding** scheme is often used, where a set of initial potential decoding outputs is derived using a simple model (such as simpler acoustic model, and/or simpler language model). A more sophisticated model is then used to re-rank the decoding candidates.

Word lattices are often used in multipass decoding scheme. A word lattice is a compact representation for a set of candidates, which correspond to the highly probable decoding outputs. It is usually generated during first-pass decoding, where the word hypotheses that are active in the beam search at each time point are preserved, while the other paths are pruned away. A word lattice is often represented as a weighted finite state automata (WFSA) $\mathbf{G} = \{V, E, W\}$, where each node $v \in V$ is associated with a time point, and each edge $e \in E$ is connect to two nodes $u, v \in V$, which represents the starting and ending time for a word.

The edges are associated with acoustic likelihood and language model score. Since each word hypothesis in the lattice is associated with its acoustic likelihoods and language model probability, we can either use a more sophisticated language model, or a more complicated acoustic model to do **lattice rescoring**. We can also inject additional scores for lattice rescoring.

2.1.5 Connectionist Temporal Classification for ASR

The conventional DNN-HMM-based (or RNN-HMM-based) ASR system is often referred to as "hybrid system". In these hybrid systems, we need to obtain frame-level labeling as targets for DNN training. To obtain these labels, we need to train different GMM-HMM systems including monophone GMM systems and context-dependent GMM systems. Each of these systems involves different front-end features. Moreover, the training of these systems involve lots of expert knowledge in context-dependent phone decision trees, number of Gaussians, and number of senones.

Recently, end-to-end speech recognition using **Connectionist temporal classification (CTC)** [44, 50, 5, 97] has drawn much attention. CTC uses an objective function [43] for sequence labeling problems involving variable-length input/output sequences that do not have to be of the same length. Unlike hybrid system training, CTC does not rely on frame-level training targets derived from a forced alignment and can automatically learn the alignments between acoustic feature vectors and labels. CTC training also removes the complicated steps such as different GMM-HMM system training, design of context-dependent phone decision trees, which simplifies the ASR training framework.

CTC is used to train a RNN or LSTM network. Formally, given U input sequences $X = \{X^1, \cdots, X^U\}$ and their corresponding label sequences $Z = \{Z^1, \cdots, Z^U\}$, where each input sequence $X^u = \{\mathbf{x}_1^u, \cdots, \mathbf{x}_{T_u}^u\}$ and label sequence $Z^u = \{z_1^u, \cdots, z_{M_u}^u\}$ consist of variable lengths, the goal of CTC is to minimize the negative likelihoods of label Z given

input X defined as follows:

$$-\sum_{i=1}^{U} \log p(Z_i|X_i) \tag{2.26}$$

The labels are defined at the phoneme level, or character level. For English speech recognition, these labels usually consist of 45 phonemes, in addition to a blank symbol which means no label is observed at the output. To train RNNs or LSTMs, the parameters are optimized by SGD using the objective function in Equation 2.26. Note that the length of Z_i is usually smaller than that of X_i. To bridge the length difference, a **CTC path** [43] is used. A CTC path for utterance u is defined as $C^u = \{c_1^u, \cdots, c_{T_u}^u\}$, where T_u is the length of X_u. The CTC path includes blank symbols, and repetitions of different labels, and has the same length as the input sequence. The likelihood $p(Z_i|X_i)$ can be computed as follows:

$$p(Z_i|X_i) = \sum_{C_i \in \Phi_i} p(C_i|X_i) \tag{2.27}$$

where Φ_i is a set that consists of all possible CTC paths. Similar to HMM, Equation 2.27 can be computed on a trellis using forward-backward algorithm. The CTC objective is differentiable to the RNN/LSTM output and gradients can be efficiently computed using back-propagation.

2.2 Semi-supervised Learning

In this section, we give a brief walk-through of different semi-supervised learning methods. There are roughly three types of machine learning paradigms. The first two paradigms are supervised learning and unsupervised learning:

1. **Supervised learning:** In supervised learning, a set of training samples is given $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, where $\mathbf{x}_i$ is the feature vector of a sample i, and y_i is the label associated with the sample. Supervised learning trains a classifier $f : \mathbf{x}_i \rightarrow y_i$ using the training samples, with the goal to predict the labels of unseen test data. To train the classifier, we usually find the best parameter that minimizes the empirical loss function defined on the training set $\mathcal{L}$.

2. **Unsupervised learning:** In unsupervised learning, a set of samples is given $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$ *without any supervision information* (i.e. label information). The goal is to cluster similar samples into groups, or perform dimensionality reduction that preserves the original geometric information. Examples of unsupervised learning include image segmentation, k-means clustering, and principle component analysis.

Semi-supervised learning (SSL) is a machine learning paradigm between supervised learning and unsupervised learning. In SSL, samples from both labeled and unlabeled data are modeled jointly. Here we follow the conventional notations by defining two sets for SSL. The first set $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^l$ contains the labeled data with its class information, and the second set $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^n$ contains the unlabeled data, where $n = l + u$. Each data example $\mathbf{x}_i$ lies in a d-dimensional vector space $\mathbb{R}^d$. Depending on the goal, SSL can be categorized into two broad classes:

- **Transductive learning**: the goal is to train a classifier f using both $\mathcal{L}$ and $\mathcal{U}$, and infer the class labels for $\mathcal{U}$.

- **Inductive learning**: the goal of inductive learning is to learn a classifier f using $\mathcal{L}$ and $\mathcal{U}$, and predict the class labels for unseen test data. In other word, inductive learning can be generalized to any other unseen test data, whereas transductive learning can only be applied to one specific test set.

2.2.1 Self-training

The simplest SSL method is the self-training approach. In this approach, the learning process iteratively teaches itself by using the predictions from the previous model. To be precise, the self-training approach starts with the labeled set $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^l$ and unlabeled set $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^n$. An initial model f is trained using only the labeled data using standard supervised learning. The resulting model f is then used to make predictions on $\mathcal{U}$, where the most confident predictions are removed from $\mathcal{U}$ and added to $\mathcal{L}$ together with their

corresponding class predictions. In the next learning cycle, the model f is refined with this augmented set $\mathcal{L}$. This predict-and-augment procedure is the key to self-training. A critical assumption of the self-training method is that the predictions added to the initial labeled set are reliable enough; otherwise in some degenerate situations, adding false predictions to the training set tends to make the model worse. Self-training is a wrapper method, and different learning algorithms can be applied using the self-training framework. Algorithm 1 shows the framework of self-training approach.

Algorithm 1 Self-training approach.

1: Given labeled set $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ and unlabeled set $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^{n}$, a learning rate k
2: **repeat**
3: Train a classifier f using $\mathcal{L}$;
4: Make prediction on $\mathcal{U}$;
5: Add k most confident predictions $\mathcal{U}' \subseteq \mathcal{U}$, augment $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{U}'$ and remove them from $\mathcal{U}$: $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{U}'$;
6: **until** All unlabeled data is consumed or no improvement of f is observed.

2.2.2 Co-training

Co-training, another standard SSL method, is proposed in [15]. Co-training resembles self-training, with a distinct feature of "multiple views". In co-training, it is assumed that the examples $\mathbf{x}$ have two views, $[\mathbf{x}^{(1)}, \mathbf{x}^{(2)}]$. For example, in the web page classification task, the texts in the webpage (i.e. bag-of-word features) can be considered as the first view of the example; the links and other metadata in the HTML files is the second view of the example. It also assumes that each view can be used to make good classification given sufficient training data, and each view is conditionally independent of the other given the class label.

Co-training procedure starts by duplicating $\mathcal{L}$ into two identical training data sets: $\mathcal{L}_1$ and $\mathcal{L}_2$. Iteratively, we train two classifiers: $f^{(1)}$ from $\mathcal{L}_1$ using the first view $\mathbf{x}^{(1)}$ of the examples, and $f^{(2)}$ from $\mathcal{L}_2$ using the second view $\mathbf{x}^{(2)}$. The unlabeled data $\mathcal{U}$ is classified

using $f^{(1)}$ and $f^{(2)}$. The most confident predicted samples $\mathbf{x}$ using the first classifier, along with their predicted labels $f^{(1)}(\mathbf{x})$ are added to $\mathcal{L}_2$; and the most confident predicted samples $\mathbf{x}$ using the second classifier, along with their predicted labels $f^{(2)}(\mathbf{x})$ are added to $\mathcal{L}_1$. The selected examples are then removed from $\mathcal{U}$. Co-training method is also a wrapper method in which different learning algorithms can be applied. Algorithm 2 shows the co-training framework.

Algorithm 2 Co-training approach.

1: Given labeled set $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ and unlabeled set $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^{n}$, a learning rate k
2: **Initialize** $\mathcal{L}_1 = \mathcal{L}_2 = \mathcal{L}$.
3: **repeat**
4: Train classifier $f^{(1)}$ using $\mathcal{L}_1$ and classifier $f^{(2)}$ using $\mathcal{L}_2$;
5: Make prediction on $\mathcal{U}$ using $f^{(1)}$ and $f^{(2)}$ separately;
6: Add k most confident predictions $\mathcal{U}_1' = \{\mathbf{x}, f^{(1)}(\mathbf{x})\}$, and augment $\mathcal{L}_2 \leftarrow \mathcal{L}_2 \cup \mathcal{U}_1'$;
7: Add k most confident predictions $\mathcal{U}_2' = \{\mathbf{x}, f^{(2)}(\mathbf{x})\}$, and augment $\mathcal{L}_1 \leftarrow \mathcal{L}_1 \cup \mathcal{U}_2'$;
8: Remove them from $\mathcal{U}$: $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{U}_1' \setminus \mathcal{U}_2'$;
9: **until** All unlabeled data is consumed or no improvement of f is observed.

2.2.3 SSL for Generative Models

Generative models are trained to maximize the probability of generating training data under the model assumption. Before we explain SSL for generative models, let us first visit generative model training in supervised learning scenario, where we have $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$. In supervised learning, we learn the parameters θ^* of the generative model by maximizing the log-likelihood as follows:

$$\log p(\mathcal{L}|\theta) = \log \prod_{i=1}^{l} p(\mathbf{x}_i, y_i|\theta) \tag{2.28}$$

This is the well-known maximum likelihood estimation (MLE) criterion for training generative models. In the SSL scenario [104, 22], we maximize the log-likelihood on both $\mathcal{L}$ and $\mathcal{U}$.

The objective function can be rewritten as follows:

$$\log p(\mathcal{L}, \mathcal{U}|\theta) = \log \prod_{i=1}^{l} p(\mathbf{x}_i, y_i|\theta) + \log \prod_{i=l+1}^{l+u} p(\mathbf{x}_i|\theta) \quad (2.29)$$

$$= \sum_{i=1}^{l} \log(p(y_i|\theta)p(\mathbf{x}_i|y_i, \theta)) + \sum_{i=l+1}^{l+u} \log p(\mathbf{x}_i|\theta) \quad (2.30)$$

where in addition to the likelihood defined on $\mathcal{L}$, we have a marginal probability defined on $\mathcal{U}$. Unlike the supervised learning case, where the log-likelihood is concave, this objective is not concave. EM algorithm can find a local optimal solution θ.

The key assumption of SSL for generative models is the model correctness. With an incorrect model assumption, the learning process may result in erroneous solutions.

2.2.4 SSL for Discriminative Models: Avoiding Dense Regions

In contrast to the generative models, discriminative models directly work on the conditional probability $p(y|\mathbf{x})$. Support vector machine (SVM) [21] is a well known discriminative model. Given additional unlabeled examples, can we build a better model for support vector machines? In a seminal work [68], **transductive SVM** was proposed. The key idea of a transductive SVM is the low-density separation criterion. In the supervised SVM, the goal is to find a hyperplane in the Reproducing Kernel Hilbert Space such that the hyperplane separates different classes by a large margin. In the transductive SVM, the goal is to use the additional unlabeled data to refine this hyperplane. Intuitively, the unlabeled data guides the hyperplane to avoid dense regions.

Figure 2.6 shows an example of the intuition of the transductive SVM. The examples in white circles and white squares are the labeled data which correspond to two classes; the remaining examples in black circles are the unlabeled data. In a supervised SVM, the best hyperplane will be the solid line which separates the labeled examples at the maximum margin; with the unlabeled data, however, the best hyperplane becomes the dotted line. Obviously, this decision boundary avoids cutting through the dense region in the space. Given the labeled data $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, the SVM objective can be formulated as minimizing

the hinge loss function defined as follows:

$$\min_{\mathbf{w},b} \sum_{i=1}^{l} [1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)]_+ + \lambda||\mathbf{w}||^2 \tag{2.31}$$

where $[x]_+ = \max(x, 0)$ is the hinge loss function. However, for transductive SVM, the class of the unlabeled data is not known so it is hard to incorporate the information from the unlabeled data directly into the hinge loss function for SVMs. Nevertheless, suppose the classifier makes the right decision on the unlabeled data, we can use the putative labels on $\mathcal{U}$ for the hinge loss function. The modified objective is formulated as follows:

$$\min_{\mathbf{w},b} \sum_{i=1}^{l} [1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)]_+ + \lambda||\mathbf{w}||^2 + \nu \sum_{i=l+1}^{l+u} [1 - |\mathbf{w}^T\mathbf{x}_i + b|]_+ \tag{2.32}$$

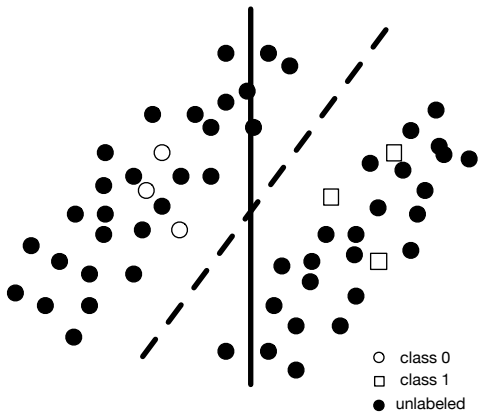


Figure 2.6: Example of transductive SVM. The solid line is the decision boundary using only the labeled data. The dotted line is the decision boundary using both labeled and unlabeled data; this boundary tries to find a low-density separation.

A similar idea was also presented in **entropy regularization** [41], which uses the conditional entropy $H(y|x)$ on the unlabeled data $\mathcal{U}$ for regularization. Although not expressed explicitly in their original work, entropy regularization can be viewed as an analogue of transductive SVM with logistic regression. Recall that logistic regression models the posterior $p(y_i|\mathbf{x}_i)$ by $p(y_i|\mathbf{x}_i) = 1/(1 + \exp(-y_i f(\mathbf{x}_i)))$, where $f(\mathbf{x}_i)$ is a linear model of $\mathbf{x}_i$

parameterized by $\mathbf{w}$ and b: $f(\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i + b$. Training a supervised logistic regression model is equivalent to minimizing the following empirical risk function:

$$\min_{\mathbf{w},b} \sum_{i=1}^{l} \log(1 + \exp(-y_i f(\mathbf{x}_i))) + \lambda||\mathbf{w}||^2 \tag{2.33}$$

As shown in their work [41], the entropy encourages the model to have as great a certainty as possible on the label prediction of $\mathcal{U}$. By incorporating the entropy regularization term into the above function, we have:

$$\min_{\mathbf{w},b} \sum_{i=1}^{l} \log(1 + \exp(-y_i f(\mathbf{x}_i))) + \lambda||\mathbf{w}||^2 + \nu \sum_{i=l+1}^{l+u} H(1/(1 + \exp(-f(\mathbf{x}_i)))) \tag{2.34}$$

where $H(\cdot)$ is the entropy term, and λ, ν are regularization constants.

2.3 Graph-based Semi-supervised Learning

In this section, we describe the background on graph-based SSL, a popular subclass of the SSL methods. In graph-based semi-supervised learning, the data samples are represented as an undirected weighted graph $\mathcal{G} = (V, E, W)$, where V consists of data samples in $\mathcal{L}$ and $\mathcal{U}$, and E are the edges on the graph, weighted by $w \in \mathbf{W}$. The weight captures the similarity between a pair of samples.

The goal of graph-based semi-supervised learning is to infer the class labels on $\mathcal{U}$ by propagating the information from $\mathcal{L}$ on the weighted graph. The inference of labels on the weighted graph can be formulated as minimizing an energy function defined on the graph, and we will describe the formulation in details.

The key assumptions of graph-based SSL are the manifold assumption, smoothness assumption and cluster assumption:

1. **Manifold assumption**: the data has a low-dimensional manifold structure;

2. **Smoothness assumption**: data points that are close to each other are likely to receive the same class label;

3. **Cluster assumption**: data points tend to form clusters, and points in the same cluster are likely to share a label.

Unlike other SSL methods, a distinct feature of graph-based SSL is that it utilizes *not only similarities between training and test data, but also similarities between different test data.*

There are two essential elements in graph-based SSL: 1) graph inference: inferring labels on the graph; 2) graph construction: constructing a graph over all training and test samples. In Section 2.3.1, we describe different inference algorithms and their objective; in Section 2.3.2, we describe different graph construction techniques for graph-based SSL.

2.3.1 Graph-based Semi-supervised Learning Algorithms

Different graph-based semi-supervised learning methods have been proposed in the past. They include label propagation [158, 160], local and global consistency [157], modified adsorption [132], measure propagation [130], quadratic cost criterion [8] and more [51]. Before we describe these graph-based SSL algorithms in detail, we first define some conventional notations that are used in this chapter in Table 2.1.

Label Propagation

Label propagation [158] iteratively propagates the information from the labeled data on a graph according to Algorithm 3. Given a similarity matrix $\mathbf{W}$ constructed over $\mathcal{L}$ and $\mathcal{U}$, we first compute a transition matrix $\mathbf{T}$, where each entry is the probability of jumping from node j to i. Iteratively, the label information is propagated using the transition matrix and the row in $\hat{Y}$ is renormalized after each propagation step. A key property of label propagation is that the first n rows in $\hat{Y}$ have to be reset to Y after each iteration.

Label propagation is also known as Gaussian random fields (GRF), which is proposed in the authors' later work [160]. As one of the first graph-based SSL algorithms, label propagation has a significant impact on subsequent work and is still used as a de facto baseline in many applications. It enjoys many properties such as a closed-form solution and

Symbols	Description
$\mathbf{W}$	Similarity matrix
w_{ij}	Similarity between node i and j
D_{ii}	Degree of node i
$\mathcal{L} = \{(x_i, y_i)\}_{i=1}^{l}$	Labeled Set
$\mathcal{U} = \{(x_i)\}_{i=l+1}^{l+u}$	Unlabeled Set
l	Number of labeled samples
u	Number of unlabeled samples
$n = l + u$	Number of labeled and unlabeled samples
m	Number of Classes
$Y \in \mathbb{R}^{l \times m}$	Seed Label Matrix
$\hat{Y} \in \mathbb{R}^{n \times m}$	Predicted Label Matrix
$\hat{Y}_L \in \mathbb{R}^{l \times m}$	First l rows in $\hat{Y}$ (labeled portion)
$\hat{Y}_U \in \mathbb{R}^{u \times m}$	Last u rows in $\hat{Y}$ (unlabeled portion)
$y_i, \hat{y}_i \in \mathbb{R}^m$	Label distribution of node i
$y_{ik}, \hat{y}_{ik}$	Probability of node i being assigned to class k

Table 2.1: Notations in Graph-based SSL Algorithms

Algorithm 3 Label Propagation Algorithm

1. Construct a similarity matrix $\mathbf{W}$ (self-weight $\leftarrow 0$);
2. Compute the transition matrix $\mathbf{T}$: $T_{ij} = \frac{w_{ij}}{\sum_{k=1}^{n} w_{kj}}$, and initialize the first l rows in $\hat{Y}$ by Y, and the remaining u rows randomly;
3. Label propagation by $\hat{Y} \leftarrow T\hat{Y}$;
4. Normalization for rows in $\hat{Y}$;
5. Fix the labeled portion in $\hat{Y}$ and repeat from step 3 until convergence.

a natural extension to multi-class classification. However, there are several main drawbacks of label propagation:

- The labeled portion in $\hat{Y}$ has to be fixed at every iteration. However it is highly possible that there are incorrectly labeled data points in the training set. This is even true for speech recognition tasks where the labels, such as phoneme or senones, are less reliable. It would be preferable to change the initial label assignment in those cases.

- Label propagation can often lead to degenerate inference in complex scenarios.

Local and Global Consistency (LGC)

In [157], the authors propose another objective "Local and Global Consistency (LGC)" for graph-based SSL. In LGC, the goal is to minimize the following objective function:

$$\sum_{c=1}^{m}[\mu \sum_{i}(\hat{y}_{ic} - y_{ic})^2 + \sum_{i}\sum_{j} w_{ij}(\frac{\hat{y}_{ic}}{\sqrt{D_{ii}}} - \frac{\hat{y}_{jc}}{\sqrt{D_{jj}}})^2] \tag{2.35}$$

The first term in the objective is the supervised loss term and the second term is the manifold smoothness term. The LGC objective overcomes a problematic assumption in label propagation, where the first l rows in $\hat{Y}$ have to be reset to Y. Also, the label value in LGC is normalized by the degree of the node, which regularizes the influence of nodes with higher degrees.

Modified Adsorption

Modified adsorption [132] represents a controlled random walk. Unlike traditional random walks, each node v has three alternative actions: injection, continuation, and abandon, associated with probabilities $p_v^{inj}, p_v^{conj}, p_v^{abdn}$, respectively, where $p_v^{inj} + p_v^{conj} + p_v^{abdn} = 1$. Injection refers to the termination of a random walk and uses prior knowledge about the vertex (labeled points in the training data); continuation refers to the normal continuation of the random walk according to the transition matrix of the graph, and abandon refers to abandoning a random walk and defaulting to a dummy label.

In modified adsorption, the predicted label matrix $\hat{Y}$ is augmented by an additional column. The last column corresponds to a 'dummy' label. In addition, we also define a *default matrix* $\mathbf{R} \in \mathbb{R}^{n \times (m+1)}$. The goal of the modified adsorption algorithm is to minimize the following objective function:

$$\sum_i [\mu_1 \sum_k p_i^{inj} (y_{ik} - \hat{y}_{ik})^2 + \mu_2 \sum_{j \in \mathcal{N}(i)} \sum_k p_i^{cont} w_{ij} (\hat{y}_{ik} - \hat{y}_{jk})^2 + \mu_3 \sum_k p_i^{abdn} (\hat{y}_{ik} - r_{ik})^2] \quad (2.36)$$

This objective can be decomposed into three parts: 1) the output is close to the given supervised knowledge; 2) the manifold should be as smooth as possible (i.e. for samples with a high pairwise similarity value w_{ij}, the label assignments are encouraged to be as similar as possible); 3) the output is a dummy default label if the first two terms are not favored. The loss function in Equation 2.36 can be considered a convex optimization problem that can readily be solved by Jacobi iteration. The values of $p_v^{inj}, p_v^{conj}, p_v^{abdn}$ can be set using grid search, or can be computed by tuning a hyperparameter as described in [132]. Compared to the previous graph-based SSL, the 'dummy' label in modified adsorption allows the user to use an external classifier to inject label information. Thus, modified adsorption can prevent degenerate scenarios.

Measure Propagation

In previous graph-based SSL methods, a quadratic loss function is defined on the label distribution. Measure propagation [130] minimizes a Kullback-Leibler divergence (KLD) based loss instead of the quadratic loss. Here we define two sets of probability distribution $\mathbf{r}_i \in \mathbf{R}^m, i = 1, \cdots, l$, and $\mathbf{p}_i \in \mathbf{R}^m, i = 1, \cdots, n$, where $\mathbf{r}_i$ is the given label distribution of vertex i of the labeled data (or reference), and $\mathbf{p}_i$ is the predicted label distribution. Both $\mathbf{r}_i$ and $\mathbf{p}_i$ live in the m-dimensional probability simplex. Since the labels of training data are already given, $\mathbf{r}_i$ is a 'one-hot' vector with a single entry set to 1 in the given class position and 0 otherwise. The objective function of measure propagation to minimize the following

objective:

$$\sum_{i=1}^{l} D_{KL}(\mathbf{r}_i||\mathbf{p}_i) + \mu \sum_{i=1}^{n} \sum_{j \in \mathcal{N}(i)} w_{ij} D_{KL}(\mathbf{p}_i||\mathbf{p}_j) - \nu \sum_{i=1}^{n} H(\mathbf{p}_i) \tag{2.37}$$

This objective function also consists of three parts: 1) the first term ensures that the predicted probability distribution matches the given distribution on labeled vertices as closely as possible; 2) the second term stands for the smoothness on the manifold (i.e. close points should have a smaller KL divergence); and 3) the third term encourages high-entropy output (if the first two terms are not preferred, the output is then expected to have a uniform distribution). This problem can be solved via interior-point methods but the computation is expensive. In [130], alternating minimization (AM) has been proposed to solve the optimization problem. To see a detailed procedure to derive the update equation and proof of convergence for AM, the reader is to referred to [128]. Measure propagation has many advantages: 1) KL-divergence is a natural loss function for probability distribution, rather than quadratic loss; 2) it allows soft label assignment on $\mathcal{L}$; 3) inference using measure propagation can be parallelized and can handle large-scale datasets. In the past, measure propagation has been used in large-scale dataset such as WebKB [129] and Switchboard [131].

Quadratic Cost (QC) Criterion

In [8], the authors formulate a unified objective of the graph-based SSL by minimizing a quadratic cost function. The objective function is defined as follows:

$$||\hat{Y}_L - Y||^2 + \mu \hat{Y}^T L \hat{Y} + \nu ||\hat{Y}||^2 \tag{2.38}$$

The first term, namely the supervised loss term, penalizes the inconsistency with initial labels and predicted labels of set $\mathcal{L}$. The second term reflects the smoothness on the manifold. The last term in the objective is added to prevent degenerate situations.

The whole optimization can be derived in closed-form

$$\hat{Y} = (\mathbf{S} + \mu L + \nu \mathbf{I})^{-1} \mathbf{S} Y \tag{2.39}$$

where $\mathbf{S}$ is a diagonal matrix, where $S_{ii} = 1, \forall i \leq l$ and $S_{ii} = 0, \forall i > l$. The QC criterion has a big impact on graph-based SSL: it generalizes most graph-based SSL objectives. In fact, label propagation can be formulated as a QC criterion under certain conditions, and is discussed in detail in [8]. The QC criterion also sheds light on explaining the measure propagation algorithm: the first two terms in Equation 2.37 are the supervised loss term and the manifold smoothness penalty term, respectively; as for the third term, minimizing the ℓ_2 norm of the label distribution in the third term in QC has a similar effect to maximizing the entropy of the label distribution in Equation 2.37.

2.3.2 Graph Construction

As stated in [158], the importance of the graph quality far exceeds the learning algorithm, and constructing a good graph is 'more art than science'. Even with a 'perfect' graph-based SSL algorithm, inference on a poorly constructed graph can result in incorrect prediction. A 'good' graph should consist of the following attributes: 1) The weights on the graph should reflect the pairwise similarity. This requires us to have a robust feature representation and a good distance metric. 2) The graph should preserve the manifold information. This requires us to have a good structure of the graphs (such as the degrees of the nodes). In this section, we give a brief overview on graph construction methods in graph-based SSL.

Graph Construction Methods

We first introduce various approaches that result in different graph structures. The most common techniques are listed as follows:

1. **k-nearest-neighbor (kNN) graph**: given a vertex x_i, we select k-nearest neighbors of x_i. The resulting graph has a constant degree on all the vertices, but the graph is asymmetric. This can be resolved by making the kNN graph symmetric: if vertex j is among the k-nearest neighbor to vertex i, we add the vertex i to the nearest neighbors for j. The resulting degree of the vertices on the graph is then at least k, and the

graph is guaranteed to have no isolated component. Efficient exact kNN search, or approximate kNN search, can be done by using data structures such as kd-trees [11].

2. **Mutual k-nearest-neighbor (mkNN) graph**: given a vertex x_i, we preserve the edges only if a vertex x_j is among the k-nearest neighbors to x_i and x_i is also among the k-nearest neighbors to x_j. The resulting graph is guaranteed to be symmetric, but unlike the kNN graph, the degree of the vertices on the mkNN graph is at most k and is usually sparse. To avoid isolated components in the graph, a maximum spanning tree is constructed and the minimum number of edges are added to the graph [107]. The mKNN graphs are usually used to avoid hubs (nodes with very high degrees) in the graph.

3. **ϵ-ball neighbor graph**: given a pre-defined search radius ϵ, we preserve the edges if the weight w is smaller than ϵ. ϵ-ball neighbor graph behaves similarly to the mKNN graph, which is guaranteed to be symmetric, but has no guarantee to avoid isolated components. In practice, the ϵ parameter is sensitive to the graph quality. Thus kNN graph is often preferred over the ϵ-ball neighbor graph and mkNN graph.

4. **b-matching graph**: In [66], the authors propose a graph construction scheme such that each node on the graph has exactly b neighbors. They argue that the kNN graphs are irregular and imbalanced, and the learning algorithms can degrade significantly on such graphs. However, constructing a b-match graph is shown to be computationally expensive. The empirical complexity of a fast b-matching algorithm [57] is $O(n^{2.5})$, which makes the method impractical on large-scale graph.

Edge Weighting

Second, we describe several edge weighting methods for graphs. To compute the similarity between a pair of feature vectors, we can the following similarity measures:

1. **Binary weighting:** The simplest edge weighting method is the binary weight. If there exists an edge between two samples x_i and x_j on the graph, the corresponding weight w_{ij} is set to 1; otherwise, $w_{ij} = 0$.

2. **Weighting with distance metric:** A more conventional similarity measure is to convert distance measures using a radial basis function (RBF) kernel function, which is defined as follows:

$$w_{ij} = \exp(-\frac{d(x_i, x_j)}{2\sigma^2}) \tag{2.40}$$

where $d(x_i, x_j)$ is the distance between a pair of sample x_i and x_j, and σ is the window size of the RBF kernel function which controls the flatness of the kernel function. There are many options for the distance measures:

- **Euclidean distance**: $d_e(x_i, x_j) = ||x_i - x_j||^2$;
- **Mahalanobis distance**: $d_m(x_i, x_j) = (x_i - x_j)^\intercal \Sigma^{-1}(x_i - x_j)$, where $\Sigma \succeq 0$ is a positive semidefinite matrix. Learning a good Mahalanobis distance metric is critical, and has been studied in previous literatures [147, 143].
- **Cosine distance** [30]: $d_c(x_i, x_j) = 1 - \cos(x_i, x_j) = 1 - \frac{x_i \cdot x_j}{||x_i|| ||x_j||}$. The cosine distance is also a widely used distance metric and has been applied in many natural language processing and information retrieval tasks.
- **Kullback-Leibler divergence**: $d_{KL}(x_i, x_j) = \sum_{d=1}^{m} x_i^d \log \frac{x_i^d}{x_j^d}$. The KL divergence is used to calculate the distance between two probability distribution, where x_i and x_j live on a d-dimensional probability simplex.
- **Jenson-Shannon divergence:** $d_{JS}(x_i, x_j) = \frac{d_{KL}(x_i, x_k) + d_{KL}(x_j, x_k)}{2}$. The KL divergence is not a valid metric because of its asymmetric property. Jenson-Shannon divergence is a popular method to compute the distance between two probability distributions. Again, x_i and x_j live on a d-dimensional probabilistic simplex. x_k is the equal-weight interpolation of x_i and x_j, i.e. $x_k^d = (x_i^d + x_j^d)/2$.

3. **Weighting with local reconstruction:** A different edge weighting method is proposed in [138, 24], which is based on the idea of the locally linear embedding method [117]. Instead of defining an explicit distance measure, each example x is considered to be reconstructed from its neighboring examples. In this case, we want to solve the following objective:

$$\min \quad \sum_i ||x_i - \sum_{j \in \mathcal{N}_i} w_{ij} x_j||^2 \tag{2.41}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{N}_i} w_{ij} = 1 \tag{2.42}$$

$$w_{ij} \geq 0 \tag{2.43}$$

where $\mathcal{N}_i$ is the neighboring nodes of example x_i.

Chapter 3

GRAPH-BASED SEMI-SUPERVISED LEARNING ALGORITHM FOR ACOUSTIC MODELING

In this chapter, we propose an extension to measure propagation [130] which is referred to as the **'prior-regularized measure propagation'** (pMP) algorithm for acoustic modeling. First, we describe semi-supervised learning techniques used in acoustic modeling in Section 3.1. Then, we describe the mathematical formulation of the pMP algorithm in Section 3.2. Finally, we describe the optimization procedures for pMP algorithm in Section 3.3.

3.1 Semi-supervised Learning in Acoustic Modeling

Semi-supervised learning techniques for acoustic modeling can be generally categorized into three types:

1. Self-training approaches;

2. Entropy-based approaches;

3. Graph-based approaches.

3.1.1 Self-training in Acoustic Modeling

Although there are many semi-supervised learning methods, as reviewed in the previous chapter, the most dominant semi-supervised learning method for acoustic modeling is self-training. In self-training, an initial "seed" acoustic model is trained using a set of training utterances. After the seed model has been trained, the remaining unlabeled utterances are decoded. The most reliable machine-transcribed utterances are selected according to the decoder confidence score, and added back to the training data set. The augmented training

data is then used to train a refined acoustic model. These steps are repeated for a number of iterations, or until there is no further improvement of the model. The diagram in Figure 3.1 shows an example framework of self-training for acoustic models.

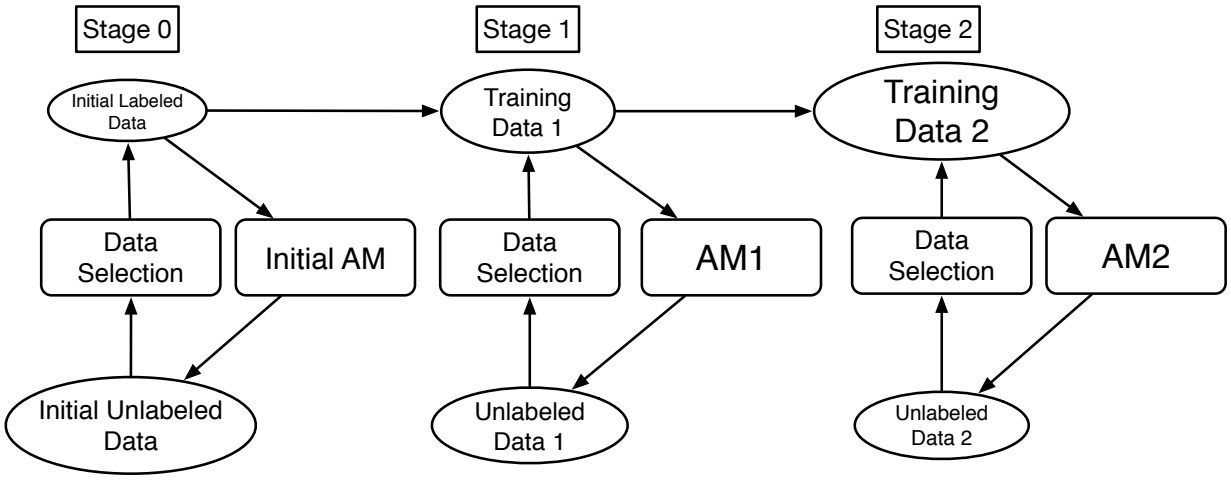


Figure 3.1: Self-training framework for semi-supervised acoustic model training. Given a set of transcribed data, and a set of untranscribed data, an initial acoustic model is trained using only the transcribed data. The model is used to decode the remaining untranscribed data. Data selection method is used to select the most reliable data, which is added back to the original transcribed data. The steps are repeated for a number of iterations, or stopped if there is no further improvement of the system.

These self-training methods have been studied in GMM-based acoustic models [76, 77, 48, 146, 105, 153]. In recent studies [136, 45, 60, 82], self-training methods are also used in DNN-based acoustic model training. The success to self-training hinges on two aspects: first, the "seed" model has to be reasonably good; second, the confidence level and unit selection is important in bootstrapping the system performance in later iterations.

3.1.2 Entropy-based Approaches

Contrary to these bootstrapping approaches, other methods are proposed which jointly models the labeled and unlabeled data. One family of the SSL is motivated from entropy minimization, and has been proposed in [59, 151].

In [151], the authors proposed a unified global entropy reduction maximization (GERM) framework for active learning and semi-supervised learning for acoustic modeling. They showed that both the traditional confidence-based active learning and semi-supervised learning approaches can be improved by maximizing the lattice entropy reduction over the whole dataset.

In [59], the authors proposed to jointly model the labeled and unlabeled data in a conditional entropy minimization framework [41]. In addition to maximizing the posteriors on the labeled data, the authors also maximized the empirical conditional entropy regularization term posed on the unlabeled data. This additional regularizer encourages the model to have as great certainty as possible on the label prediction on the unlabeled data. The optimization of the framework was performed using the extended Baum-Welch algorithm. The method was evaluated in different speech recognition tasks such as phonetic classification and phonetic recognition tasks [58], where improvement was obtained on the TIMIT dataset compared to a supervised discriminatively trained GMM model (with MMI estimation criteria).

3.1.3 Graph-based Approach

Another family of methods involves graph-based semi-supervised learning, which is the main theme of this thesis. In these graph-based methods, it is assumed that samples that are close to each other on the manifold are encouraged to receive the same class label. The goal of graph-based SSL is to infer the class label distributions for all samples in $\mathcal{U}$ defined on the graph G according to an objective function. Figure 3.2 shows an illustration of graph-based approaches for acoustic modeling. In this example, each frame in the labeled and unlabeled data is represented as a node in the graph. The weights on the graph encode the similarity

for a pair of samples. Here we have four phonetic labels /ey/, /ow/, /n/ and /h/. The goal is to infer the label distribution of all samples in $\mathcal{U}$ according to an objective function defined on the graph G. Different objective functions, such as label propagation [158], measure propagation [130], and modified adsorption [132], are defined on the graph.

Note that unlike the self-training and entropy-based approaches, where "unlabeled training data" are used to train a better model, most graph-based approaches are in a *transductive learning* setting. The unlabeled data in graph-based approaches is the test data and the goal of the learning algorithm is to infer labels for a given test set. Whenever a new test set is encountered, we need to repeat the process.

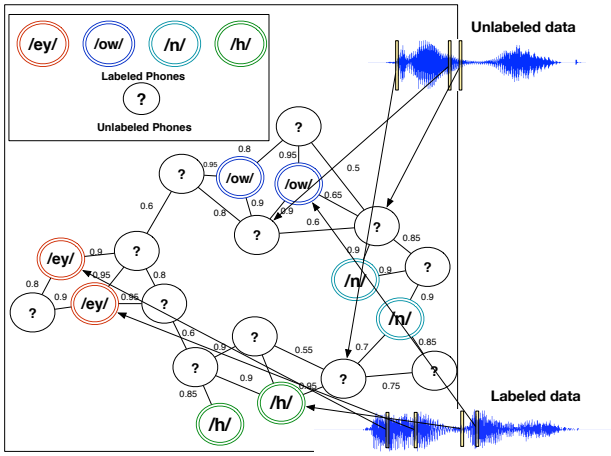


Figure 3.2: An example of graph-based SSL for acoustic modeling. Each node in the graph is associated with a frame in either labeled set $\mathcal{L}$ or unlabeled set $\mathcal{U}$. The weights on the graph indicate the pairwise similarity between two frames. The goal is to infer the label distribution for all samples in $\mathcal{U}$ (nodes with question marks).

In [3], the authors used the label propagation algorithm for vocal classification tasks. The work was evaluated on Voice Joystick dataset [71], a 8-vowel classification task that was used to develop voice-controlled assistive devices for patients with motor impairments. It was shown that the method outperforms a purely supervised GMM and MLP classifier.

In [131], the authors proposed to use measure propagation [130] for Switchboard-I phonetic transcription. The motivation of the project was to annotate the original Switchboard-I dataset at phonetic and syllable levels. Since the human efforts for phonetic annotation is prohibitive, time-consuming, and error-prone, the authors used 75 minutes of annotated speech to infer the phonetic labels of the remaining 320 hours of speech using graph-based SSL. This is the largest graph-based SSL known to date, which consists of 120 million nodes.

In [94], instead of optimizing the label distribution on the graph, the authors added the graph-based manifold regularization term to the multi-layer perceptron (MLP) objective. Recall that the optimal weights θ of a MLP can be obtained by cross entropy training, the authors of [94] added a graph regularization term to the cross entropy objective as follows:

$$J(\theta) = \sum_{i=1}^{L} D_{KL}(\mathbf{t}_i||\mathbf{p}(\mathbf{x}_i;\theta)) + \gamma \sum_{i,j=1}^{n} w_{ij} D_{KL}(\mathbf{p}(\mathbf{x}_i;\theta)||\mathbf{p}(x_j;\theta)) + \beta \sum_{i=1}^{n} D_{KL}(\mathbf{p}(\mathbf{x}_i;\theta)||\mathbf{u}) + \lambda||\theta||_2 \tag{3.1}$$

where $\mathbf{t}_i$ is the target label distribution for sample i, $\mathbf{p}(\mathbf{x}_i;\theta)$ is the label prediction obtained from the MLP output layer, $\mathbf{u}$ is the uniform distribution, $D_{KL}(\cdot,\cdot)$ is the KL divergence between two probability distribution, and $||\theta||_2$ is a ℓ_2 regularization term for the MLP weights. γ, β, λ are the regularization constants. Equation 3.1 is inspired from the measure propagation algorithm and the new objective is similar to the objective in Equation 3.2. The method was evaluated on the frame-level phonetic classification task and showed consistent improvement over the supervised MLP baseline.

In [73], the authors used the modified adsorption algorithm [132] for phonetic classification. Experimental results on the TIMIT dataset showed that the modified adsorption algorithm outperformed the label propagation as well as the supervised MLP baseline system.

Unfortunately, none of these works have addressed the fundamental problem of how graph-based SSL can help the ASR performance: these works neither showed a systematic integration into a fully-fledged ASR system, nor did they show its efficacy on a real continuous word recognition task.

3.2 Prior-Regularized Measure Propagation

Different graph-based objectives have been used for frame-level phonetic classification tasks. However, these graph-based methods are evaluated using different benchmark datasets with different setup. It is hard to gauge which algorithm works best for acoustic modeling in practice. In this section, we describe the prior-regularized measure propagation algorithm - an extension to "measure propagation" [130]. We describe the mathematical formulation of the algorithm as well as the optimization procedures. In the next chapter, we will compare its performance to several other state-of-the-art graph-based SSL methods on the TIMIT benchmark dataset under identical experimental setup.

Symbols	Description
$\mathbf{W}$	Similarity matrix (w_{ij} is the similarity between node i and j)
$\mathcal{L} = \{(x_i, \mathbf{r}_i)\}_{i=1}^{l}$	Labeled Set: l samples in total
$\mathcal{U} = \{(x_i)\}_{i=l+1}^{l+u}$	Unlabeled Set: u samples in total
$\mathbf{r}_i \in \mathbb{R}^m$	Label distribution of node i (m is the number of classes)
$\mathbf{p}_i \in \mathbb{R}^m$	Predicted label distribution of node i
$\tilde{\mathbf{p}}_i \in \mathbb{R}^m$	Prior label distribution of node i
n	Total number of labeled and unlabeled samples
$\mathcal{N}_i$	nearest neighbors of node i

Table 3.1: Notations in prior-regularized measure propagation and measure propagation.

First, we define the notations in Table 3.1. We have a set of labeled data $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{r}_i)\}_{i=1}^{l}$, and a set of unlabeled data $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^{l+u}$. $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional acoustic feature vector for a given sample (frame) i. For acoustic modeling, $\mathbf{x}$ is usually a MFCC feature vector, possibly with additional preprocessing steps such as splicing, LDA or fMLLR transformations, etc. $\mathbf{r}_i \in \mathbb{R}^m$ is the reference distribution over class labels (e.g., phone labels, senones, etc.) for sample i. l and u are the number of frames in the labeled (training) and unlabeled

(test) set, respectively.

We construct a graph $G = \{V, E, W\}$ over all samples from $\mathcal{L}$ and $\mathcal{U}$, where V is the set of nodes in the graph (representing samples in $\mathcal{L} \cup \mathcal{U}$), E is the set of edges that connect a pair of nodes i and j, and each edge is associated with a weight w_{ij} that encodes the similarity between them. The goal is to infer the label distribution $\mathbf{p}_i$ for $i \in \mathcal{U}$.

3.2.1 Objective Functions

To infer the label distribution of all samples in $\mathcal{U}$ on the weighted graph G, we need to design an objective function. Unlike many graph-based SSL algorithms which minimize a quadratic loss function, measure propagation minimizes a Kullback-Leibler divergence (KLD) based loss. In fact, KL divergence is a more natural distance function to measure the distance between label distributions, rather than the Euclidean distance.

As a brief recap, the measure propagation minimizes the following objective defined on a graph G:

$$\begin{aligned} &\sum_{i=1}^{l} D_{KL}(\mathbf{r}_i || \mathbf{p}_i) \\ &+\mu \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i} w_{ij} D_{KL}(\mathbf{p}_i || \mathbf{p}_j) - \nu \sum_{i=1}^{n} H(\mathbf{p}_i) \end{aligned} \tag{3.2}$$

Here, $\mathbf{p}_i$ is the predicted label distribution of all samples in $\mathcal{L} \cup \mathcal{U}$, which we want to optimize. $\mathbf{r}_i$ is reference label distribution of all samples in $\mathcal{L}$, and is usually a one-hot vector (derived from forced alignment). $H(\cdot)$ is the entropy of a probability distribution. $D_{KL}(\cdot, \cdot)$ is the Kullback-Leibler divergence between a pair of probability distribution. w_{ij} is the similarity between sample i and j. $\mathcal{N}_i$ is the set of nearest neighbors of sample i. This objective function consists of three parts: 1) the first term ensures that the predicted probability distribution matches the given distribution on labeled vertices as closely as possible; 2) the second term stands for the smoothness on the manifold (i.e. close points should have a smaller KL divergence); and 3) the third term encourages high-entropy output (if the first

two terms are not preferred, the objective is minimized by assigning a uniform distribution to all of the unlabeled frames).

For the pMP algorithm, instead of maximizing the entropy distribution in the third term in Equation 3.2, we use the prior information as an additional regularization term. Formally, the objective function of the pMP algorithm can be written as:

$$\sum_{i=1}^{l} D_{KL}(\mathbf{r}_i||\mathbf{p}_i) + \mu \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i} w_{ij} D_{KL}(\mathbf{p}_i||\mathbf{p}_j)$$
$$+\nu \sum_{i=1}^{n} D_{KL}(\mathbf{p}_i||\tilde{\mathbf{p}}_i) \tag{3.3}$$

The notation is the same as defined in measure propagation, with an additional set of variables $\{\tilde{\mathbf{p}}_i\}_{i=1}^{n}$, which is the prior label distribution. Instead of encouraging maximizing the entropy of all samples, we penalize the inconsistency between the predicted label distribution and the prior label distribution. We show that adding the prior regularization term can help prevent degenerate scenarios. The objective function can be solved efficiently using alternating minimization (AM), which will be described in the following section.

3.3 Optimization for pMP using AM algorithm

The objective function in Equation 3.3 is convex with respect to $\mathbf{p}_i$. However, there is no closed-form solution. By taking the derivative of the objective function 3.3 with respect to $\mathbf{p}_i$ and setting to zero, we have the following forms: $k_1 p_i(y) \log p_i(y) + k_2 p_i(y) + k_3$, where $p_i(y)$ is the probability of sample i being classified as class y, k_1, k_2, and k_3 are some constants. Consequently, numerical optimization procedures are needed to solve the pMP problem. Method of multipliers and interior methods [16] can be applied to solve the problem.

Similar to measure propagation, we use the alternating minimization (AM) algorithm to solve the optimization. Given an objective function $d(\mathbf{p}, \mathbf{q})$ where variables $\mathbf{p}$ and $\mathbf{q}$ belong to a separate set, AM algorithm finds an optimal pair of variables $(\mathbf{p}, \mathbf{q})$ that minimizes the function. The procedures can be described as the following iterative steps:

- At $t = 0$, choose a random initial point $\mathbf{q}^{(0)} \in \mathcal{Q}$;

- For $t > 0$, repeat the following two steps until convergence:

  - P-Step: Find an optimal projection $\mathbf{p}^{(t)} \in \mathcal{P}$ that minimizes $d(\mathbf{p}^{(t)}, \mathbf{q}^{(t-1)})$;
  - Q-Step: Fix $\mathbf{q}^{(t)}$, find an optimal projection $\mathbf{q}^{(t)} \in \mathcal{Q}$ that minimizes $d(\mathbf{p}^{(t)}, \mathbf{q}^{(t)})$.

In order to have a formulation that is amenable for AM algorithm, a third set of probability distributions $\{\mathbf{q_i}\}_{i=1}^{n}$ is introduced. To be complete, the AM formulation of the pMP problem in Equation 3.3 is rewritten as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{l} D_{KL}(\mathbf{r}_i||\mathbf{q}_i) \\
& +\mu \sum_{i=1}^{n} \sum_{j \in \mathcal{N}'(i)} w'_{ij} D_{KL}(\mathbf{p}_i||\mathbf{q}_j) \\
& +\nu \sum_{i=1}^{n} D_{KL}(\mathbf{p}_i||\tilde{\mathbf{p}}_i) \qquad (3.4) \\
\text{subject to} \quad & \sum_{y} p_i(y) = 1, p_i(y) > 0 \\
& \sum_{y} q_i(y) = 1, q_i(y) > 0 \\
& w'_{ij} = [\mathbf{W}']_{ij} \\
& \mathbf{W}' = \mathbf{W} + \alpha \mathbf{I}_n \\
& \mathcal{N}'(i) = \{i\} \bigcup \mathcal{N}(i), \alpha \geq 0 \qquad (3.5)
\end{aligned}
$$

where $\{\mathbf{p}_i\}_{i=1}^{n}$ and $\{\mathbf{q}_i\}_{i=1}^{n}$ are two sets of probability distributions to optimize. $p_i(y), q_i(y)$ is the probability of sample i being classified as class y. α is hyper-parameter (and the only one) in this formulation, which plays a role in enforcing that $\mathbf{p}_i$ and $\mathbf{q}_i$ are similar. When $\alpha \to \infty$, the AM formulation is the same as the original formulation. Again, the optimization problem in Equation 3.4 is a convex optimization over the variables $\mathbf{p}_i$ and $\mathbf{q}_i$.

3.3.1 Update Equation for pMP using AM algorithm

By writing out the Lagrangian of the objective function in Equation 3.4 and setting the partial derivative w.r.t. p and q to zero, we arrive at a closed-form update equation for p:

$$p_i^{(t)}(y) = \frac{\exp(\frac{\beta_i'(y)}{\alpha_i'})}{\sum_y \exp(\frac{\beta_i'(y)}{\alpha_i'})} \tag{3.6}$$

where

$$\begin{aligned} \alpha_i'(y) &= \mu \sum_j w_{ij}' + \nu \\ \beta_i'(y) &= \mu \sum_j w_{ij}' (\log q_j^{(t-1)}(y) - 1) + \nu(\log \tilde{p}_i(y) - 1) \end{aligned}$$

Similarly, a closed-form update equation for q can be derived as following:

$$q_i^{(t)}(y) = \frac{r_i(y)\delta(i \leq l) + \mu \sum_j w_{ji}' p_j^{(t)}(y)}{\delta(i \leq l) + \mu \sum_j w_{ji}'} \tag{3.7}$$

Chapter 4

GRAPH-BASED SEMI-SUPERVISED LEARNING IN PHONETIC CLASSIFICATION

In this chapter, we describe how to use graph-based semi-supervised learning for phonetic classification tasks [89]. Phonetic classification tasks are relatively artificial but are very informative in evaluating novel acoustic modeling methods. In this chapter, we focus on two different phonetic classification tasks. The first task is the conventional **frame-level phone classification**. In frame-level phone classification, it is assumed that the time boundary information for phone segments is given. Each frame is assigned to a phone label. In semi-supervised learning, we are given a set of labeled data $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, and a set of unlabeled data $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^{l+u}$, where l and u are the number of samples in $\mathcal{L}$ and $\mathcal{U}$, $\mathbf{x}_i \in \mathcal{R}^d$ is a feature vector in d-dimensional space and $y_i \in \{1, \cdots, C\}$ is the phoneme class label, the goal is to infer the class labels for the unlabeled set $\mathcal{U}$. We will evaluate the performance of graph-based SSL in frame-level phone classification in Section 4.1 and Section 4.2. The second task is **segment-level phone classification**. In segment-level classification, each acoustic segment $\mathbf{x} = \{\mathbf{x}_1, \cdots, \mathbf{x}_s\}$ is assigned to a phone label. Since these segments consist of a variable-length sequence of frames, it is challenging to use graph-based SSL, which was developed for fixed-length feature vectors. In Section 4.3, we will propose several novel graph construction methods that allow graph-based SSL to handle variable-length input vectors. In Section 4.4, we will evaluate the proposed methods for segment-level classification. In both these tasks, we compare the pMP algorithm from Chapter 3 to several state-of-the-art graph-based SSL algorithms and show a consistent improvement over other graph-based methods as well as supervised baseline system. We wrap up this chapter in Section 4.5.

4.1 Frame Level Phone Classification

In this section, we investigate graph-based SSL for phonetic classification tasks. In previous work [3, 131, 73, 106], graph-based SSL algorithms have been adopted for phone classification tasks. On one hand, these studies show promising results that the graph-based SSL method can improve over a purely supervised baseline system; on the other hand, it is difficult to gauge which graph-based SSL is most suitable for acoustic modeling tasks. They either use different graph construction methods, or evaluate on different benchmark datasets with different baseline systems. In this thesis, we first evaluate the pMP algorithm against three other state-of-the-art graph-based SSL algorithms: label propagation (LP) [158], measure propagation (MP) [130], and modified adsorption (MAD) [132]. We evaluate the performance on the popular phonetic classification benchmark dataset TIMIT, with identical experimental setups.

4.1.1 Benchmark Dataset

We use the TIMIT dataset [35] to evaluate frame-level and segment-level phone classification. The TIMIT dataset is a popular benchmark dataset in phonetic classification. It was designed by Massachusetts Institute of Technology (MIT), SRI International (SRI) and Texas Instruments, Inc. (TI), and is used for acoustic-phonetic studies. TIMIT contains recordings of read speech from 630 speakers of 8 dialects of American English. The sentences were created to be phonetically rich, and were read by different speakers. The TIMIT dataset provides not only the word-level transcription, but also a careful frame-level phonetic annotation. Thus, phonetic classification using TIMIT is a well-designed task for fast evaluation of novel acoustic modeling methods.

Following the conventional setup, we use 3696 sentences from 462 speakers, excluding all the dialect sentences (*sa* sentences), as training data, 192 sentences (24 speakers) from the core test set as evaluation data. We also use a development set of 210 sentences from 50 speakers [49]. The TIMIT dataset has 61 phoneme labels. For training, we collapse the 61

phone labels into 48 phone labels; for evaluation, we further collapse them into 39 phones. We exclude all the glottal stop segments. Table 4.1 shows the phone labels used in TIMIT. For front-end processing, we extract 39-dimensional feature vectors (consisting of 12 MFCC coefficients, 1 energy coefficient, deltas, and delta-deltas) every 10ms with a window of 25ms. Mean and variance normalization of the feature vectors are applied on a per-speaker basis. The total number of samples for the training, development and test sets, are 1044671, 63679, and 57908, respectively. This corresponds to 121385, 7416, and 6589 phone segments. To simulate different training conditions for semi-supervised learning, we use 10%, 30%, 50% and 100% of the training data, respectively, to investigate the performance of graph-based SSL algorithms on varying amounts of training data.

4.1.2 Graph Construction for Frame-level Classification

The graph can be obtained from a nearest neighbor graph (either kNN, or ϵ-NN) as described in Section 2.3.2. In our specific case for frame-level classification, we adopt the same graph construction method in [39, 3], which is shown in Figure 4.1. To be precise, for each sample in $\mathcal{U}$, we find k_l nearest neighbors in the labeled data $\mathcal{L}$, and k_u nearest neighbors in the unlabeled data $\mathcal{U}$. We also perform a symmetrization on the graph such that if sample i is one of the nearest neighbors of sample j, then we also add j to the nearest neighbors of sample i.

For graph construction, a common approach is to use the Euclidean distance between MFCC feature vectors, and convert it into similarity with an RBF kernel function. In this task, we follow another graph construction approach based on similarity for probability distributions [73]. First, a first-pass supervised classifier is trained on the labeled data set. In our case, we train a 3-layer multi-layer perceptron (MLP), which takes as input a window of 9 acoustic frames. Thus, the dimensionality of the input vector is $39 \times 9 = 361$. We use 2000 hidden units in the hidden layer with a sigmoid nonlinear activation function. The output layer consists of 48 units, which corresponds to 48 phone labels. The output layer uses a softmax function. This MLP also serves as the supervised baseline system. For each training

	Phone Label	Mapping (48/39)		Phone Label	Mapping (48/39)		Phone Label	Mapping (48/39)
1	iy	iy/iy	22	ch	ch/ch	43	en	en/n
2	ih	ih/ih	23	b	b/b	44	eng	ng/ng
3	eh	eh/eh	24	d	d/d	45	l	l/l
4	ey	ey/ey	25	g	g/g	46	r	r/r
5	ae	ae/ae	26	p	p/p	47	w	w/w
6	aa	aa/aa	27	t	t/t	48	y	y/y
7	aw	aw/aw	28	k	k/k	49	hh	hh/hh
8	ay	ay/ay	29	dx	dx/dx	50	hv	hh/hh
9	ah	ah/ah	30	s	s/s	51	el	el/l
10	ao	ao/aa	31	sh	sh/sh	52	bcl	vcl/sil
11	oy	oy/oy	32	z	z/z	53	dcl	vcl/sil
12	ow	ow/ow	33	zh	zh/sh	54	gcl	vcl/sil
13	uh	uh/uh	34	f	f/f	55	pcl	cl/sil
14	uw	uw/uw	35	th	th/th	56	tcl	cl/sil
15	ux	uw/uw	36	v	v/v	57	kcl	cl/sil
16	er	er/er	37	dh	dh/dh	58	q	
17	ax	ax/ah	38	m	m/m	59	pau	sil/sil
18	ix	ix/ih	39	n	n/n	60	epi	epi/sil
19	axr	er/er	40	ng	ng/ng	61	h#	sil/sil
20	ax-h	ax/ah	41	em	m/m			
21	jh	jh/jh	42	nx	n/n			

Table 4.1: TIMIT phone labels.

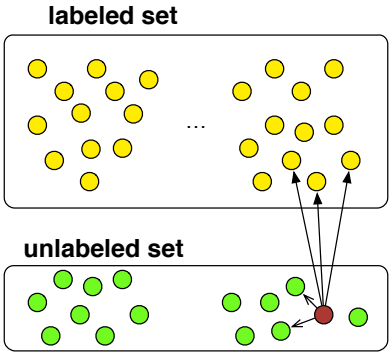


Figure 4.1: kNN graph for acoustic modeling. For each node in the unlabeled set $\mathcal{U}$ (for example, the red node), we find k_l nearest neighbors in the labeled data $\mathcal{L}$, and k_u nearest neighbors in the unlabeled data $\mathcal{U}$.

condition the MLP only receives the same training data as the subsequent graph-based SSL algorithms, i.e. 10%, 30%, 50% or 100%.

After the MLP classifier has been trained, we derive a phone label distribution for each sample in the test data. These probability distributions are used as a new feature representation for graph construction. In our previous work [88], we showed that this graph construction yielded far superior classification performance than using the raw MFCC feature vectors. With a first-pass classifier, we can remove noise from the original acoustic feature space. In addition, we can significantly reduce the training data size. Compared to naive graph construction which requires $n = l + u$ nodes and n^2 similarity computation, we only need $C + u$ nodes in the graph, where u is usually much smaller than l [1] and $C = 48$. Because we are using phone label distribution as the feature space, instead of using all training samples for graph construction, we only need to create 48 canonical ('seed') feature vectors. For each phone class, a one-hot feature vector is created with 1 at the corresponding label and 0 elsewhere.

[1] For TIMIT, $l \approx 10^7$ and $l \approx 10^5$.

To compute the similarity measures on the weighted graph, we can use the Jenson-Shannon divergence as described in Section 2.3.2. As a recap, the Jenson-Shannon divergence for probability distributions is defined as follows:

$$d_{JS}(\mathbf{x}_i, \mathbf{x}_j) = \frac{d_{KL}(\mathbf{x}_i, \mathbf{x}_k) + d_{KL}(\mathbf{x}_j, \mathbf{x}_k)}{2} \tag{4.1}$$

where, $\mathbf{x}_i$ and $\mathbf{x}_j$ live on a d-dimensional probabilistic simplex. $\mathbf{x}_k$ is the equal-weight interpolation of $\mathbf{x}_i$ and $\mathbf{x}_j$, i.e. $\mathbf{x}_k^{[i]} = (\mathbf{x}_i^{[i]} + \mathbf{x}_j^{[i]})/2$, where $\mathbf{x}^{[i]}$ is the ith dimension of the probability distribution. We set $k_l = k_u = 10$ for the nearest neighbor graph construction. A fast nearest neighbor search procedure using kd-trees [11] is used to select the training and test data points within a given radius r around each test point. ($r = 7.0$ for selecting training points, $r = 0.02$ for test points). For MP and pMP, since the Jensen-Shannon divergence between a given unlabeled vertex and all the seeds can be very small (i.e. $< 10^{-10}$), we set the largest weight to 0.1 and scale up the other weights accordingly.

4.2 Experimental Results: Frame Level Classification

We evaluate the pMP algorithms against three other widely used graph-based SSL algorithms: label propagation (LP) [158], measure propagation (MP) [130], and modified adsorption (MAD) [132]. The supervised baseline system on each training condition has an accuracy between 65.94% and 72.45%. Better accuracy can be achieved using a more advanced classifier, such as a deep neural network; however, we concentrate on comparing the pMP algorithms to other graph-based methods under identical setup. We will show how pMP algorithms can improve over a DNN-based system in Chapter 5.

Both MAD and pMP algorithms can utilize prior information. For the pMP algorithm, we use the MLP output as the prior term $\tilde{\mathbf{p}}$ in Equation 3.3. For MAD, the predicted label distribution consists of a 'dummy' label. In case the 'dummy' label receives highest probability, we use the MLP output as the prediction. Table 4.2 shows the frame-level classification accuracies under different training conditions using four different graph-based SSL algorithms. The numbers in bold face indicate a significant improvement over the

supervised baseline system at $p = 0.05$ using z-test. We observe that for each training condition, LP and MP do not have a better classification accuracy compared to the supervised baseline system. On the contrary, the best graph-based SSL algorithms are those that incorporate prior information, namely MAD and pMP. The best results are obtained by pMP; improvements over the supervised baseline range between 1.28% and 1.82% absolute. The pMP algorithm is consistently outperforming MAD, and we assume that this is because pMP uses the prior information in the objective function and *during* label inference, while MAD injects a hard label assignment provided by the prior *after* the label inference.

	Amount of training data			
System	10%	30%	50%	100%
MLP	65.94	69.24	70.84	72.45
LP	65.47	69.24	70.44	71.46
MP	65.48	69.24	70.44	71.46
MAD	**66.53**	**70.25**	71.60	**73.01**
pMP	**67.22**	**71.06**	**72.46**	**73.75**

Table 4.2: Accuracy rates (%) for frame-based phone classification for the baseline (MLP) and various graph-based learners. Bold-face numbers are significant ($p < 0.05$) improvements over the baseline.

4.3 Segment Level Phone Classification

4.3.1 Motivation

Frame-level modeling has been the predominant approach. However, when we apply graph-based SSL to frame-level modeling, the resulting graph is usually very large, consisting of millions of nodes. To alleviate the computational overhead, we can apply graph-based SSL at the segment level. The segments can be extracted from lattices, or confusion networks.

The resulting number of segments is usually much smaller than the number of frames.

In graph-based SSL, each node is associated with a feature vector. In frame classification task, the feature vector is usually a fixed-dimensional MFCC feature vector. In segment classification task, each node represents a variable-length speech segment. In the following sections, we propose several novel graph construction methods that enable the graph-based SSL to handle variable-length input vectors. These methods either transform the variable-length input vectors into fixed-dimensional feature vectors, or directly compute the pairwise similarity between a pair of segments. To be specific, we will describe the following three graph construction approaches:

1. First-pass classifier approach;

2. Rational kernel approach;

3. Fisher kernel approach.

We assume that the time boundaries of the speech segments in the classification task are provided, along with their corresponding phone labels. In a real segment-level ASR system, however, these time boundaries and the labels need to be inferred and are not usually provided a priori.

4.3.2 First-pass Classifier Approach

First-pass classifier approach is already described in graph construction for frame-level classification. In short, each acoustic feature vector $\mathbf{x} \in \mathbb{R}^d$ is mapped into a probabilistic feature space $\mathbf{p} \in \mathbb{R}^m$ using a supervised classifier (such as a neural network), where d and m are the dimensionality of the original feature space and new feature space, respectively. To calculate the similarities in the new feature space, Jenson-Shannon divergence is applied. The same idea can also be used in segment-level classification task. We need to compute the phoneme distribution over a segment. To compute this, for each phone $i \in \{1, \cdots, m\}$, we train a

monophone HMM, where each state is parameterized by a Gaussian mixture model. To compute the label distribution over a phoneme segment, we compute the Viterbi score s_i (represented in log-likelihood) for each phone i. The phone label distribution of a segment $p(i|\mathbf{X})$ is derived by the following equation:

$$p(i|\mathbf{x}) = \frac{s_i}{\sum_{i=1}^{m} s_i} \tag{4.2}$$

Thus, we can map a variable-length speech segment $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ to a fixed-length probabilistic feature space $\mathbb{R}^m$. To compute the pairwise distances for graph construction, Jenson-Shannon divergence is applied. The distances are converted into similarity values using an RBF kernel function. Figure 4.2 shows an illustration of the first-pass classifier approach.

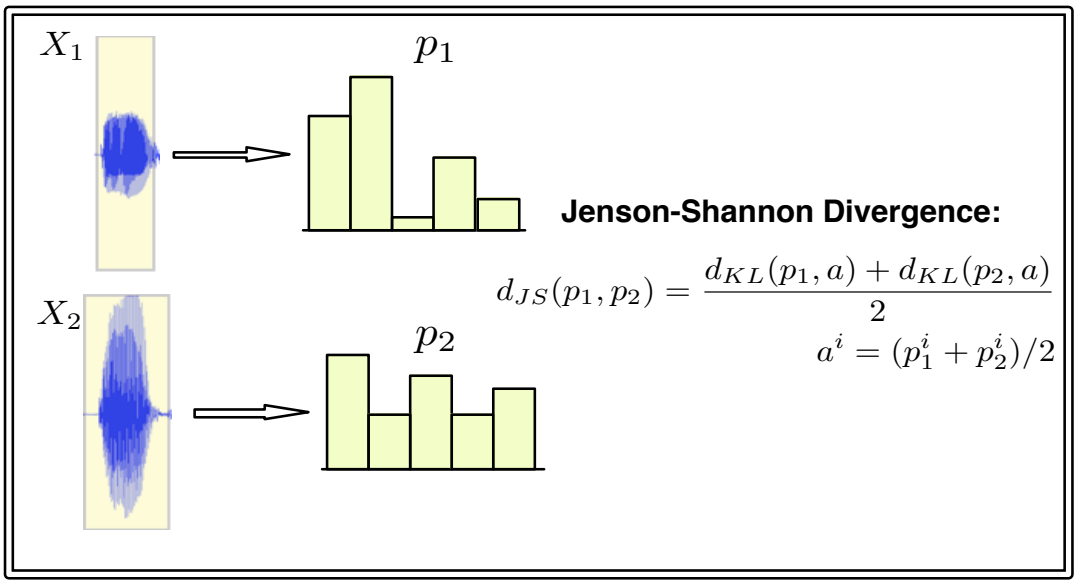


Figure 4.2: First-pass classifier approach for variable-length inputs. Variable inputs are converted to fixed-length posterior distributions using a supervised classifier.

4.3.3 Rational Kernel Approach

An alternative similarity measure is a symbol-based similarity measure. Instead of mapping a variable-length input into a continuous vector space, a symbol-based approach represents the original input segment as a string of symbols. For our purpose, we generate a string of linguistic units (such as phonemes) for each speech segment using a tokenizer. To measure the similarity between a pair of strings, we can use several well-defined string similarity measures. These similarity measures include:

- **Bag-of-Word feature similarity**: In the bag-of-word feature approach, each phone sequence is represented as a vector, with each entry representing the number of occurrences of the corresponding units. Then cosine similarity function is applied to derive the similarity measure between two sequences.

- **TF-IDF weighted N-gram cosine similarity**: The cosine similarity between phone sequences s_i and s_j is computed as:

$$\text{sim}(i,j) = \frac{\sum_{w \in s_i} \text{tf}_{w,s_i} \cdot \text{tf}_{w,s_j} \cdot \text{idf}_w^2}{\sqrt{\sum_{w \in s_i} \text{tf}_{w,s_i}^2 \text{idf}_w^2} \sqrt{\sum_{w \in s_j} \text{tf}_{w,s_j}^2 \text{idf}_w^2}} \tag{4.3}$$

  where tf_{w,s_i} is the count of n-gram w in s_i and idf_w is the inverse document count of w (each sentence is a document).

- **String kernel**: String kernel [116] is a kernel defined on a pair of strings, which can be variable-length. Formally, we define a string s as a concatenation of symbols from a finite alphabet Σ and an embedding function from strings to feature vectors, ϕ: $\Sigma_* \rightarrow H$. The string kernel function $\mathcal{K}(s,t)$ computes the distance between the resulting vectors for two strings s_i and s_j. The embedding function is defined as:

$$\phi_u^k(s) := \sum_{\mathbf{i}:u=s(\mathbf{i})} \lambda^{|\mathbf{i}|}, u \in \Sigma^k \tag{4.4}$$

where k is the maximum length of subsequences, $|\mathbf{i}|$ is the length of i, and λ is a penalty parameter for each gap encountered in the subsequence. $\mathcal{K}$ is defined as

$$\mathcal{K}(s_i, s_j) = \sum_u \langle \phi_u(s_i), \phi_u(s_j) \rangle w_u \tag{4.5}$$

where w is a weight dependent on the length of $u, l(u)$. Finally, the kernel score is normalized by $\mathcal{K}(s_i, s_i) \cdot \mathcal{K}(s_j, s_j)$ to discourage long string from being favored.

- **Rational kernel**: Instead of computing the pairwise similarity between two strings, rational kernels [20] consider sets of strings. They are generalizations of the above kernels, and can be implemented efficiently using a finite-state transducer (FST) framework. For speech this means that rather than using a single sequence of phones or HMM states we can use lattices of symbols for each utterance, and compute the kernel between the lattices. The idea behind rational kernels is intuitive: two sequences, or more generally, two lattices, are similar when they share some common structure. In short, a kernel $\mathcal{K}$ is rational if there exists a weighted finite-state transducer T such that for all weighted automata A and B, the following equation holds:

$$\mathcal{K}(A, B) = \sum_{x,y} [[A]](x) \cdot [[T]](x, y) \cdot [[B]](y) \tag{4.6}$$

where $[[A]](x)$ is the sum of the weights of all successful paths labeled with x; $[[T]](x, y)$ is the sum of the weights of all successful paths with input x and output y. More formally, we recap some algebraic definitions and notations. In abstract algebra, a system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a semiring that may lack negation, where $\oplus, \otimes$ are the binary operations: addition and multiplication. An 8-tuple $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ is a weighted finite-state transducer over a semiring K, where Σ is the finite set of input symbols of the transducers, Δ is the finite set of output symbols of the transducers, Q is the finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, E is the set of transitions, and λ, ρ denotes the weights of initial states and final states, respectively. The above definition can be generalized to the case of an arbitrary semiring for more general operations as follows:

Definition 1. *Let A and B be two acyclic weighted automata over the semiring $\mathbb{K}$, a kernel $\mathcal{K}$ is rational if there exist a weighted transducer T over $\mathbb{K}$ and a function $\phi : \mathbb{K} \to \mathbb{R}$ such that for all A and B:*

$$\mathcal{K}(A, B) = \phi(\bigoplus_{x,y} [[A]](x) \otimes [[T]](x, y) \otimes [[B]](y)) \tag{4.7}$$

A weighted finite-state transducer supports operations such as union, intersection, and composition, which can be found in [20] in details. Rational kernels have been successfully applied in ASR research, such as speech recognition [100], spoken-dialog classification [19] and emotion detection in speech [125]. In graph-based SSL, suppose we are given multiple sequences of phones or HMM state labels for an utterance, we can express each utterance in a weighted finite-state automaton - a special case of weighted finite-state transducer with identical input and output symbols at each transition. To compute the similarity between a pair of sets of sequences for two utterances, we can apply the $\oplus, \otimes$-operation which computes the sum of counts of all matching (sub)sequences on the two sets according to Equation 4.6.

4.3.4 Fisher Kernel Approach via Generative Models

Fisher kernels [63] approach maps a variable-length feature vector into a fixed-length high-dimensional feature space (aka Fisher score vector). Fisher kernels have been used in audio classification [101], speaker verification [137] and image classification [109]. Formally, given a sequence of feature vectors $\mathbf{X} = \{x_1, x_2, \cdots, x_T\}$ and a generative model (such as a GMM, or a HMM) with parameter vector θ that models the underlying generation process, the Fisher score vector U_X is the vector of derivatives of the log-likelihood of $\mathbf{X}$ with respect to the parameters θ by the following equation:

$$U_X^\theta = \bigtriangledown_\theta \log P(\mathbf{X}|\theta) \tag{4.8}$$

Each component in U_X^θ is a derivative of the log-likelihood of the segment $\mathbf{X}$ with respect to a particular parameter of the generative model. When multiple generative models $\theta_1 \cdots \theta_m$

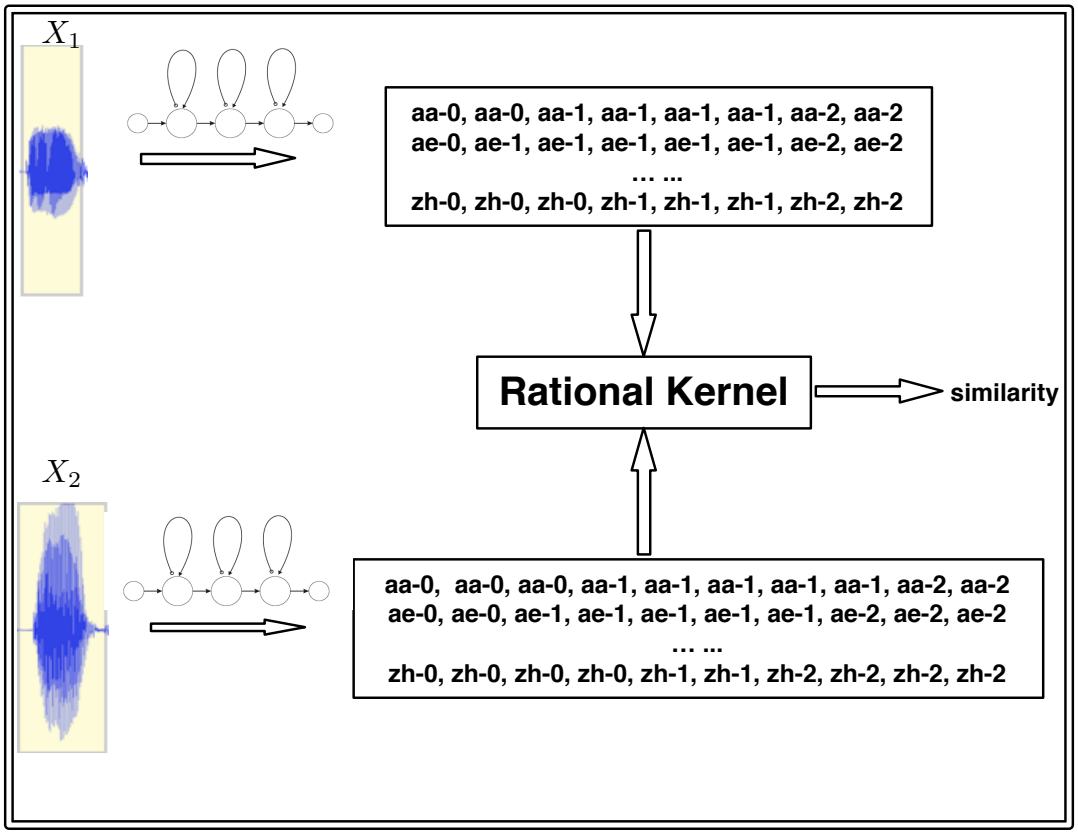


Figure 4.3: Rational-kernel approach for variable-length inputs.

are used, the resulting Fisher score vectors are derived by simply concatenating each Fisher score vector as follows:

$$U_X^\theta = ((U_X^{\theta_1})^\intercal, (U_X^{\theta_2})^\intercal, \cdots, (U_X^{\theta_m})^\intercal)^\intercal \tag{4.9}$$

In our case, we use a GMM-HMM as the generative model. Each HMM consists of 3 states, and each state is parameterized by a N-component Gaussian mixture model. Each GMM consists of the mean vector μ_i, the diagonal covariance matrix $\mathbf{\Sigma}_i$. The derivatives of a speech segment $\mathbf{X} = \{x_1, x_2, \cdots, x_T\}$ with respect to the mean vector are given as follows:

$$\bigtriangledown_{\mu_i} \log P(\mathbf{X}|\theta) = \sum_{t=1}^{T} P(i|\mathbf{x_t})\Sigma_i^{-1}(\mathbf{x}_t - \mu_i) \tag{4.10}$$

where $P(i|\mathbf{x_i})$ is the posterior of mixture i given $\mathbf{x_t}$.

Similarly, the derivatives of a speech segment $\mathbf{X} = \{x_1, x_2, \cdots, x_T\}$ with respect to the diagonal covariance matrix are given as follows:

$$\bigtriangledown_{\mathbf{\Sigma}_i} \log P(\mathbf{X}|\theta) = \frac{1}{2} \sum_{t=1}^{T} P(i|\mathbf{x_t})(-\Sigma_i^{-1} + \Sigma_i^{-1}(\mathbf{x}_t - \mu_i)(\mathbf{x}_t - \mu_i)^\intercal \Sigma_i^{-1}) \tag{4.11}$$

After the Fisher score vectors have been obtained, the Fisher kernel is then defined as

$$K(X_i, X_j) = U_X^\intercal I^{-1} U_Y \tag{4.12}$$

where I is the covariance matrix of the Fisher score vectors, also called the Fisher information matrix. For computational reasons, I is often omitted, or it is approximated by its diagonal. Alternatively, we can also use the Euclidean distance between two Fisher score vectors, and convert it to similarity using a RBF kernel function. Figure 4.4 shows an illustration of Fisher kernel approach for segment classification.

Depending on the number of models and parameters in the generative models, the Fisher kernel vector can be extremely high-dimensional, and many dimensions may be redundant. As a short digression, we will describe a novel feature selection method that alleviates the intensive computational cost in the following section.

4.3.5 Digression: Feature Selection in Fisher Score Vector Space

The Fisher score vector is a long vector of derivatives of the log-likelihood of the data with respect to the parameters θ. Depending on the number of models and number of parameters per model, Fisher score vectors can be very high-dimensional. For example, in segment-level phonetic classification task where we use 48 3-state monophone HMM models, each of which is trained with a 16-component Gaussian mixture model, the resulting acoustic score feature vector is extremely high-dimensional: more than $180,000$ in dimensionality. This could be a potential problem for performing practical graph-based SSL procedures: 1) in many of these dimensions, there might exist little information or redundancy with other dimensions, some of which could be noisy as well. 2) in graph construction, the computational complexity

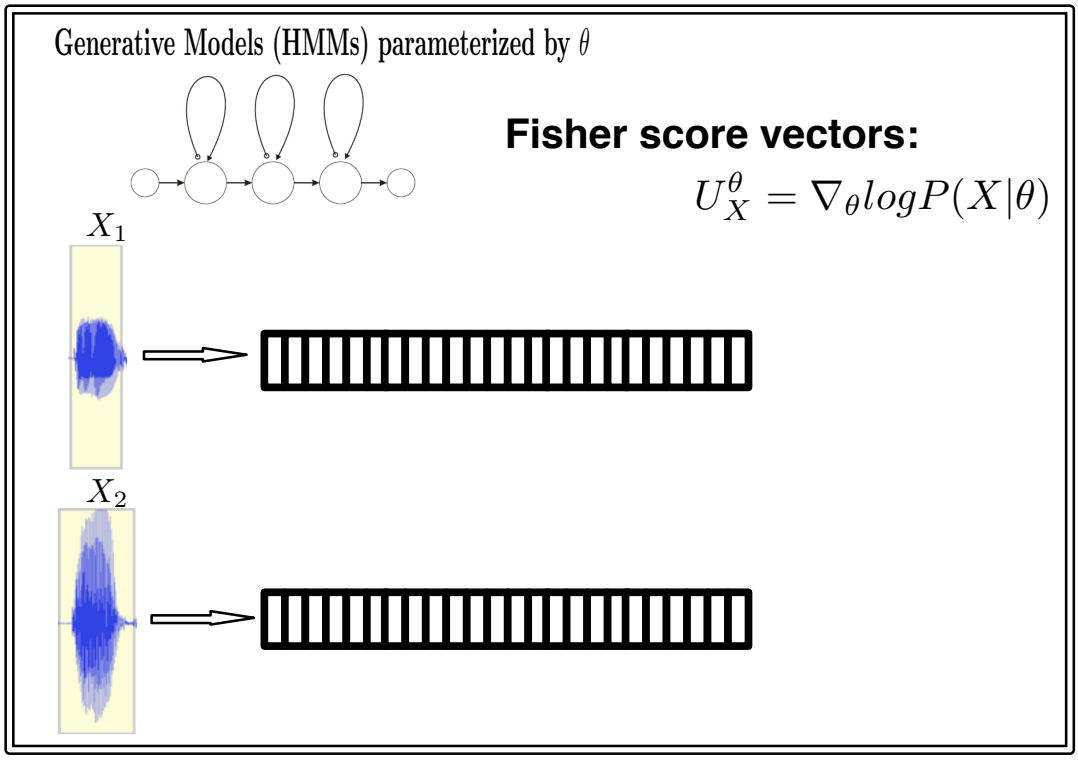


Figure 4.4: Fisher-kernel approach for variable-length inputs. Fisher kernel maps variable-length input vectors to fixed-length Fisher score vectors.

of a pairwise Euclidean distance is linear with respect to the dimensionality of the feature vectors. Thus, an efficient dimensionality reduction procedure is needed, both to reduce the computational cost in graph construction, and to remove the noise and redundancy in the feature vectors.

Dimensionality reduction algorithms are often used to map the original high-dimensional feature space into a low-dimensional embedding. The mapping is learned by preserving the locality information of samples. However, in a very high-dimensional feature space, spectral based dimensionality reduction algorithms, such as principle component analysis (PCA) [69], Isomap [133], local linear embedding (LLE) [117], maximum variance unfolding (MVU)

[144], become impractical as these algorithms need to invert a graph Laplacian. Instead of performing dimensionality reduction, feature selection methods are more preferable as they select a subset of important features or attributes. Unlike dimensionality reduction, where a mapping is learned to transform the high-dimensional feature space into a low-dimensional space, feature selection methods select a subset of features or attributes. A good feature selection method can produce a meaningful subset of features that is *interpretable* by users; it also reduces the complexity of the model which saves time for both training and testing.

Different feature selection methods have been proposed in the past. A classical method is Lasso [135], which was initially used in training a linear regression model. Lasso penalizes the model parameters with an ℓ_1 norm, which explicitly forces many of the parameter weights to be zero. The remaining features with non-zero weights are selected as the features. Lasso is a widely used technique in different models, including logistic regression, support vector machines, and deep neural networks [92]. More recently, a feature selection based on gradient boosting [148] was proposed. Another feature selection method is based on the dependency between features and class labels. Common methods include the mutual information, pointwise mutual information and Pearson correlation coefficient [46]. While these feature selection methods capture the most relevant features for classification tasks, they ignore the redundancy within the selected feature sets. A popular feature selection method "maximum relevance minimum redundancy (mRMR)" was proposed in [108]. On one hand, mRMR explores the most relevant features about the target labels; on the other hand, it tries to minimize the redundancy among the selected features. However, a caveat of the method is the quadratic complexity, which is impractical for a feature selection task in a feature space with hundreds of thousands of dimensions.

For feature selection in Fisher score vectors, previous works either selectively used the derivatives of only some parameters [101], such as only mean vectors or mixture weights of Gaussian mixtures, to reduce the number of dimensions; or they have applied binary compression [110]. In this thesis, we introduce a novel feature selection method that is applicable to high-dimensional feature space using submodular function optimization [32].

Problem Statement

Formally, the feature subset selection problem is formulated as maximizing a monotone submodular function f, subject to a cardinality constraint. To be specific, given the number K, which is the number of features to be selected, we are solving the following optimization problem:

$$\underset{S \subseteq V}{\operatorname{argmax}} \{f(S) : |S| \leq K\} \tag{4.13}$$

where $V = \{v_1, v_2, \cdots, v_N\}$ is the entire set of features, $|S|$ is the cardinality of the feature subsets and K is the number of features to be selected. We define the function f as the following form:

$$f(S) = \mathcal{L}(S) + \lambda \mathcal{R}(S) \tag{4.14}$$

where the first term $\mathcal{L}(S)$ measures how well the selected feature *covers*, or *represents*, the original feature sets; the second term $\mathcal{R}(S)$ measures how *diverse* the selected feature subset is. The notation is similar to a traditional machine learning optimization problem, where the first term measures the accuracy or fitness of the model, and the second term serves as a regularization term to control the complexity of the model to avoid overfitting. For concrete function instantiations, readers are referred to our previous work [93, 141, 75, 74].

The objective in Equation 4.14 is a monotone non-decreasing submodular function, hence, the formulated optimization problem for feature selection can be solved near-optimally using a greedy algorithm. The greedy algorithm is guaranteed to approximate the optimal solution to within a constant factor $1 - \frac{1}{e}$. In our case, we use an accelerated greedy algorithm [99] to further speed up algorithm.

Connection to Maximum Relevance Minimum Redundancy Feature Selection

Maximum Relevance Minimum Redundancy (mRMR) feature selection, as proposed in [108], is a widely used feature selection method. The motivation of the method is to select features that are most relevant in class prediction, while keeping the redundancy within the selected features as small as possible. Our proposed submodular feature selection is analogous to

mRMR: 1) the $\mathcal{L}(S)$ term in the objective function is the maximum relevance term as we want to cover as many features as possible to represent the original feature set; 2) the $\mathcal{R}(S)$ term serves as a regularizer so that the redundancy within selected features is small. Since the $\mathcal{R}(S)$ term encourages features to be more diverse, selecting redundant features is not favored.

The submodular feature selection method also scales to very high-dimensional feature spaces. In mRMR approach, a naive greedy algorithm is applied where the complexity is $O(N^2)$. When we need to rank the importance of hundreds of thousands of features, it is not practical to apply mRMR. For the submodular feature selection, because of the accelerated greedy algorithm [99], it can handle a very high-dimensional feature space.

4.4 Experimental Results: Segment-Level Classification

Similar to the previous frame-level classification task, we use the TIMIT dataset [35] with the same experimental setup. The number of samples for the training, development and test set are 121385, 7416, and 6589, respectively. To simulate different training conditions for semi-supervised learning, we also use 10%, 30%, 50% and 100% of the training data, to investigate the performance of graph-based SSL algorithms on varying amounts of training data.

4.4.1 Supervised Baseline Systems

We train a supervised baseline system for the segment classification task. For each training data condition, we train the baseline system separately using the GMTK toolkit [13]. The supervised baseline system is a 3-state monophone HMM system for each phone. In each monophone HMM, we use 16-component Gaussian mixture models with diagonal covariance matrices for each phone state. To classify the phoneme segment, we run each one of the 48 HMMs over the segment. The phone class prediction is given by the following equation:

$$c = \text{argmax}_i p(\mathbf{O}; \theta_i) \tag{4.15}$$

where θ_i are the parameters in the i-th monophone HMM, and $p(\mathbf{O};\theta_i)$ is the log-likelihood of segment $\mathbf{O}$ being generated by that model. Table 4.3 shows the classification accuracy of the baseline system under each training condition. The performance of the baseline system saturates at 30% of the training data; for the higher percentage cases, better performance might be achieved by using more mixture components; however this would lead to even higher-dimensional Fisher score spaces, as explained below.

	Amount of training data			
System	10%	30%	50%	100%
HMM	66.81	68.09	67.92	68.02

Table 4.3: Supervised baseline system accuracy rates (%) for segment classification.

4.4.2 First-pass classifier approach

The first experiment is to evaluate the first-pass classifier approach. Each test speech segment is converted into a probability distribution by the baseline classifier, and 48 canonical vectors are defined as the training examples. Jenson-Shannon divergence is used to compute the distance, and the similarity value is derived using an RBF kernel function. A 10-NN graph is constructed for each training condition.

Table 4.4 shows the test set results. In the table, we observe significant improvements over the baseline systems in different training conditions. This proves our method to be an efficient way to handle variable-length input vectors for graph-based SSL methods.

4.4.3 Rational kernel approach

In this work, we apply rational kernels on a N-best list of hypotheses. To be specific, we use each one of the 48 HMMs to generate a hypothesis over the segments, where each hypothesis is a string of HMM states and takes the following form: ax-[0],ax-[0],ax-[1],ax-

	Amount of training data			
System	10%	30%	50%	100%
HMM	66.81	68.09	67.92	68.02
LP	**66.87**	**68.45**	**68.77**	**69.10**
MP	**66.93**	**68.80**	**68.86**	**69.84**
MAD	**68.81***	**69.30**	**69.25**	**69.19***
pMP	**69.33***	**69.59***	**69.39**	**69.74***

Table 4.4: Accuracy rates (%) for segment classification, baseline system (HMM) and the various graph-based SSL algorithms. Similarity measure is based on Jenson-Shannon divergence between pairwise segment posteriors. Bold-face numbers indicate improvement over the baseline; bold-face numbers with $*$ marks are significant ($p < 0.05$) improvements over the baseline.

[1],ax-[1],ax-[2],ax-[2]. In total, we can generate a 48-best list for each segment. To compute the similarity between a pair of 48-best lists, a rational kernel can be applied. We use the C++ implementation based on OpenFST and OpenKernel [4]. Here the order of the kernel is 3; the maximum gap allowed between two symbols is set to 2; and the gap penalty is set 0.3. The state output for the labeled segments can be extracted by performing forced alignment. After the similarity has been computed, a 10-NN graph is constructed and we again run 4 different graph-based SSL algorithms on the graphs. In Table 4.5, we observe consistent improvements over the baselines and significant improvements with the proposed pMP algorithm. In the 10% case, this method outperforms the other two graph construction procedures, indicating that the rational kernel is indeed helping to build a more accurate graph with very limited amount of labeled data. In this particular case, all the graph-based learners gain accuracy. To our knowledge, this is the first time that a discrete symbol-based graph construction procedure is performed on graph-based SSL.

	Amount of training data			
System	10%	30%	50%	100%
HMM	66.81	68.09	67.92	68.02
LP	**68.02**	**69.34**	**69.25**	**69.42**
MP	**63.32**	**69.48**	**69.78***	**69.49**
MAD	**69.02***	**69.45**	**69.40***	**69.69***
pMP	**70.03***	**71.30***	**71.27***	**70.69***

Table 4.5: Accuracy rates (%) for segment classification, baseline system (HMM) and the various graph-based SSL algorithms. Similarity measure is based on 3-gram rational kernel, with gap penalty $\lambda = 0.3$. Bold-face numbers indicate improvement over the baseline; bold-face numbers with $*$ marks are significant ($p < 0.05$) improvements over the baseline.

4.4.4 Fisher kernel approach

To compute the Fisher score vector for each speech segment, we need a generative model. In this case, for each one of the 48 phones, we train 3-state HMMs, where each state is parameterized by a 16-component GMM. In order to compute the Fisher score vectors, we take derivatives of all parameters in the GMMs, and stack them into one single vector using the GMTK toolkit. The resulting dimensionality of the Fisher kernel vector is 182017.

The huge dimensionality makes graph construction computationally prohibitive: the computation for pairwise similarity is too costly, and the space cost for storing the vectors is demanding. To reduce the dimensionality, we first prune the original feature set by eliminating each feature i in the initial feature set F for which its mutual information with the classes, $MI(f_i; C)$, is less than a threshold τ (set to 0.01 in our case). In order to compute mutual information on continuous Fisher scores, they are first quantized into 50 equal-width bins. This is similar to traditional feature selection based on mutual information value ranking. We then apply our submodular feature selection method [93, 75] on the remaining feature

set (73978 features), and reduce the dimensionality to 800 final features. To compute the similarities, we first compute the dot product on these vectors, and then normalize them by the following equation:

$$\tilde{K}_{ij} = \frac{K_{ij}}{\sqrt{K_{ii}K_{jj}}} \tag{4.16}$$

Again, a 10-nearest neighbor graph is constructed for each training condition. Table 4.6 shows improvements of different graph-based SSL algorithms over the baseline system under each training condition. Again, the proposed pMP algorithm and MAD outperform label propagation and measure propagation by a large margin. The best improvements over the supervised baseline classifier are achieved by the pMP algorithm, with an absolute gain from 2.62% to 3.64%.

	Amount of training data			
System	10%	30%	50%	100%
HMM	66.81	68.09	67.92	68.02
LP	62.01	67.26	**68.72**	**69.43**
MP	63.32	67.49	**69.01**	**70.45***
MAD	**67.23**	**69.42**	**69.62***	**70.21***
pMP	**70.45***	**70.71***	**70.89***	**71.15***

Table 4.6: Accuracy rates (%) for segment classification, baseline system (HMM) and the various graph-based SSL algorithms. Similarity measure is based on Euclidean distance between pairwise Fisher kernel vectors. Bold-face numbers indicate improvement over the baseline; bold-face numbers with $*$ marks are significant ($p < 0.05$) improvements over the baseline.

In addition, we compare the number of features used for graph construction and the effects on the classification accuracy. Recall that in order to alleviate the computational cost in high-dimensional Fisher score vector space, we perform feature selection and reduce the

dimensionality of the score vectors. Here, we compare our submodular (SubMod) feature selection to a modular feature selection. In the modular (Mod) feature selection, we select the most informative features based on mutual information value between feature and class labels. This method is modular because it does not allow any direct interaction between the features (unlike submodular approach which does). We select the same amount of features using both approaches, and use the reduced feature vectors for graph construction. After the graph has been constructed, we run the graph-based SSL algorithm and compare the classification accuracy.

Table 4.7 shows the results of segment classification using submodular feature selection. Here, the baseline HMMs have an accuracy of 68.02%, which corresponds to the system using 100% training data. We reduce the size of the Fisher score vectors to 400, 800, 2000, 4000 and 10,000. With only 400 features, the modular feature selection leads to a significant drop in performance, resulting in $\sim 42\%$ classification accuracy; however, using the top 400 features selected by the submodular method we achieve $\sim 67\%$. With 800 features selected by the submodular method, we obtain a significant improvement over the baseline model with only 0.4% the size of the original features. This shows that our method based on submodular optimization is highly effective in selecting the most useful features. More importantly, the selection procedures can be done quite rapidly. For additional applications of the proposed submodular feature selection, readers are referred to [93, 75].

	400		800		2k		4k		10k	
	LP	MP	LP	MP	LP	MP	LP	MP	LP	MP
Mod	42.07	42.95	63.83	64.23	62.80	64.09	68.05	68.99	63.48	64.30
SubMod	**66.46**	**67.16**	**69.43**	**70.45**	**68.87**	**69.25**	**69.24**	69.48	**67.04**	**67.14**

Table 4.7: Accuracy rates for segment classification, with modular (mod) and submodular (sub) feature selection. The baseline model (monophone HMMs, without graph-based learning) has a classification accuracy of 68.02%. Bold-face numbers are significant ($p < 0.05$) improvements over the modular MI-based method.

4.5 Summary

In this chapter, we evaluate the pMP algorithm on TIMIT phone classification tasks. We perform evaluation on both frame-level and segment-level classification. For segment-level classification, we propose several novel graph construction techniques that allow graph-based SSL algorithms to handle variable-length input vectors. We show that under identical conditions the pMP algorithm consistently outperforms three state-of-the-art graph-based SSL algorithms as well as supervised baseline system.

Chapter 5

GRAPH-BASED SEMI-SUPERVISED LEARNING IN SPEECH RECOGNITION

In this chapter, we describe a system integration that combines graph-based SSL with a fully-fledged DNN-based ASR system. Previous works were limited to phone classification tasks. It is yet to be discovered how to integrate graph-based SSL methods into a fully-fledged ASR system. More importantly, previous works only show improvements over a GMM-based system. In this chapter, our goal is to investigate the graph-based learning with a state-of-the-art acoustic modeling approach (i.e. DNN-based acoustic models).

In the following sections, we propose a lattice-based system integration, which allows the graph-based SSL to be integrated into a fully-fledged DNN-based system. We show that graph-based SSL method can achieve a significant improvement in frame-level accuracy over a state-of-the-art DNN system, and a consistent improvement in the word error rate. In Section 5.1, we describe the lattice-based integration framework. In Section 5.2, we describe the datasets and baseline DNN systems. In Section 5.3, we describe various graph improvement methods using different feature representations for graph construction. In Section 5.4, we show experimental results on phone recognition and continuous word recognition tasks. In Section 5.5, we describe the limitations of the lattice-based integration framework. We wrap up this chapter in Section 5.6.

5.1 Lattice-based System Integration

In this section, we describe a proposed lattice-based integration framework [90]. A word lattice is a compact representation of a set of candidates, which correspond to the highly probable decoding outputs. It is often represented as a weighted finite state automata

(WFSA) $\mathbf{G} = \{V, E, W\}$, where each node $v \in V$ is associated with a time point, and each edge $e \in E$ is connect to two nodes $u, v \in V$, which represents the starting and ending time for a word. There are two associated scores on the edges: an acoustic model (AM) score, and a language model (LM) score. In this framework, we inject information from the graph-based learner into the lattices. The information we inject is the senone posteriors from the graph-based SSL algorithm. These posteriors are converted to likelihoods and then linearly interpolated into the lattice. To distinguish these likelihood scores from the DNN acoustic likelihood scores, we refer to them as "**graph likelihood**" scores. Figure 5.1 shows the framework of integrating graph-based SSL into a fully-fledged DNN-based system.

In Figure 5.1, the solids arrows correspond to the procedures in a conventional DNN system. During the training stage of the DNN-based system, the training samples in $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$ are used to train a DNN acoustic model, where $\mathbf{x}$ is the acoustic feature vector, and y_i is the target context-dependent phone state (senone) which is derived from forced alignment.

During test time, we obtain the senone posterior $p(y|\mathbf{x})$ for each frame in the test set using the trained DNN model. To get the acoustic likelihoods $p(\mathbf{x}|y)$, we divide the senone posterior by its prior $p(y)$. With the pronunciation dictionary and language model, a decoder is used to generate lattices and the best paths are obtained.

The graph-based SSL method can be incorporated into the conventional supervised framework. The dotted arrows in Figure 5.1 show graph-based SSL procedures. The steps of our graph-based method for acoustic modeling are as follows:

1. Given both training and test samples, a graph is constructed over the entire set $\mathcal{L}$ and $\mathcal{U}$. Each node is the graph is associated with an acoustic feature vector (i.e. MFCC, fMLLR) from $\mathcal{L}$ and $\mathcal{U}$. To alleviate the huge computational overhead, a sparse graph is usually constructed instead of a fully dense graph. For each node in the test sample, we find k nearest neighbors in $\mathcal{L}$ and k nearest neighbors in $\mathcal{U}$. The nearest-neighbor graph is further symmetrized: if node i is a nearest neighbor for node j, we also add

node j as the nearest neighbor for node i.

2. The prior-regularized measure propagation algorithm is then run over the graph to produce a new set of senone posteriors $\mathbf{p}_i$ for each frame $i \in \mathcal{U}$. To run the graph inference, we derive the reference label distributions over senones $\mathbf{r}_i, i \in \mathcal{L}$ for each node in the training data from the forced alignment. The prior distributions over senones $\tilde{\mathbf{p}}_i, i \in \mathcal{U}$ are derived using the supervised DNN classifier. We minimize the pMP objective defined in Equation 2.37, and derive a set of posteriors $\mathbf{p}$ over $\mathcal{U}$. These posteriors are the are converted to "graph likelihood" (to distinguish them from the ordinary acoustic likelihood) as follows:

$$p(\mathbf{x}|s_j) \propto \frac{p(s_j|\mathbf{x})}{p(s_j)} \tag{5.1}$$

where $p(s_j|\mathbf{x})$ is the posterior of senone s_j derived from pMP algorithm, and $p(s_j)$ is the prior distribution for senone s_j.

3. The new graph likelihoods are interpolated with the original acoustic score and the LM score in the original lattices for lattice rescoring. Let the acoustic likelihood, language model likelihood and graph likelihood be AM_{score}, LM_{score} and G_{score}, respectively. We use the following equation for each edge on the lattice for rescoring:

$$AM_{score} + \lambda LM_{score} + \theta G_{score} \tag{5.2}$$

The optimal constants are tuned on a development set.

5.2 Data and System

In this section, we describe the datasets and baseline systems. We evaluate the proposed framework on a continuous phone recognition task and two continuous word recognition tasks. For the phone recognition task, we use the TIMIT dataset [35]. We use 3696 sentences (462 speakers), 200 sentences (25 speakers), 192 sentences (24 speakers) as training,

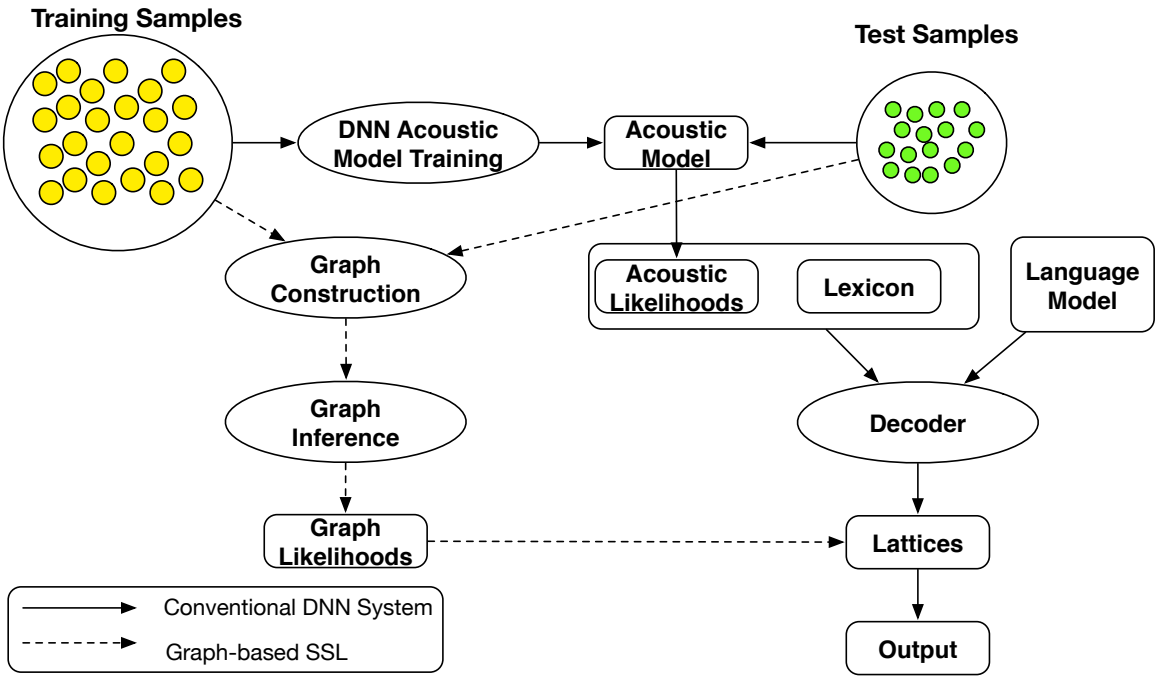


Figure 5.1: A flowchart of integrating GB-SSL into DNN-HMM system.

development and evaluation datasets, respectively. For the continuous word recognition task, we evaluate the proposed framework on two small datasets. The first task is the DARPA Resource Management (RM) task [114]. For training we use the RM-SI dataset, which consists of 3990 sentences. We use the *test-feb89* set as our dev set, which consists of 300 sentences. As evaluation sets, we use *test-oct89*, *test-feb91*, and *test-sep92*, each of which consists of 300 sentences. The second task is a subset of the Switchboard-I dataset [87] that was selected to have a limited vocabulary while being acoustically rich. For evaluation purposes, we use the 100-vocabulary subset of Switchboard-I, which we refer to as the SWB-100 dataset. Table 5.1 shows the sizes of the data sets (amount of non-silence speech) and of the vocabulary sizes for the word recognition task. Although these two word recognition tasks are much smaller compared to a LVCSR task, they represent a fair amount of acoustic variations, multiple speakers, and different recording conditions (both read speech and conversational speech).

Set	Training	Dev	Test	Vocab
RM	3.8	0.27	0.83	1000
SWB-100	3.79	1.27	1.26	100

Table 5.1: Vocabulary sizes and data set sizes (in hours) for Resource Management (RM) and Switchboard subset (SWB-100) tasks.

For front-end feature processing, we extract 13-dimensional MFCC feature vectors (12 MFCC coefficients and 1 energy coefficient) every 10 milliseconds with a window size of 25 milliseconds. Cepstral mean vectors are subtracted on a per speaker/ conversation-side basis. We also concatenate feature vectors with a context size of 9 frames (± 4 frames) and use LDA to project the features down to a 40-dimensional feature vector for the GMM-HMM training. For TIMIT phone recognition, we also apply a maximum likelihood linear transformation (MLLT) [40] (also known as global semi-tied covariance transform [34]), followed by a feature-space MLLR (fMLLR) transform per speaker at both training and test time using a GMM-HMM system. We further splice the fMLLR features with a context size of 9 frames, and use the 360-dimensional feature as inputs to the DNN.

Our baseline DNN system is bootstrapped from the GMM-HMM system. The forced alignments from GMM-HMM system are used as the target labels for the DNN training. The deep neural networks for the recognition tasks have 4 hidden layers. The nonlinear activation function in each layer is a *tanh* function. The output layer uses a softmax function, and represents the posterior of the senones. The network architecture for the three recognition tasks is listed in Table 5.2.

We use the cross entropy training for the DNN with greedy layer-wise supervised training [10, 124]. The network weights are randomly initialized with a normal distribution multiplied by 0.1, and the biases of the sigmoid units are initialized by sampling uniformly from the interval $[-4.1, -3.9]$. We use 20 epochs to train the baseline DNN, with a mini-batch size of 256. For the first 15 epochs, we decrease the learning rate from 0.01 to 0.001

Task	Network 1					
	I	H1	H2	B	H3	O
TIMIT	360	1024	1024	1024	1024	1951
RM	40	1070	1070	1070	1070	1421
SWB-100	40	1185	1185	1185	1185	624

Table 5.2: Sizes of networks' input (I), hidden (H), bottleneck (B) and output (O) layers for TIMIT, RM, and SWB-100 tasks.

and fix the learning rate at 0.001 for the last 5 epochs. We use the Kaldi toolkit [111] to train all systems. The language models are trained using SRILM [127]: we train a bigram phone model for TIMIT, a bigram model for the RM task and a trigram language model with standard backoff for the SWB-100 task.

5.3 Improvements in Graph Constructions

In this section, we describe several techniques to improve the quality of the graphs. It is widely acknowledged that the quality of the graph far outweighs the graph-based SSL algorithm [158]. The quality of the graph hinges on the feature representation and similarity measures. In previous studies of graph-based SSL methods in speech processing, all the graphs use the front-end features (i.e. MFCC features) directly for graph construction. However, as pointed in [88], with a better baseline classifier, the improvements become smaller. We describe and evaluate different feature representations for graph construction. We use the SWB-100 development set to evaluate the accuracy of frame-level HMM state classification as well as word error rate using different graph construction techniques.

5.3.1 Feature Representations for Graph Construction

In this section, we explore different feature representations that are directly related to the DNN system. We compare the performance of both frame classification accuracy and word

error rate using different features on the SWB-100 development set. The features we considered to construct the graph are the following:

1. Front-end features (LDA): the input features to the DNN (spliced MFCC features with a window of 4 frames, and a LDA projection down to 40 dimensions).

2. DNN posterior features: posterior distributions from the softmax function of the baseline DNN; we concatenate posterior distributions within a window of 2 frames.

3. DNN hidden-layer features (HLF): the linear outputs from the last hidden layer in the DNN.

4. DNN bottleneck-layer features (BNF): the linear outputs from the bottleneck layer in the DNN, which is inserted between the last two hidden layers. We splice the features with a window of 9 ($\pm$4 frames) to get the final features.

After the graphs have been constructed, we run the pMP algorithm over the graphs and evaluate the classification accuracy. To evaluate the frame-level state classification accuracy, we use the HMM state labels derived from a forced alignment of the HMM states with the baseline DNN and use them as ground-truth reference. Figure 5.2 shows the frame-level senone classification accuracy using graphs with different feature representations. We observe a consistent improvement in the classification accuracy using pMP compared to the supervised baseline system. The improvements over the supervised baseline are all significant at $p = 0.001$. However, with improved frame-level accuracy the word error rate does not necessarily improve with all features. Table 5.3 shows the word error rate using different feature representations. Among these features, the bottleneck features yield the best performance, outperforming the baseline DNN by a relative improvement of over 20% in frame-level accuracy and a 1% absolute reduction in word error rate. On the contrary, the input-level (LDA features) and output-level features (posterior features) do not yield an improvement in word error rate (sometimes they even result in higher word error rate). The raw

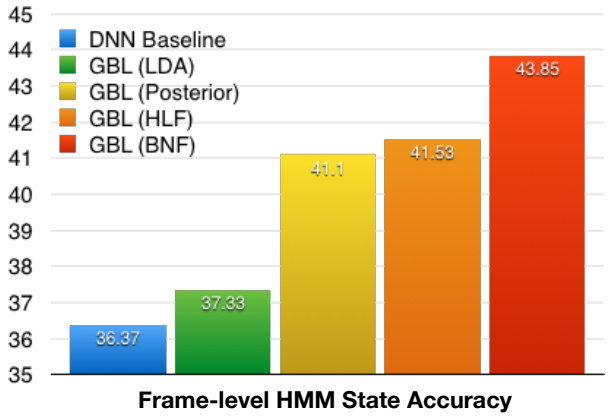


Figure 5.2: Frame-level HMM state prediction accuracy using different feature representations on the SWB-100 dev set.

input-level features may be too noisy to discriminate between different senones. Posterior features, on the other hand, typically represent sharply peaked, low-entropy distributions that tend to introduce search errors during the subsequent Viterbi decoding process when predictions are wrong but highly confident. Hidden layer features and bottleneck features are automatically learned in deep neural networks via multiple layers of nonlinear transformation. These features are more discriminative than the input features but at the same time more fine-grained than the posterior features. The bottleneck features outperform the hidden layer features because, due to their low dimensionality, they are concatenated across 4 frames, thus providing context-dependent information. Similar splicing could be applied to hidden layer features but the resulting high dimensionality of the feature vectors would slow down the graph construction process.

System	WER
DNN Baseline	27.40%
GBL: LDA	27.06%
GBL: Posterior	28.42%
GBL: HLF	26.99%
GBL: BNF	26.45%

Table 5.3: Word error rate (WER) of graphs using different feature representations, SWB-100 development set.

5.4 Experimental Results

5.4.1 Phone recognition

In this section, we show the experimental results using graph-based SSL on the TIMIT phone recognition task. Unlike phone classification tasks in the previous chapter, where the frame-level labels are provided, phone recognition needs to recognize continuous phone sequences. Because the language model is much simpler than continuous word recognition, phone recognition is often used as a pilot study to evaluate acoustic modeling methods on continuous recognition task.

To simulate different learning conditions, we use 30%, 50% and 100% of the training data. A 40-dimensional bottleneck layer feature is used for graph construction. The number of nearest neighbors is set to 10. After the graph is constructed over the training and test samples, we use the pMP algorithm to do the inference and derive a new set of senone posteriors. These posteriors are converted to likelihoods by dividing the senone prior distributions. The optimal weights for acoustic likelihoods, graph likelihoods and language model scores are 1, 0.4 and 3, respectively. We also replace the acoustic likelihood with the graph likelihoods on the lattices. Table 5.4 shows the frame-level accuracy and phone error rate (PER) on the development set and evaluation set. The row 'baseline DNN' indicates the

baseline DNN performance; the row 'GBL only' shows the results by replacing all acoustic likelihoods with the graph likelihoods on the lattices; the row 'GBL+DNN' shows the results of interpolating graph likelihoods into the acoustic likelihoods and language model scores on the lattices.

Among all training conditions, the frame-level accuracy improves by at least 5% absolute. The phone error rate improves by 0.7% - 1% absolute when we replace the original DNN acoustic likelihoods with the new graph likelihoods. The phone error rate further improves by 1.2% absolute when the graph likelihoods are interpolated into the lattices.

Graph-based SSL can be viewed as an adaptation technique at test time. The baseline DNN systems in this task use speaker adaptively trained features (fMLLR features). Nevertheless, the graph-based SSL method further improves over a speaker-dependent DNN system significantly. This shows that graph-based SSL is complementary to existing speaker adaptation techniques such as feature-space MLLR, and it can be used to exploit additional useful information.

	Core Eval Set					
	30%		50%		100%	
	Frame Acc.	PER	Frame Acc.	PER	Frame Acc.	PER
Baseline DNN	45.35%	28.30%	48.66%	26.39%	49.85%	24.16%
GBL only	50.78%	27.60%	51.75%	25.53%	55.01%	23.13%
GBL + DNN	X	27.53%	X	25.42%	X	22.90%

Table 5.4: Frame accuracy and WER on TIMIT under different learning scenarios: 30%, 50% and 100% training data is used, respectively. Baseline DNN: 4-layer DNN system; GBL only: recognition performance by replacing original acoustic likelihoods with the graph likelihoods; GBL + DNN: recognition performance by interpolating graph likelihoods with other scores on the lattices.

5.4.2 Word recognition

In this section, we show the experimental results using graph-based SSL for continuous word recognition on SWB-100 and Resource Management (RM) datasets. The dimensionality of the bottleneck layer for SWB-100 and RM is 40 and 42, respectively. We build the kNN graph and set the number of nearest neighbors to 10. The optimal weights for acoustic likelihoods, graph likelihoods and language model score are 1, 0.3, and 20, respectively on SWB-100. In addition to using a standard trigram for decoding, we also run a decoding pass with a unigram language model (optimal score weights: 1, 0.5, and 15) to better assess the contribution of our method under a weak language model. The optimal weights for Resource Management task are 1, 1, and 6, respectively.

Table 5.5 shows results obtained on the SWB-100 test set. We see a consistent improvement over the DNN-HMM system. With a simpler language model, the improvement is around 1.6% absolute, compared to 0.4% using a trigram language model.

Table 5.6 shows frame-level accuracy and word error rates on Resource Management corpus. The baseline system is a state-of-the-art DNN system with a bigram LM, which already achieves a very low word error rate on this task (with a WER ranging from 1.73% to 4.03%). However, the GBL system gains a further improvement on all test sets except for *test-feb91*, where the word error rate remains constant.

System	Trigram	Unigram
DNN	29.11%	40.64%
GBL	28.69%	39.06%

Table 5.5: WERs on SWB-100 test set for baseline DNN system and best GBL system.

RM	Test Feb89 (Dev)		Test Feb91		Test Oct89		Test Sep92	
	DNN	GBL	DNN	GBL	DNN	GBL	DNN	GBL
Frame Acc.	58.91%	**64.44%**	57.91%	**63.09%**	58.49%	**63.98%**	54.51%	**60.19%**
WER	2.42%	2.26%	1.73%	1.73%	2.68%	2.46%	4.03%	3.52%

Table 5.6: Frame accuracy and WER on Resource Management datasets.

Speaker-dependent (SD) vs speaker-independent (SI) graph construction

In all these experiments, we construct a graph for each speaker and we refer to this graph construction technique as *speaker-dependent (SD) graph construction*. Recall that the underlying assumption of our approach is that the test data lies on a manifold. The manifold structure is dependent on the characteristics of the test data, such as speaker, channel, and noise. Constructing a graph separately for each speaker rather than globally for the entire test data is therefore a better choice. On the other hand, *speaker-independent (SI) graph construction* integrates the entire unlabeled data set into the learning process.

Figure 5.3 shows an example of SD and SI graph construction. In this figure, different colors in the nodes correspond to different speakers. In SI graph construction, for each sample in $\mathcal{U}$, we find k nearest neighbors in $\mathcal{L}$ and k nearest neighbors in $\mathcal{U}$. These neighbors can come from different speakers. In SD graph construction, we find k nearest neighbors in $\mathcal{L}$ and k nearest neighbors only in samples from that speaker rather than the entire test data $\mathcal{U}$.

We compare the performance of SD graph and SI graph in SWB-100 development set. Figure 5.4 shows the frame-level accuracy on the SWB-100 development set. We use the DNN hidden layer feature and bottleneck layer feature with speaker-independent and speaker-dependent graphs. The frame-level accuracy with a SD graph is consistently better than the accuracy achieved with a SI graph. Table 5.7 shows a comparison of the word error rates using SI and SD graph with the bottleneck layer features. The word error rate obtained with

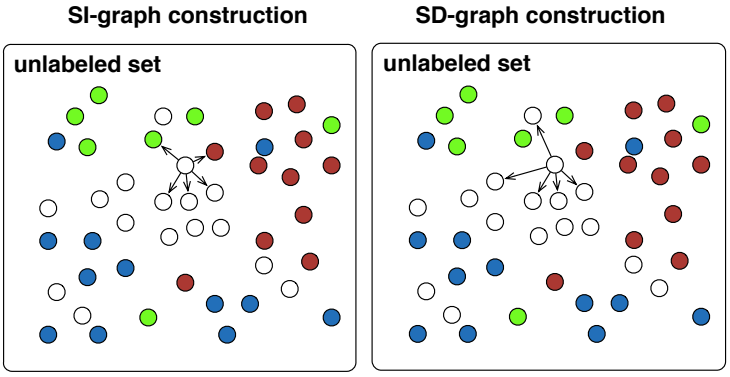


Figure 5.3: A comparison between speaker-independent (SI) and speaker-dependent (SD) graph. In SI graph construction, we select nearest neighbors from all unlabeled samples; on the contrary, in SD graph construction, we only select nearest neighbors from samples of a particular speaker.

the SD graphs is better than that obtained from the SI graph.

SI graph		SD graph	
Frame Acc.	WER	Acc	WER
42.80%	27.65	43.85%	26.45

Table 5.7: Frame-level HMM state prediction accuracy and WER for speaker-independent (SI) vs.speaker-dependent (SD) graphs on the SWB-100 dev set.

Comparison to Self Training

We also compare our results against a self-training approach. Following the conventional self training [76, 77, 48, 146, 105, 153] approach, we first train an initial DNN-HMM system using the training data, and decode on the test data. For data selection, we pick utterances with the highest average per-frame decoding likelihood and add them to the training data.

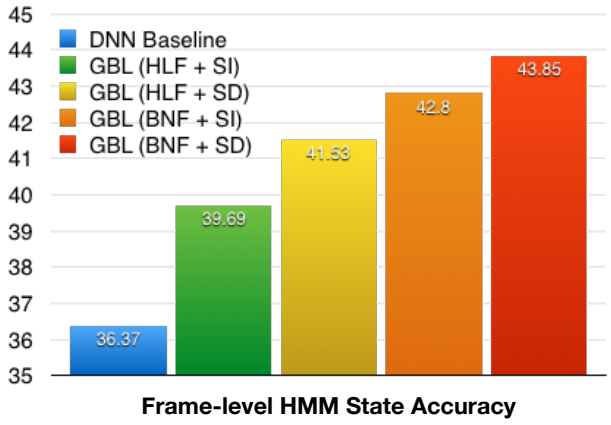


Figure 5.4: Frame-level HMM state prediction accuracy using speaker-independent (SI) and speaker-dependent (SD) graphs on the SWB-100 dev set.

Selection is stopped when the number of frames in the selected set is equal to the number of test samples used by the GBL systems, to ensure comparable conditions. This results in about 1100 utterances for SWB and 260 utterances for RM. Three iterations of self-training are performed, after which no further improvements are obtained. In addition, we also select the entire decoded test data, and add all of of them to the training data. We compare the supervised baseline system, self training without data selection and self training with data selection in Table 5.8.

Self-training with data selection achieves better results on SWB-100 and one of the RM test sets while GBL yields better results on the remaining two RM test sets. Differences are not statistically significant; however, GBL can be done in a single iteration and is very fast in itself. Graph construction and inference for the graphs used in this study took less than 10 minutes and 1-2 minutes, respectively, on a 64-core Intel 2.00GHz machine. By contrast, one iteration of baseline system retraining took 1 hour (with DNN training paralleled across 64 CPUs).

	RM				SWB-100	
System	Test Feb89 (Dev)	Test Feb91	Test Oct89	Test Sep92	dev	test
Baseline	2.42%	1.73%	2.68%	4.03%	27.40%	29.11%
Self-training, all	**1.99%**	1.77%	2.61%	3.99%	29.66%	28.95%
Self-training	**1.99%**	**1.69%**	2.98%	3.67%	**26.45%**	**28.01%**
GBL	2.26%	1.73%	**2.46%**	**3.52%**	**26.45%**	28.69%

Table 5.8: WERs for baseline system, self-trained system, and GBL. Boldface numbers indicate the best-performing system.

5.5 Limitations of the Lattice-Based Integration Framework

Acoustic modeling requires algorithms that are scalable to large-scale dataset with real-time performance. One major caveat of graph-based SSL arises from the prohibitive computational cost: when dealing with large-scale training data, the graph usually consists of millions or billions of nodes. For example, the Switchboard dataset has 120 million nodes in total, which is equivalent to 310 hours of speech. A naive similarity graph construction requires n^2 operations of the pairwise distance computations, where n is the number of nodes in the graph. To achieve faster graph construction, data structures such as k-d tree [31], ball trees [85] and cover trees [12] can be adopted. These tree-based data structures partition the input data into blocks of bounding regions. During test-time, nearest neighbor search can be sped up by examining a small number of blocks. These tree-based structures perform poorly in higher dimensional feature spaces. Other approximation techniques for graph construction have been proposed [17, 155, 26].

A second computational bottleneck arises during test time: for an acoustic modeling task, we need a real-time solution. For DNNs, real-time performance at test-time can be achieved using one pass of the forward algorithm for each test sample. However, the graph inference algorithm can be time-consuming on large graphs. Most graph-based SSL algorithms are

formulated to minimize a quadratic energy function defined on the graphs. The naive closed-form solution requires an inversion of a graph Laplacian, which takes $O(n^3)$ in time. The Jacobi iterative algorithm is often used to avoid inverting the graph Laplacian. In the pMP algorithm, we also use an iterative solver. Nevertheless, with $n = 10^8$, graph inference can take up to hours. Moreover, we need to store a large graph in memory to do the inference. Finally, whenever we have a new test set, we need to build a new graph over all training samples and the new test samples, which becomes problematic for acoustic modeling tasks.

A third problem with the lattice-based integration framework is its lack of optimization with other components in an ASR system. In graph-based SSL, the output posteriors are optimized according to the graph-based SSL criterion only. In other words, the optimization of graph-based SSL does not take other factors such as language modeling, pronunciations, and the decoders, into consideration. As a result, it is often observed that a significant improvement in frame-level accuracy does not always lead to a better word error rate.

In the next chapter, we address these issues by proposing a novel neural graph embedding approach. Instead of optimizing the frame-level accuracy in the back-end during lattice rescoring, we learn a compact representation that encodes the manifold information for each sample, and use these features as front-end inputs to the DNNs. The DNN can thus implicitly combine different information sources in the best possible way. Moreover, learning graph embeddings can also get rid of the expensive graph construction, and can achieve real-time performance during test time.

5.6 Summary

In this chapter, we describe a novel lattice-based framework that integrates the graph-based SSL into a fully-fledged DNN-based system. We show several techniques that improve the quality of graphs, and consequently obtain better frame-level accuracy and better word error rates. The framework is evaluated on both phone recognition and two small-to-medium vocabulary word recognition tasks. While previous work shows promising improvement in phone classification over a supervised GMM system, this is the first framework that shows

the efficacy of graph-based SSL in DNN-based speech recognition system. We also show that the graph-based SSL is orthogonal to existing speaker adaptation techniques in DNNs. The proposed framework can also be potentially useful in acoustic model training for low-resource languages, where only a few training data is available for each speaker.

Chapter 6

ACOUSTIC MODELING USING NEURAL GRAPH EMBEDDINGS

In the previous chapters, we described a lattice-based framework which integrates graph-based SSL into a fully-fledged DNN system. A graph is constructed over all training and test samples. After that, the graph-based SSL algorithm is run over the graph, and a new set of posteriors over senones is obtained. The new posteriors are converted to graph likelihoods and are linearly interpolated with the original scores (acoustic likelihoods and language model scores) in the lattice for lattice rescoring. The framework shows significant improvements in frame level accuracy, and a consistent reduction in phone error rate, as well as word error rate in small-to-medium vocabulary continuous word recognition tasks. However, there are several major challenges that hamper the application of graph-based SSL methods in acoustic modeling to large vocabulary conversational speech recognition (LVCSR) tasks as described in Section 5.5.

In this chapter, we address the computational and optimization challenges in the standard graph-based SSL framework for acoustic modeling by proposing a novel neural graph embedding method. The neural graph embedding approach learns a compact feature representation for each sample in the graph. These resulting features are used as inputs to the DNN classifier [91]. This chapter will be organized as follows: In Section 6.1, we give a brief introduction to neural graph embedding features. In Section 6.2, we recap previous work on learning graph embedding features and introduce an autoencoder-based graph embedding feature framework. We also describe how to integrate the graph embedding features into the DNN training. In Section 6.3 and Section 6.4, we describe two types of neural graph embedding features. The first method is based on local neighborhood information, which

is referred to as the neighborhood graph embedding; the second method is based on global manifold information, which is referred to as similarity graph embedding. In Section 6.5, we describe the LVCSR datasets and baseline systems. In Section 6.6, we show experimental results obtained with the graph embedding features. In Section 6.7, we show the connections between the proposed neural graph embedding approach and other adaptation techniques used in DNN-based acoustic models. We wrap up the chapter in Section 6.8.

6.1 Introduction

In the previous chapter, we reported improvements in both frame-level accuracy and word error rate over a standard DNN baseline system. However, our previous work has also identified two main drawbacks of the standard GBL setup: First, although the frame-level classification accuracy consistently improves it does not always result in decreases in word error rate; rather, word error rates only decrease when the frame-level improvement is substantial. In the standard setup the graph-based learner is trained and applied independently at the output level of a DNN-based HMM state classifier. The resulting modified posterior distributions are then combined with the original posteriors and are used for lattice rescoring. Thus, the GBL is not optimized jointly with other system components. Second, graph construction incurs a high computational cost, which limits the scalability to large data sets.

In this chapter, we therefore propose a different framework for integrating GBL, where the information provided by the graph is used at the *input level* to the DNN-based classifier: by embedding the similarity graph into continuous space through an autoencoder we learn novel features that are then used directly in the input layer to the DNN. The DNN learner can thus implicitly combine different information sources in the best possible way. While neural graph embeddings have been utilized previously in clustering [134] and in natural language understanding [52], their use in acoustic modeling is to our knowledge novel. In addition, we address the computational issues by landmark-based graph construction, which reduces the cost by orders of magnitude while yielding good performance.

6.2 Graph Embedding using Autoencoders

Mapping graph representations into continuous space has been proposed in previous work [134, 52]. For this purpose, an autoencoder is often used to learn a compact feature representation for a given input. The simplest form of an autoencoder is a feedforward neural network, consisting of an input layer, a hidden layer and an output layer. The hidden layer usually consists of fewer number of nodes than the number of nodes in the input layer. The output layer has the same number of nodes as the input layer. The autoencoder first maps an input vector $\mathbf{x}$ to a hidden representation $\mathbf{g}$ via a nonlinear transformation, defined as $\mathbf{g} = s(\mathbf{W}^{(1)}\mathbf{n} + \mathbf{b}^{(1)})$. This procedure is often termed as the "encoding" step, where the original feature vector is encoded into a compact feature vector. After the encoding step, the hidden representation $\mathbf{g}$ is then passed through another nonlinear transformation to reconstruct the input, i.e. $\tilde{\mathbf{x}} = s(\mathbf{W}^{(2)}\mathbf{g} + \mathbf{b}^{(2)})$. This step is often referred to as the "decoding" step. Here, $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$ are the parameters of the network and s is a nonlinear activation function, which can be either sigmoid function or *tanh* function. To train the autoencoder, we use the standard backpropagation. Instead of minimizing cross entropy, we minimize the reconstruction loss of the training samples $\sum_{i=1}^{N} ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||_2$. The nonlinear activation output of the hidden layer is used as the compact feature representation. Throughout this thesis, we refer to these features as the *neural graph embedding features*. Similar to other dimensionality reduction methods, an autoencoder learns a lower-dimensional embedding via several nonlinear transformations.

6.2.1 Previous Work

In [134], the authors use a deep autoencoder to map the similarity graph into a low-dimensional continuous vector space. Let the similarity matrix be defined as $\mathbf{S} \in \mathbb{R}^{n \times n}$, and the diagonal degree matrix as $\mathbf{D} = diag\{d_1, d_2, \cdots, d_n\}$, where d_i is the degree of node i, and n is the total number of nodes in the graph. The input to the autoencoder for each

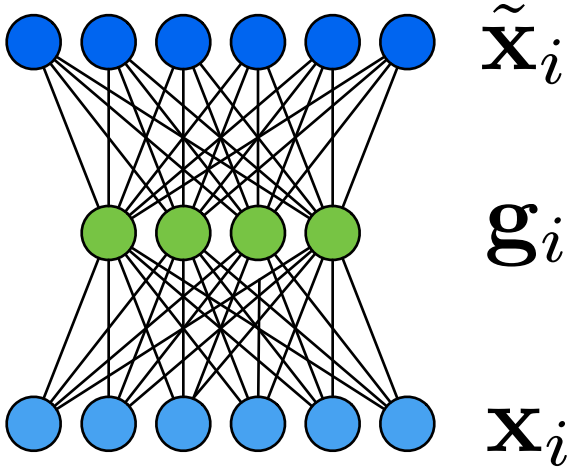


Figure 6.1: A depiction of an autoencoder.

sample i is the row vectors of the normalized similarity matrix, defined as follows:

$$\mathbf{x}_i = [\frac{S_{i1}}{d_i}, \cdots, \frac{S_{in}}{d_i}] \tag{6.1}$$

where S_{ij} is the similarity value between sample i and j. Let the autoencoder output be defined as $\tilde{\mathbf{x}}_i$ for each sample i. An autoencoder is trained to minimize the reconstruction loss $\sum_i ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||_2$. The nonlinear activation outputs from the last hidden layer are extracted as the 'graph embedding features', which are then used as inputs to a standard k-means clustering algorithm. Empirical results show that clustering with the graph embedding features yields better results than clustering with the similarity graph row vectors using k-means, or spectral clustering on the similarity graph.

In [52], the authors use neural embeddings of a semantic knowledge-graph for the purpose of semantic parsing. In [139], the authors propose a 'generalized autoencoder' which extends the traditional autoencoder by taking manifold information into the reconstruction term of the autoencoder training. Our proposed approach is similar to [134] but different in many ways. The framework in [134] is not scalable to large-scale acoustic modeling task because of the huge construction overhead of the similarity graph. Moreover, the graph is built over

one specific data set and cannot extend to any new test data. To apply on new test data, we must rebuild a new similarity graph, which becomes impractical for acoustic modeling.

6.2.2 Connections between Graph Embedding and Spectral Clustering

In this paragraph, we briefly describe the findings in [134] that showed a close connection between autoencoder-based graph embeddings and the spectral clustering approach. Given the similarity matrix $\mathbf{S}$ and the diagonal degree matrix $\mathbf{D}$, we define the graph Laplacian matrix $\mathbf{L}$ as $\mathbf{L} = \mathbf{D} - \mathbf{S}$. We also define a matrix $\mathbf{Q} = \mathbf{D}^{-1}\mathbf{L}$. For spectral clustering, we perform an eigenvalue decomposition of $\mathbf{Q}$ as $\mathbf{Q} = \mathbf{Y}\mathbf{\Lambda}\mathbf{Y}^{\intercal}$, where $\mathbf{Y} \in \mathbb{R}^{n \times n}$ is the eigenvector matrix with each column representing an eigenvector. $\mathbf{\Lambda}$ is a diagonal matrix, where each element Λ_{ii} is the ith eigenvalue and $\Lambda_{11} > \Lambda_{22} > \cdots > \Lambda_{nn}$. For spectral clustering, we extract k columns from $\mathbf{Y}$ that correspond to the smallest non-zero eigenvalues as the **embedding matrix $\mathbf{Y_k}$**. Each row vector in $\mathbf{Y_k}$ is a new feature representation for a sample, and is used as input to a standard k-means algorithm.

To relate spectral clustering with reconstruction, Eckart-Young-Mirsky Theorem [28] shows the reconstruction nature of spectral clustering.

Theorem 1 (Eckart-Young-Mirsky Theorem [28]: Reconstruction Nature of Spectral Clustering)**.** *Given a rank-r matrix $\mathbf{P} \in \mathbb{R}^{m \times n}$, let the singular value decomposition (SVD) be $\mathbf{P} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\intercal}$, where $\mathbf{U}^{\intercal}\mathbf{U} = \mathbf{I}$, $\mathbf{V}^{\intercal}\mathbf{V} = \mathbf{I}$ and $\mathbf{\Sigma} = diag\{\sigma_1, \sigma_2, \cdots, \sigma_r, 0, \cdots, 0\}$. Then, for any rank $k < r$, we can find a rank-k matrix $\tilde{\mathbf{P}}$ as follows:*

$$argmin_{\tilde{\mathbf{P}} \in \mathbb{R}^{m \times n}, rank(\tilde{\mathbf{P}}) = k} ||\mathbf{P} - \tilde{\mathbf{P}}||_F = \mathbf{U}\tilde{\Sigma}\mathbf{V}^{\intercal} \tag{6.2}$$

where $\tilde{\mathbf{\Sigma}} = diag\{\sigma_1, \sigma_2, \cdots, \sigma_k, 0, \cdots, 0\}$. $\tilde{\mathbf{\Sigma}}$ is the same matrix as $\mathbf{\Sigma}$, and the only difference is that it only contains the k largest singular values and the other singular values are replaced with 0.

When we replace $\mathbf{P}$ matrix with a symmetric similarity matrix $\mathbf{S}$ (or normalized similarity matrix $\bar{\mathbf{S}} = \mathbf{D}^{-1}\mathbf{S}$), we have eigenvalue decomposition of $\mathbf{S}$ as follows: $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\intercal}$, where

$\mathbf{\Lambda}$ is the diagonal matrix that contains the eigenvalues in $\mathbf{S}$. According to the Eckart-Young-Mirsky Theorem, we can reconstruct the similarity matrix $\mathbf{S}$ using k largest eigenvectors, which is the best rank-k approximation under the Frobenius norm. In [134], the key observation is that **the reconstruction of a similarity matrix can also be achieved by the use of an autoencoder**. Instead of minimizing the Frobenius norm, an autoencoder minimizes the following reconstruction loss:

$$\sum_{i=1}^{n} ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||_2 \tag{6.3}$$

where $\mathbf{x}_i$ is defined in Equation 6.1, which is the row vector of a normalized similarity matrix. It can be observed that the objective of autoencoders is quite similar to the loss in Frobenius norm in Equation 6.2.

6.2.3 Neural Graph Embeddings in Acoustic Modeling

In this thesis, we propose two different approaches that encode information expressed by the graph structure at different levels. The first type of information we can encode is the neighborhood information for a given sample. To infer the label distribution, the label information of the neighboring nodes plays an important role. In this approach, we will extract information that represents the neighborhood information and use this information as the inputs to the autoencoder. Another type of information we can encode is the global information for each sample in the graph. These global information can be expressed by the manifold structure, such as the distances to each node in the graph. In [134], the row vectors in a similarity matrix were used as input information for the autoencoder training. In the following two sections, we will describe how to extract neighborhood graph embeddings and similarity graph embeddings in detail, and will evaluate their performance in LVCSR tasks.

After the graph embedding features have been extracted using the autoencoder, the next question is how to combine the new features with the conventional acoustic features. A conventional input acoustic feature vector to an DNN is usually a MFCC feature vector, with possible feature preprocessing including splicing, LDA, MLLT and fMLLR feature transfor-

mation. The graph embedding features can be integrated with the acoustic features in various ways. Figure 6.2 and Figure 6.3 show the early stage and late stage feature integration. The white nodes at the input layer correspond to the raw acoustic features. The green nodes correspond to the graph embedding features and the similarity values. The blocks show different feature preprocessing steps such as initial splicing, LDA, MLLT, fMLLR transformation, and additional splicing. The dotted blocks are optional. During early stage feature integration as depicted in Figure 6.2, the graph embeddings are appended to the raw acoustic features, followed by additional preprocessing modules such as splicing, LDA and MLLT. During late stage integration as depicted in Figure 6.3, the raw acoustic features are first processed. The graph embeddings are then appended to the processed acoustic features.

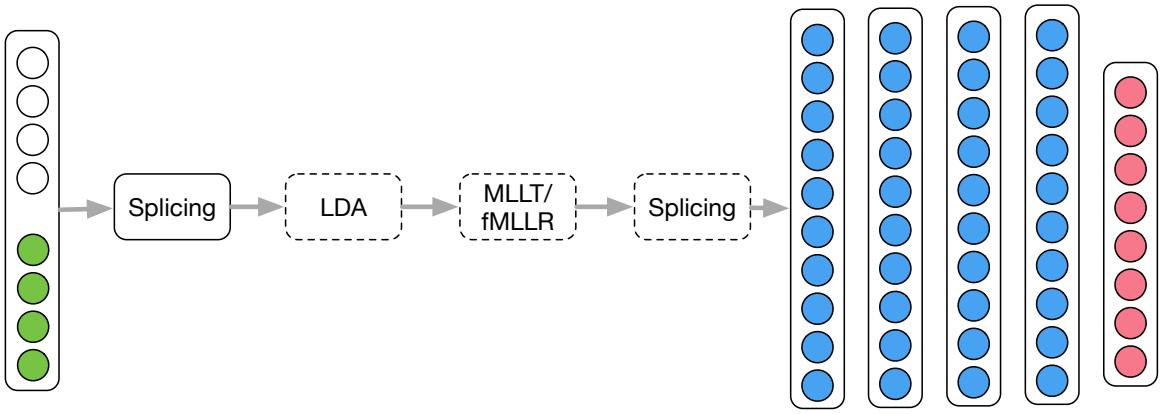


Figure 6.2: Early stage feature integration using graph embedding features. Blocks in dotted lines are optional.

6.3 Neighborhood Graph Embedding Features

In this section, we will describe the first type of neural graph embedding features - **neighborhood graph embedding features**. In the pMP algorithm, the most important factor for graph inference is the label information of the neighboring nodes for each sample. To extract graph embedding features that encode neighborhood information, we need to design

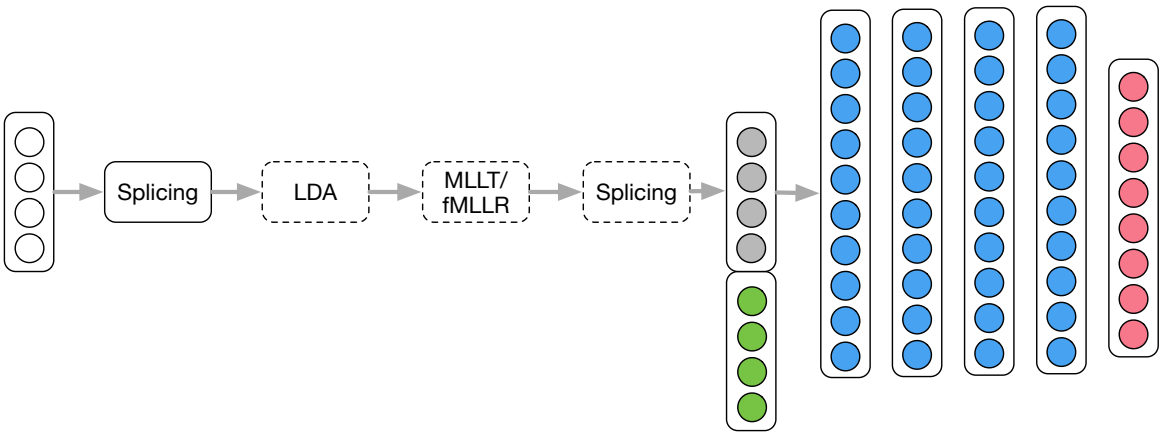


Figure 6.3: Late stage feature integration using graph embedding features. Blocks in dotted lines are optional.

a feature that represents the neighborhood information and use this feature for autoencoder training. For this purpose, we define a **neighborhood encoding vector** (n-vector in short) for each sample as follows:

$$\mathbf{n}_i = [\mathbf{l}_1, \cdots, \mathbf{l}_j, \cdots, \mathbf{l}_k] \;\; \forall j[j = 1, ..., k \rightarrow j \in \mathcal{N}_i] \tag{6.4}$$

Here, $\mathcal{N}_i$ are the neighbors of sample i. $\mathbf{l}_j$ is the label distribution of the jth neighbor of a sample. The labels can be defined at the monophone level or senone level. We have two options to create the label distribution vector $\mathbf{l}$. The first choice is a one-hot vector representation. This is a binary vector representation with 1 representing the top-scoring class and 0 for all other classes. For training samples, the top-scoring class is determined by forced alignment of the training transcriptions with the acoustic data; for test samples, they are determined by a first-pass decoding output. The second choice is a soft-label representation. This is the full probability distribution over class labels obtained from a supervised classifier. We can either use the DNN classifier from our baseline ASR system to compute the posteriors, or we can train a separate classifier. Having obtained the n-vectors for the training samples, we train an autoencoder to map them to a more compact

representation.

In fact, the neighborhood information of a sample can be viewed as a special graph - a **weighted bipartite graph** $\mathcal{G} = \{U, V, E, \mathbf{W}\}$. A bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V. Each edge $e \in E$ connects a vertex $u \in U$ to a vertex in $v \in V$. $\mathbf{W} \in \mathbb{R}^{|U| \times |V|}$ is a weight matrix that encodes the similarity between $u \in U$ and $e \in E$. Figure 6.4 shows an example of a bipartite graph that encodes the neighborhood information of 3 samples. Each sample consists of 5 nearest neighbors and all neighbors are included in set U. Set V consists of the labels - in Figure 6.4, there are 3 labels.

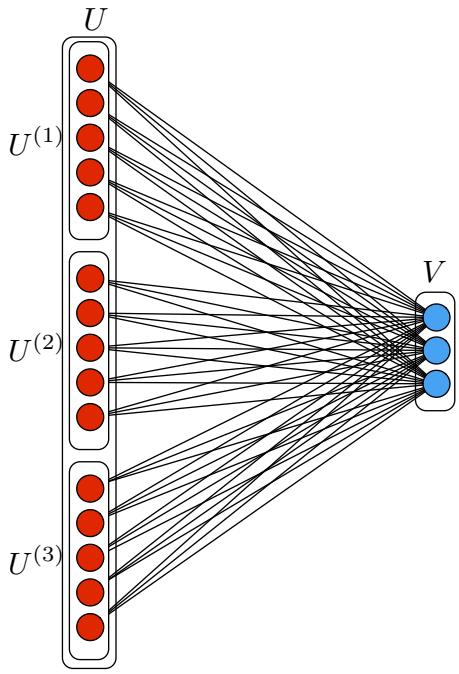


Figure 6.4: An example of a weighted bipartite graph. The nodes on the left partition (U) correspond to the k nearest neighbors for each frame; the nodes on the right partition (V) correspond to labels, which can either be monophone labels or senones.

The n-vectors are vectorized representation of the weight matrix defined on the bipartite graph. To give an intuition of how the neighborhood information can reveal the similarity

between a pair of samples, we show the 5 nearest neighbors of three different samples as bipartite graphs in Figure 6.5. These three graphs are the subgraphs in Figure 6.4, where each subgraph encodes the neighborhood information for a sample. The neighbors for each sample i are in set $U^{(i)} = \{u_1^{(i)}, \cdots, u_k^{(i)}\}$, where $u_j^{(i)}$ is j nearest neighbor of i. The labels for each neighbor are in set $V = \{v_1, \cdots, v_C\}$. The weight matrix $\mathbf{W}^{(i)}$ is shown below each graph. To compare the neighborhood information between a pair of node i and j, we can use the Frobenius norm as follows:

$$||\mathbf{W}^{(i)} - \mathbf{W}^{(j)}||_F = \sqrt{Tr((\mathbf{W}^{(i)} - \mathbf{W}^{(j)})(\mathbf{W}^{(i)} - \mathbf{W}^{(j)})^\intercal)} \tag{6.5}$$

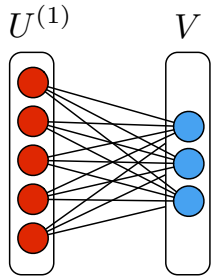
Or equivalently, we can just compute the ℓ_2 distance between $\mathbf{n}_i$ and $\mathbf{n}_j$.

In Figure 6.5, the first and third samples have similar neighborhood information: for example, the 5 nearest neighbors of the first sample are $1, 2, 1, 2, 3$, respectively; the 5 nearest neighbors of the third sample are also $1, 2, 1, 2, 3$. The ℓ_2 distance between $\mathbf{n}_1$ and $\mathbf{n}_3$ is 0.51. On the other hand, the neighbors of the first and second samples are slightly different: the 5 nearest neighbors of the second sample are $1, 1, 1, 2, 1$. Consequently, the ℓ_2 distance between $\mathbf{n}_1$ and $\mathbf{n}_2$ is 1.2.

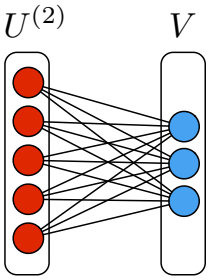
6.3.1 Dataset Compression using Landmarks

The computational bottleneck in extracting neighborhood graph embeddings lies in the extraction of n-vectors. To create the n-vectors, we need to perform a nearest neighbor search in millions of samples. To speed up the process, instead of performing kNN search in a huge amount of samples, we select a set of landmark points[1] from the entire set of samples and perform kNN search using these landmark points only. Landmarks are widely used in manifold learning. In previous work on manifold learning [126] and graph-based SSL [86], it is shown how large-scale graph construction can be rendered manageable by selecting a much smaller set of "landmark" samples instead of the entire set of samples.

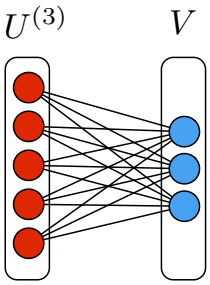
[1]The use of the term "landmark" here is unrelated to acoustic-phonetic landmarks.



	L1	L2	L3
N1	**0.8**	0.1	0.1
N2	0.1	**0.7**	0.2
N3	**0.5**	0.1	0.4
N4	0	**0.9**	0.1
N5	0.2	0	**0.8**



	L1	L2	L3
N1	**0.7**	0.1	0.2
N2	**0.5**	0.2	0.3
N3	**0.7**	0.1	0.2
N4	0.3	**0.6**	0.1
N5	**0.5**	0.4	0.1



	L1	L2	L3
N1	**0.8**	0	0.2
N2	0.2	**0.6**	0.2
N3	**0.6**	0.2	0.2
N4	0.1	**0.8**	0.1
N5	0.1	0.3	**0.6**

Figure 6.5: An example of three subgraphs from the weighted bipartite graph in Figure 6.4. Each subgraph is a neighborhood representation for a sample.

These landmarks can be obtained by uniform subsampling of the original data set, or according to a principled objective function (e.g., [93, 140, 142, 87]), or by k-means clustering. In this work, we use the k-means centroids as landmarks. We extract landmarks from the training and test samples, and denote these landmark sets as $\mathcal{Z}_{\mathcal{L}}$ and $\mathcal{Z}_{\mathcal{U}}$, respectively. As a result, we only need to perform nearest neighbor search using a set of landmarks in each set rather than searching in the entire labeled training set and unlabeled set.

6.3.2 Framework

Figure 6.6 summarizes the framework of extracting neighborhood graph embedding features. Given training data $\mathcal{L}$ and test data $\mathcal{U}$, we extract separate sets of landmarks $\mathcal{Z}_{\mathcal{L}}$ and $\mathcal{Z}_{\mathcal{U}}$ using k-means. We also select a subset of the training data to train a simple MLP, which is used to produce the soft-label representation of the n-vectors. For each sample in $\mathcal{L}$, we run

kNN search in $\mathcal{Z}^{\mathcal{L}}$ and use the MLP classifier to create the n-vectors. These n-vectors from $\mathcal{L}$ are used to train the graph embedding feature extractor (autoencoder). The hidden layer outputs are used as the embedding features $\mathbf{g}$ for $\mathcal{L}$.

For test samples, we create two n-vectors $\mathbf{n}_{\mathcal{L}}$ and $\mathbf{n}_{\mathcal{U}}$ for each sample. The resulting n-vectors are then passed through the trained autoencoder separately to derive two embedding features $\mathbf{g}_{\mathcal{L}}$ and $\mathbf{g}_{\mathcal{U}}$. In addition, we further enrich the graph embedding features with a similarity vector $\mathbf{s}$, which contains the similarity values to the neighbors. Define the acoustic feature vector as $\mathbf{a}$, which can be MFCC features or fMLLR features. The final input feature vector to the DNN takes the following form:

- Training data: $\mathbf{x} = [\mathbf{a}^{\intercal}, \mathbf{g}_{\mathcal{L}}^{\intercal}, \mathbf{g}_{\mathcal{L}}^{\intercal}, \mathbf{s}_{\mathcal{L}}^{\intercal}, \mathbf{s}_{\mathcal{L}}^{\intercal}]^{\intercal}$
- Test data: $\mathbf{x} = [\mathbf{a}^{\intercal}, \mathbf{g}_{\mathcal{L}}^{\intercal}, \mathbf{g}_{\mathcal{U}}^{\intercal}, \mathbf{s}_{\mathcal{L}}^{\intercal}, \mathbf{s}_{\mathcal{U}}^{\intercal}]^{\intercal}$

Note that in order to have the same dimensionality, the graph embedding features $\mathbf{g}$ and similarity feature vectors $\mathbf{s}$ for training samples are duplicated.

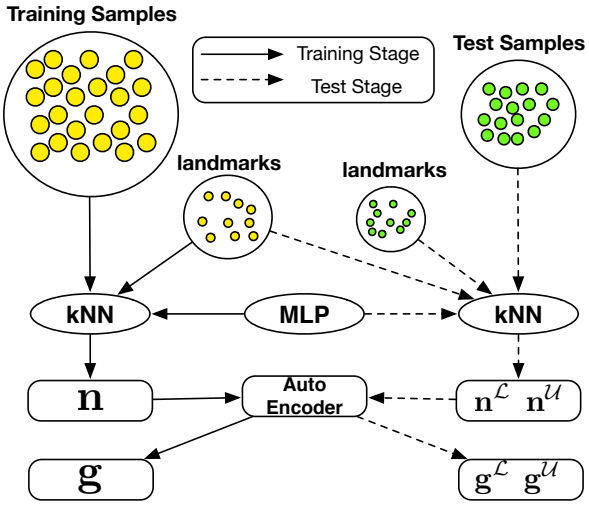


Figure 6.6: Procedure for extracting graph-embedding features.

6.3.3 Speaker-dependent Graph Embeddings

In the previous chapter, we proposed speaker-dependent graph construction. We showed that constraining the nearest neighbor search to a set of samples from a specific speaker yields better performance than using the entire set. Similarly, we can extract graph embedding features in a speaker-dependent approach.

Recall that the graph embedding features in the previous section have the following form: for training samples, we extract graph embedding features $\mathbf{g}$ using the n-vectors extracted from the landmarks $\mathcal{Z}_{\mathcal{L}}$, which are extracted from the entire training samples. For test samples, we extract graph embedding feature $\mathbf{g}_{\mathcal{L}}$ using landmarks $\mathcal{Z}_{\mathcal{L}}$, and graph embedding feature $\mathbf{g}_{\mathcal{U}}$ using landmarks $\mathcal{Z}_{\mathcal{U}}$.

For speaker dependent graph embeddings, in addition to the landmarks extracted from the entire training samples, we also extract landmarks on a per-speaker basis. In Figure 6.6, we show our framework for extracting speaker-dependent graph embedding features. We first extract a set of landmarks from the entire training samples, and we refer to them as *global* landmarks $\mathcal{Z}_{global}$. For each speaker, we also extract a set of landmarks from all samples in that speaker, and refer to this landmark set as *local* landmarks $\mathcal{Z}_{local}$. For each sample, we create two n-vectors $\mathbf{n}_{global}$ and $\mathbf{n}_{local}$, by doing nearest neighbor search in $\mathcal{Z}_{global}$ and $\mathcal{Z}_{local}$. The resulting n-vectors are passed into the autoencoder to extract two graph embeddings feature vectors $\mathbf{g}_{global}$ and $\mathbf{g}_{local}$. We also create two similarity vectors $\mathbf{s}_{global}$ and $\mathbf{s}_{local}$. The final input feature vector to the DNN takes the following form:

$$\mathbf{x} = [\mathbf{a}^{\intercal}, \mathbf{g}_{global}^{\intercal}, \mathbf{g}_{local}^{\intercal}, \mathbf{s}_{global}^{\intercal}, \mathbf{s}_{local}^{\intercal}]^{\intercal} \tag{6.6}$$

For real-time performance, speaker-dependent graph embedding features are preferred over the speaker-independent features. For speaker-independent features, we need to create landmarks from the entire test set; however, for speaker-dependent features, we can create local landmarks much more rapidly. Moreover, the global and local graph embedding features outperform the speaker-independent features, as shown in the next section.

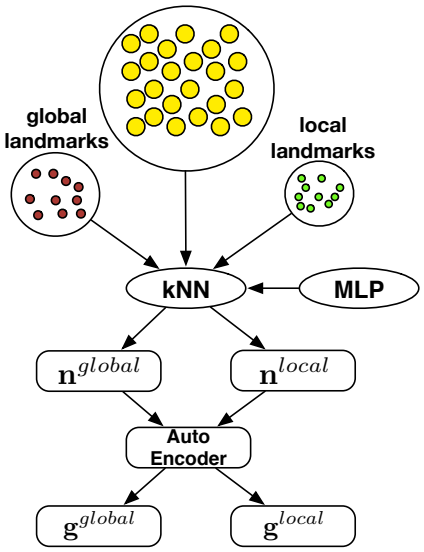


Figure 6.7: Procedure for extracting speaker-dependent neighborhood graph embedding features.

6.4 Similarity Graph Embedding Features

In the previous section, we describe a neighborhood graph embedding feature extraction procedure. These features are extracted by encoding neighborhood information into a compact feature representation using an autoencoder. The neighborhood information is local - we use the label information of the k nearest neighbors for a given sample. In this section, we try to learn a different embedding feature by encoding global manifold information.

Following [134], a direct way to encode the manifold information is to use the row vectors of a similarity matrix to train the autoencoder. Figure 6.8 illustrates the framework. First, we construct a similarity graph over all training and test samples. Then, the row vectors of the similarity matrix are used to train an autoencoder. The nonlinear activation outputs from the hidden layer are the graph embedding features. In contrast to the previous neighborhood graph embedding features, we refer to this type of embedding features as **similarity graph**

embedding features.

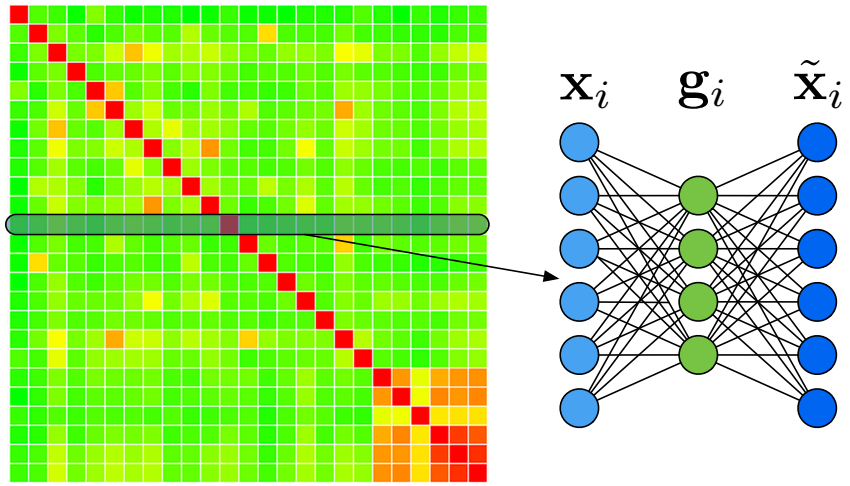


Figure 6.8: Training autoencoder using row vectors in a similarity matrix. Each row vector is the input to the autoencoder. The nonlinear activation outputs from the hidden layer are the graph embedding features.

However, this approach is not directly applicable to LVCSR tasks. First, we need to construct a similarity graph over all training and test samples, which is computationally expensive. More importantly, a new graph needs to be built on every new test set. Thus, this approach cannot be used as is for acoustic modeling. In this section, we propose an approximation framework to extract similarity graph embedding features. We leverage the notion of landmarks again to approximate the similarity matrix using low-rank approximation. The resulting framework addresses both computational bottleneck and achieve real-time performance.

6.4.1 Similarity Matrix Approximation using Landmarks

To construct a similarity matrix with n nodes, the time complexity is $O(n^2)$. On the other hand, to train the autoencoder, we need to store all the row vectors in a large similarity

matrix. In [86], an anchor graph approach was proposed. The idea of anchor graph is to approximate the large similarity matrix using a small set of "landmark" points or anchor points. The notion of landmark was also used in the previous chapter, where we extracted n-vectors by doing kNN selection using a small representative subset of "landmarks" instead of a huge amount of samples.

Formally, given a similarity matrix $\mathbf{W}$, and a set of landmarks $Z_i, i = 1, \cdots, m$, the similarity matrix can be approximated as follows:

$$\tilde{\mathbf{W}} = Z\Lambda^{-1}Z^{\intercal} \tag{6.7}$$

where $\tilde{\mathbf{W}}$ is the approximated similarity matrix of $\mathbf{W}$, $Z \in \mathbb{R}^{n \times m}$ is a low rank matrix, Λ is a diagonal matrix with $\Lambda_{kk} = \sum_{i=1}^{n} Z_{ik}$. Z is a design matrix with Z_{ij} measuring the relationship between sample i and landmark j. According to Nadaraya-Watson kernel regression [102], Z_{ij} is defined as follows:

$$Z_{ij} = \frac{K_h(\mathbf{x}_i, \mathbf{z}_j)}{\sum_{\mathbf{z}' \in \mathcal{Z}} K_h(\mathbf{x}_i, \mathbf{z}')} \tag{6.8}$$

where $\mathcal{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_m\}$ is a set of landmark points, $K_h(\mathbf{x}_i, \mathbf{z}_j)$ is a kernel function defined on a given sample $\mathbf{x}_i$ and a landmark point $\mathbf{z}_j$ (in our case, we use the RBF kernel). In this way, we can represent a large similarity matrix $\mathbf{W}$ using a much smaller low rank matrix Z. More importantly, it is not necessary to train new models whenever we have a new test set - all we need is a set of precomputed landmark points.

To train the autoencoder, we can use the row vectors in Z, which reduces the space complexity from $O(n^2)$ to $O(mn)$, where $m << n$. In our experiment for LVCSR, n is usually in the range of 10^7 whereas m is usually in the range of 10^3, which reduces the data size by 4 orders of magnitude. After the autoencoder has been trained, we can create the graph embedding features $\mathbf{g}$. To use these features, we augment the existing acoustic features $\mathbf{a}$ as follows:

$$\mathbf{x} = [\mathbf{a}^{\intercal}, \mathbf{g}^{\intercal}]^{\intercal} \tag{6.9}$$

6.5 Data and Systems

We evaluate the framework on two large vocabulary conversational speech recognition (LVCSR) tasks: SVitchboard-II [87] task and 110-hour Switchboard-I. SVitchboard-II is a set of high-quality, low-complexity conversational English speech benchmark datasets created from the original Switchboard-I dataset. The corpora were selected to be acoustically rich, while limited in vocabulary size. Compared to the original Switchboard-I corpus with 30k vocabulary, the SVitchboard-II corpora have different vocabulary sizes ranging from 50 to approximately $10,000$. We use the largest-vocabulary task in SVitchboard-II for this study, which has a vocabulary size of 9983 and refer to it as SVB-10k. The training, development, and test sizes are 67642, 8491, and 8503 utterances, which correspond to 69.1 hours, 8.8 hours and 8.8 hours, respectively. A trigram backoff language model built on the training data is used for decoding. For the Switchboard task, we use the first 100k utterances from the entire Switchboard-1 Phase 2 pack, and the resulting training data consists of 110 hours. We also use another 4k utterances as development set. For evaluation purposes, we use the Eval2000 (Hub5-00) set. The Eval2000 set consists of 20 conversation sides from Switchboard (SWB), and another 20 conversation sides from CallHome English (CHM). We report the performance on the Eval2000-SWB part. A trigram language model is trained on the Switchboard dataset (excluding Fisher corpus). We train several different baseline systems for the two tasks.

6.5.1 Speaker-Independent (SI) DNN

The SI DNN system is trained using the same procedure as described in Section 5.2. The resulting DNN network consists of 4 hidden layers with *tanh* function: for SVB-10k task, each layer consists of 1024 nodes; for Switchboard, each layer consists of 1200 nodes. The input features to the DNNs are spliced MFCCs (with a context window size of 4), followed by an LDA transformation (without dimensionality reduction) which is used to decorrelate the input features. The resulting feature vector has 117 dimensions in total. The output

layer consists of sigmoid units, and the total number of senones in the DNN for these tasks is 1864 and 2390, respectively. A softmax function is used to produce senone posteriors. We perform greedy layer-wise supervised training [10, 124] on multiple CPUs and GPUs using the Kaldi toolkit [111]. We use 20 epochs to train the DNN, with a mini-batch size of 256. For the first 15 epochs, we decrease the learning rate from 0.01 to 0.001 and fix the learning rate at 0.001 for the last 5 epochs.

6.5.2 Speaker-Dependent (SD) DNN

To train a SD-DNN, we use the features from a speaker adaptive trained (SAT) GMM-HMM system. The SAT GMM-HMM system uses 40-dimensional fMLLR features. To get the features, the MFCC features are first spliced with a context window size of 9 ($\pm$ 4 frames). They are then projected down to 40 dimensions using LDA, followed by a semi-tied covariance transform. Then, the fMLLR transform is applied to the features for each conversation side. We use the 40-dimensional fMLLR features to the DNN and refer to this baseline as "SD-DNN (40)". In addition, we further splice the fMLLR features with another context window size of 9 ($\pm$ 4 frames), and refer to it as "SD-DNN (360)". Table 6.1 shows the baseline DNN-HMM system's word error rate.

Systems	SVB-10k		Switchboard	
	Dev WER	Eval WER	Dev WER	Eval WER
SI-DNN	32.04%	32.17%	28.7%	23.7%
SD-DNN (40)	29.96%	30.01%	27.9%	22.0%
SD-DNN (360)	27.96%	27.97%	25.4%	20.0%

Table 6.1: Baseline system word error rates for SVB-10k and Switchboard.

6.6 Experimental Results

6.6.1 Standard GBL Experiment

We first evaluate the standard graph-based SSL framework described in Section 5.2 for SVB-10k tasks. The baseline system we used for the task is SI-DNN system. In line with Section 5.2, we use the bottleneck features from the baseline DNN and splice them with a context window of 9. The resulting frames (excluding silence frames) of the training and test data $\mathcal{L}$ and $\mathcal{U}$ are used to build a kNN graph. Due to the huge size of the graph (for SVB-10k, we have 17 million non-silence frames), we perform a uniform subsampling to select 10% of the samples from the training set. For each sample in $\mathcal{U}$ we select 10 nearest neighbors in $\mathcal{L}$ and 10 in $\mathcal{U}$. After the graph is constructed, we run the pMP algorithm to obtain a new set of senone posteriors. The new posteriors are converted to "graph likelihoods" by dividing the state priors, and the likelihoods are interpolated with the original scores in the lattice.

All parameters are tuned on the development data. The parameters of pMP are $\mu = 10^{-6}$ and $\nu = 8 \times 10^{-6}$; the linear score combination weights are 0.2, 0.8, and 8 for the graph likelihood score, acoustic likelihood score and language model score, respectively. Table 6.2 shows the performance of the standard GBL framework. It can be observed that the frame accuracy on the development set is improved from 32.5% to 38.2%, a 5.7% absolute. However, the significant improvement in the frame accuracy does not lead to a better word error rate. On both the development and evaluation set, the word error rates are decreased only by a small margin. This again demonstrates a major disadvantage of the standard GBL - while GBL optimizes the frame level accuracy according to the graph-based SSL objective, it does not take other ASR components into consideration. Moreover, the whole procedure cannot be done in real-time: the graph construction takes up to days on multiple CPUs.

6.6.2 Experiment: Comparison of Different Neighborhood Graph Embeddings

First, we compare different schemes to extract neighborhood graph embedding features on the SVB-10k task. For the experiment we use the same bottleneck layer feature representation of

	Dev WER	Dev Frame Acc.	Eval WER
DNN Baseline	32.04%	32.5%	32.17%
+standard GBL	31.95%	38.2%	32.13%

Table 6.2: Results for standard GBL framework.

the data as in the preceding section. The number of frames in the training data is around 25 million. We extract landmarks from this set using k-means clustering. First, we assign each frame a phone label based on the forced alignment of training transcriptions. For silence frames we run k-means with $k = 64$. For the remaining non-silence frames, we run k-means with $k = 32$ for each of the 42 non-silence phone classes. The resulting centroids are used as landmarks, resulting in 1408 landmarks, which is 4 orders of magnitude less than the original graph size. For each set (training, development and test), we create a separate set of landmarks. We perform kNN search using the landmark sets. The similarity measure we used is an RBF kernel with Euclidean distance. For the training samples we select 10 nearest neighbors in the landmark set of $\mathcal{L}$. For the test data we select 10 nearest neighbors in the landmark sets of $\mathcal{L}$ and $\mathcal{U}$, respectively.

In particular, we compare the following setups in Table 6.3:

- Graph-embedding features with different nearest neighbor encoding schemes: 1) one-hot label distribution vector and 2) soft-encoding label distribution vector (#1, #2 vs #4, #5);

- Different feature integration scheme: 1) early integration and 2) late integration (#1 vs #3) and #4 vs #6;

- Graph embedding features with different dimensionality (#1 vs #2, #4 vs #5);

- Raw n-vectors input versus embedding features (#7 vs #4): in order to evaluate the effect of the usage of an autoencoder, we directly use the raw n-vectors as inputs to the

DNN, and compare the performance to the DNN trained with the embedding features obtained from an autoencoder.

To generate the label-encoding vector $\mathbf{l}$, we experiment with both a one-hot and a soft-label vector representation. The labels are those of the 43 phone classes. For a one-hot encoding the frame-level forced alignment information is used for the training data and the first-pass decoding output for the test data. To create soft-label neighborhood encoding vectors, we train a simple 3-layer multi-layer perceptron (MLP) with ℓ_2 regularization for phone classification using the Theano [6] toolkit. The hidden layer of the MLP consists of 2000 nodes. The nonlinearity function in the MLP is the *tanh* function. We train the MLP on a random 10% subset of the entire training data using stochastic gradient descent with a minibatch size of 128. The learning rate of the MLP is 0.01 and the total number of epochs is 1000. The regularization constant is set to 0.001. We use early stopping during training. The MLP has a classification accuracy of 67% on a 1.2 million held-out dataset. The MLP outputs are then used to generate phone label distributions for the landmark samples.

Setup	Description
#1	One-hot; 43-d embedding features; early integration
#2	One-hot; 100-d embedding features; early integration
#3	One-hot; 43-d embedding features; late integration
#4	Soft label; 43-d embedding features; early integration
#5	Soft label; 100-d embedding features; early integration
#6	Soft label; 43-d embedding features; late integration
#7	Soft label n-vectors; early integration

Table 6.3: Experimental setup for neighborhood graph embedding.

Table 6.4 shows the word error rates (WER) on the development and test sets using the different setups. For diagnostic purposes we excluded 300 sentences from the training

data and randomly selected 4000 frames as validation frames, and computed the frame-level accuracy of the HMM state label predictions on this set. Bold-face numbers indicate statistically significant improvements ($p < 0.05$).

We first compare different weighting schemes for the neighborhood encoding vectors. The neighborhood encoding vectors in Equation 6.4 can be designed in two ways: one way is to use one-hot vector encoding for $\mathbf{l}$ in Equation 6.4; the other way is to use soft-label assignment for $\mathbf{l}$. Comparing Experiment #1 versus #4, and Experiment #2 versus #5, we observe that using soft-label assignment for the n-vectors yields consistently better validation frame accuracy and the word error rate. Both the one-hot encoding vectors and soft-label assignments achieve a significant better validation frame accuracy; however, only the n-vectors using soft-label assignments show a significant improvement in the word error rate. The reason why improvements are not larger may be due to the way we derived the one-hot encodings: the labels of the training data are derived using forced alignments whereas the labels of the test data are derived using first-pass decoding output. The discrepancy between the label accuracies could be a contributing factor.

Secondly, we compare different feature integration schemes. In Section 6.2, we propose both early stage feature integration as well as late stage feature integration. The details are shown in Figure 6.2 and Figure 6.3. For SI-DNN system on SVB-10k, we use the first two modules (initial splicing and LDA transformation) in the figures for preprocessing. Comparing Experiment #1 with #3 and #4 with #6, the early stage feature integration yields significantly better validation frame accuracy (49.05% versus 46.1%) and word error rate. In Experiment #4, we achieve a WER of 30.57% on the development set using early stage feature integration; on the contrary, the late stage feature integration results in a 31.95% WER, more than 1% absolute increase.

We then compare graph embedding features with different dimensionalities (Experiment #1 vs #2, Experiment #4 vs #5). To obtain a different dimensionality of the embedding features, we try different numbers of hidden units in the autoencoder. In addition to 43 units in the hidden layer, we also increase it to 100 units. From the experiments, it can be

concluded that the dimensionality of the hidden layer in the autoencoder (Experiment #1 vs. #2 and #4 vs. #5) does not have a consistent effect.

Finally, we also use the raw n-vectors as additional input features to the DNNs. Comparing Experiments #4 and #7, the raw concatenated n-vectors only have marginal improvements over the baseline system (46.1% validation accuracy over 45.4% baseline accuracy, and 31.75% WER over 32.04% baseline WER). The system using embedding features extracted from the autoencoder yields much better validation accuracy and word error rate. This is because the autoencoder performs a nonlinear dimensionality reduction over the raw inputs; a DNN trained with the raw n-vectors is more prone to overfit the n-vector part which has hundreds of dimensions compared to the much smaller acoustic features.

WER	Dev	Test	Valid Acc.
DNN Baseline	32.04%	32.17%	45.4%
#1	**31.65%**	32.05%	**49.05%**
#2	31.76%	31.98%	**47.7%**
#3	31.76%	32.00%	46.1%
#4	**30.57%**	**30.59%**	**51.95%**
#5	**30.61%**	**30.71%**	**51.05%**
#6	31.95%	32.07%	45.73%
#7	31.75%	**31.79%**	46.1%

Table 6.4: Results for standard GBL framework vs. neighborhood graph embedding features.

In the following experiments, we adopt the optimal configuration of the neighborhood graph embedding features as follows:

- We use the soft-label encoding method to create the n-vectors;
- We use the early stage feature integration to combine graph embedding features with acoustic features.

6.6.3 Experiment: SI-DNN system with neighborhood graph embedding features

SVitchboard-10k task

We first evaluate the performance of the SI-DNN system using the graph embedding features on the SVB-10k task. In addition to the SI-DNN baseline system in Table 6.1, we add several additional competitive SI-DNN systems for further evaluations. We increased the number of hidden units per layer in the DNN from 1024 to 2048, and the number of hidden layers from 4 to 6. Additionally, we trained a SI-DNN using maxout unit [154], which tends to yield better performance. For the maxout network, we used a group size of 5 and set the number of groups to 800. The value of p of the p-norm was set to 2. The graph embedding features are fused with the acoustic features using the early stage feature integration scheme. We splice the input features with a context window size of 9. An LDA without dimensionality reduction is performed on top of the spliced features for decorrelation. The final resulting feature has a dimensionality of $9 \times (13 + 43 + 43 + 10 + 10) = 1071$.

Table 6.5 shows the experimental results using graph embedding features on top of SI-DNN systems. We can observe that augmenting the original acoustic features with graph embedding features yields consistent and significant improvements for all baseline systems. The absolute WER improvement over various conventional DNN system varies from 0.9% to 1.6%, with a relative improvement ranging from 3% to 5%. For the first SI-DNN system, adding graph embedding features achieves a similar WER to the SD-DNN(40) system. This shows that the graph embedding features can be used as an adaptation technique.

Increasing the number of nodes and layers leads to a better performance (2nd and 3rd SI-DNN systems); nevertheless, adding graph embedding features achieves additional gain. The gain becomes smaller as we use a deeper network, e.g. the 3rd system with 6 hidden layers. The best baseline performance among all systems on the development is achieved by the maxout network. Nevertheless, the combination with graph-embedding features still reduces the WER further by about 0.5% absolute.

It is worth noting that the input layer of the DNN when using the graph-embedding fea-

	Dev	Eval
(1024) 4-layer DNN	32.04%	32.17%
+ neighborhood graph embedding	**30.57%**	**30.59%**
(2048) 4-layer DNN	31.23%	31.31%
+ neighborhood graph embedding	**29.59%**	**29.72%**
(2048) 6-layer DNN	30.61%	30.76%
+ neighborhood graph embedding	**29.65%**	**29.85%**
Maxout	30.63%	30.86%
+ neighborhood graph embedding	**30.18%**	**30.32%**

Table 6.5: Neighborhood graph embedding features for different SI-DNN baseline systems on SVB-10k. The number in parentheses is the number of nodes per hidden layer. Numbers in bold face indicate a significant improvement at $p = 0.05$.

tures is generally larger than when using only acoustic features, resulting in more parameters in the new DNN system. For a controlled experiment, we increased the size of the baseline DNN by increasing the number of nodes in the hidden layer, and made the number of parameters roughly the same for both networks. To do this, we increased the number of nodes per layer from 1024 to 1145, resulting in 6.16 million parameters in total for the baseline DNN. The SI-DNN baseline system has 6.15 million parameters. Table 6.6 shows the results of an experiment where we controlled the overall number of parameters. Comparing the two systems, the system using graph embedding gave a 1.4% absolute improvement on the test set.

Switchboard-I 110-hour task

We also evaluate the performance on the Switchboard 110-hour task using the same SI-DNN system. For Switchboard-I task, we select 1504 landmarks in total (64 landmarks for

	Dev	Test
(1145) 4-layer DNN, # params: 6.16M	31.89%	32.01%
+ neighborhood graph embedding, # params: 6.15M	**30.57%**	**30.59%**

Table 6.6: WER comparison controlled for the number of parameters. The number in parentheses is the number of nodes per hidden layer. Numbers in bold face indicate a significant improvement at $p = 0.05$.

the silence phone, and 32 landmarks for each one of the remaining 45 phones). Table 6.7 shows the system performance using graph embedding features on Switchboard. The SI-DNN system achieves a WER of 23.7% on the evaluation set; graph embedding gives a boost of 0.8% in absolute WER reduction. In the following section, we also investigate the performance using stronger SD-DNN systems. Nevertheless, the experiments indicate that the graph embedding features provide auxiliary information in addition to the acoustic features, and lead to consistent improvements on SI-DNN systems.

	Dev	Eval
(1200) 4-layer DNN	28.7%	23.7%
+ neighborhood graph embedding	28.3%	**22.9%**

Table 6.7: Graph embedding features for SI-DNN systems on Switchboard 110-hour task. The number in parentheses is the number of nodes per hidden layer. Numbers in bold face indicate a significant improvement at $p = 0.05$.

6.6.4 Experiment: SD-DNN system with neighborhood graph embedding features

In the previous experiment, the graph embedding features were extracted in a speaker-independent approach. The landmarks were extracted from the entire training or test set. In this section, we investigate the speaker-dependent (SD) neighborhood graph embedding

features. We also evaluate the performance of using neighborhood graph embedding features on top of a SD-DNN system.

First, we use speaker-dependent graph embedding features on top of a speaker-independent DNN baseline system. Table 6.8 and Table 6.9 show the results of two SI-DNN systems, and the performance using two different graph embeddings. The SI graph embedding features reduce the WER from 32.17% to 30.59% on SVB-10k, and it reduces the WER from 23.7% to 22.9% on Switchboard. The DNN systems using SD graph embedding features achieve further improvement. The WER on SVB-10k is improved to 30.18%, which is a 2% absolute and a 6.2% relative improvement over the baseline system. The WER on Switchboard is improved to 22.5%, which is a 1.2% absolute and a 5% relative improvement over the baseline. The results indicate that the SD neighborhood graph embedding is explicitly performing adaptation.

	Dev	Eval
SI-DNN	32.04%	32.17%
+ neighborhood graph embedding (SI)	**30.57%**	**30.59%**
+ neighborhood graph embedding (SD)	**30.03%**	**30.18%**

Table 6.8: Comparison of SI and SD neighborhood graph embedding features for SI-DNN systems on SVB-10k. Numbers in bold face indicate a significant improvement at $p = 0.05$.

In the second set of experiments, we use the speaker dependent DNN systems as baselines. Table 6.10 shows the results of two SD-DNN systems, and the performance using graph embedding features. The SD-DNN(40) system improves the SI-DNN system by 2% absolute. Additional splicing on the fMLLR features gives another improvement of 2% absolute. Using graph embedding features, we achieve a consistent reduction in WER in both systems. However, the graph embedding features extracted in the speaker-independent approach only give a slight improvement. In SD-DNN(360), the improvement is actually negligible. On the other hand, the graph embedding features extracted in the speaker-dependent approach

	Dev	Eval
SI-DNN	28.7%	23.7%
+ neighborhood graph embedding (SI)	28.3%	**22.9%**
+ neighborhood graph embedding (SD)	**27.6%**	**22.5%**

Table 6.9: Comparison of SI and SD neighborhood graph embedding features for SI-DNN systems on Switchboard 110-hour. Numbers in bold face indicate a significant improvement at $p = 0.05$.

give a much stronger boost. In both SD-DNN(40) and SD-DNN(360) system, we achieve 1% absolute reduction in WER on both development and evaluation set. This indicates that in order to accommodate a speaker-dependent DNN system, which uses transformed features on a per speaker basis, we also need to extract neighborhood graph embedding features in a speaker-dependent approach.

	Dev	Eval
SD-DNN (40)	29.96%	30.01%
+ neighborhood graph embedding (SI)	**29.40%**	**29.60%**
+ neighborhood graph embedding (SD)	**28.79%**	**29.06%**
SD-DNN (360)	27.96%	27.97%
+ neighborhood graph embedding (SI)	27.90%	27.93%
+ neighborhood graph embedding (SD)	**26.78%**	**26.90%**

Table 6.10: Comparison of SI and SD neighborhood graph embedding features for SD-DNN systems on SVB-10k. Numbers in bold face indicate a significant improvement at $p = 0.05$.

Finally, we report the performance on the Switchboard 110-hour task. In Table 6.11, the graph embedding features give a consistent reduction in WER in SD-DNN(40) system. The

speaker-dependent graph embedding features achieve a significant improvement ($p = 0.05$) on the development set, although the improvement on the evaluation set is marginal, from 22.0% to 21.8%. In SD-DNN(360) system, the speaker-independent graph embedding features does not further improve the WER; on the other hand, the speaker-dependent graph embedding features reduce the WER from 25.4% to 24.6% on the development set, and from 20.0% to 19.5% on the evaluation set, which corresponds to a 3.1% and 2.5% relative improvement, respectively.

	Dev	Eval
SD-DNN (40)	27.9%	22.0%
+ neighborhood graph embedding (SI)	**27.4%**	21.8%
+ neighborhood graph embedding (SD)	**26.9%**	21.8%
SD-DNN (360)	25.4%	20.0%
+ neighborhood graph embedding (SI)	25.2%	20.2%
+ neighborhood graph embedding (SD)	**24.6%**	**19.5%**

Table 6.11: Comparison of SI and SD neighborhood graph embedding features for SD-DNN systems on Switchboard 110-hour. Numbers in bold face indicate a significant improvement at $p = 0.05$.

6.6.5 Experiment: Similarity graph embedding features

In this section, we compare the similarity graph embedding features to the neighborhood graph embedding features on the same tasks. The goal here is to evaluate the performance of different graph embedding features on LVCSR acoustic modeling.

To extract the similarity graph embedding features, we need to create a low rank matrix Z. Here, we use the same set of landmarks as in the previous experiments to ensure a fair comparison. The autoencoder has the same number of units in the hidden layer. After the

autoencoder has been trained, we can create a similarity graph embedding features.

We compare the neighborhood graph embedding features and similarity graph embedding features using both SI-DNN baseline system and SD-DNN baseline system. Table 6.12 shows the performance of similarity graph embedding features in SVB-10k task. In the SI-DNN system, the SI neighborhood graph embedding features achieve an absolute reduction in WER of 1.5%. The similarity graph embedding features can further reduce the word error rate by an additional 2% absolute. In the SD-DNN system using 40-dimensional fMLLR features, the SD neighborhood graph embedding features can reduce the WER by 1% absolute; the similarity graph embedding features can further reduce the WER by another 0.5%. In the last SD-DNN system, similarity graph embedding feature can achieve a similar word error rate while using fewer parameters in the network. For example, the number of parameters in the DNN system using similarity graph embedding features is only 90% of the number of parameters in the DNN using neighborhood graph embedding features.

We also show the similarity graph embeddings on the Switchboard 110-hour task in Table 6.13. In the SI-DNN system, the best neighborhood graph embedding features achieve an absolute reduction in WER of 1.2%. The similarity graph embedding features can reduce the word error rate by 2.6% absolute, an 11.0% relative improvement. In the SD-DNN system, the SD neighborhood graph embedding features achieve an absolute reduction in WER of 0.5%. The similarity graph embedding features can achieve the similar performance while using fewer parameters in the DNN system.

Comparing to neighborhood graph embeddings, similarity graph embeddings achieve consistently better performance with fewer number of parameters in the DNN system. In addition, neighborhood graph embedding features are a somewhat ad-hoc representation of local neighborhood information. By contrast, the similarity graph embedding features have a more solid theoretical foundation and a tight connection to spectral clustering.

	Dev	Eval
SI-DNN	32.04%	32.17%
+ neighborhood embedding (SI, 6.1M parameters)	**30.57%**	**30.59%**
+ similarity graph embedding (5.5M parameters)	**28.55%**	**28.60%**
SD-DNN (40)	29.96%	30.01%
+ neighborhood embedding (SD, 5.3M parameters)	**28.79%**	**29.06%**
+ similarity graph embedding (5.2M parameters)	**28.39%**	**28.53%**
SD-DNN (360)	27.90%	27.97%
+ neighborhood embedding (SD, 6.5M parameters)	**26.78%**	**26.90%**
+ similarity graph embedding (5.9M parameters)	**26.79%**	**26.80%**

Table 6.12: Comparison of neighborhood graph embedding features and similarity graph embedding features for SI-DNN/ SD-DNN systems on SVB-10k.

6.6.6 Experiment: Improving Front-end Features for LSTM-CTC using similarity graph embedding features

In this section, we evaluate graph embedding features in a LSTM-CTC-based ASR system. The DNN is used as the acoustic model to compute the emission probability in the HMM. Usually, the cross entropy objective is used as the training criterion. However, the frame-level independence assumptions are inherently too strict for speech recognition. More recently, researchers have been investigating novel architectures that do not depend on the HMM assumption. The connectionist temporal classification architecture [42, 44, 119, 96], or CTC, has been successfully applied in ASR. CTC is a sequence modeling technique that does not rely on frame-level independence assumptions. Combined with recurrent neural networks with long short-term memory cells, the new model has achieved considerable success in recent years. More importantly, a CTC-based model significantly reduces the complexity of training an ASR system. Most deep models still depend on a GMM-based system, in which

	Dev	Eval
SI-DNN	28.7%	23.7%
+ neighborhood graph embedding (SI, 8.5M parameter)	28.3%	**22.9%**
+ neighborhood graph embedding (SD, 8.5M parameter)	**27.6%**	**22.5%**
+ similarity graph embedding (7.8M parameter)	**26.4%**	**21.1%**
SD-DNN (360)	25.4%	20.0%
+ neighborhood graph embedding (SI, 8.8M parameter)	25.2%	20.2%
+ neighborhood graph embedding (SD, 8.8M parameter)	**24.6%**	**19.5%**
+ similarity graph embedding (8.1M parameter)	**24.6%**	**19.5%**

Table 6.13: Comparison of neighborhood graph embedding features and similarity graph embedding features for SI-DNN/ SD-DNN systems on Switchboard-I.

the forced alignments are used to initialize the training of a deep model. The quality of the alignment has a big impact on training the deep model. To train good GMM models, we need to spend lots of effort in developing different models such as monophone models, context-dependent triphone models. We also need expertise in linguistic knowledge to build decision trees for tied triphone states, and lots of feature preprocessing such as LDA, MLLT and fMLLR. With a CTC-based model, we can reduce the complexity in training a fully-fledged ASR system at different stages, thus achieving an end-to-end ASR architecture.

While LSTM-CTC-based ASR shows promising results on various LVCSR tasks, the studies of front-end features have been limited to filterbank features, MFCC features or fMLLR features. In the following experiment, we investigate how to leverage this novel ASR architecture with the graph embedding features.

First of all, we train a LSTM-CTC-based system on SVB-10k task using EESEN [96]. We use bidirectional LSTMs as the model and train it with the CTC objective function. The building block of the RNN is long short-term memory (LSTM) cells. As baseline systems, we

train a 2-layer and a 4-layer bidirectional LSTM network. Each layer contains 320 memory cells in the forward layers, and 320 memory cells in the backward layers. For CTC objective training, we use the phonemes as the CTC labels (45 phonemes plus a blank symbol). The input features to the LSTM network are 40-dimensional fMLLR features, with mean and variance normalization applied for each conversation side. In Table 6.14, the WER using LSTM-CTC is 31.84% using 2-layer network, and 28.47% using 4-layer network on the SVB-10k development set. On the other hand, a 4-layer DNN-HMM system can achieve a WER of 27.90%.

Next, we explore the possibility of integrating graph embedding features with the original fMLLR features. In our experiment, we use the similarity graph embedding features, and append them to the fMLLR features. After the features have been obtained, we apply mean and variance normalization on a per conversation-side basis. In Figure 6.9, we show the CTC token accuracy with different numbers of training epochs in different baseline LSTM-CTC systems. The improved front-end features using graph embeddings consistently achieve a higher token accuracy than the original front-end features. In Table 6.14, the WER of the system using graph embedding features is reduced by 1% absolute in the 2-layer network, and by 0.8% absolute in the 4-layer network. While it remains an open question of front-end feature engineering for LSTM-CTC-based systems, we show that the graph embedding features can be used as a rapid adaptation technique to improve the conventional front-end features. To our knowledge, this is the first feature augmentation based adaptation technique in a LSTM-CTC-based system.

6.7 Connections to Speaker Adaptation

In this section, we describe the connection between acoustic modeling with neural graph embeddings and speaker adaptation. The neural graph embedding approach is closely related to different adaptation techniques for DNN-based acoustic model training.

Here, we describe different DNN adaptation techniques used in the previous literatures. Generally speaking, the DNN adaptation techniques can be categorized into following types:

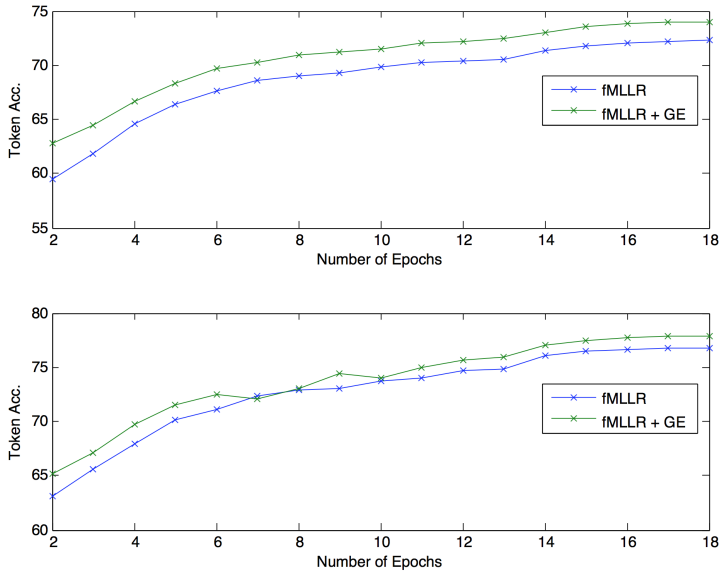


Figure 6.9: Token accuracy using LSTM with CTC-objective training. The x-axis is the number of epochs, and the y-axis is the token (phoneme) accuracy on the development set. The original front-end features are 40-dimensional fMLLR features. The improved front-end features are obtained by concatenating graph embedding features to fMLLR features (fMLLR + GE). Top figure corresponds to the 2-layer LSTM-CTC system; bottom figure corresponds to the 4-layer LSTM-CTC system.

1. Conservative training [14]

   Usually, given a limited amount of adaptation data, the DNN model tends to overfit the adaptation data. Conservative training aims at adding regularization to the adapted model training. This can be achieved by adding ℓ_2 regularization, or adapting to selected weights in the DNN [81].

2. Model transformation based adaptation [103, 36, 80, 150]

   In model transformation approach, the network is augmented with an affine transformation network to the input layer, hidden layer or the output layer. The parameters

	Dev	Eval
(320) 2-layer LSTM-CTC	31.84%	31.27%
+ similarity graph embedding	**30.22%**	**30.20%**
(320) 4-layer LSTM-CTC	28.47%	28.44%
+ similarity graph embedding	**27.66%**	**27.60%**

Table 6.14: Similarity graph embedding features for LSTM-CTC systems on SVB-10k. The number in parentheses is the number of nodes per hidden layer.

in the affine transformation networks are usually trained while fixing the rest of the networks. Thus, only a few parameters are adapted using limited adaptation data.

3. Feature based adaptation [123]

   In GMM-HMM based system, speaker adaptation is achieved using maximum likelihood linear regression (MLLR), and feature space MLLR (fMLLR). These features are used in DNN-based systems, and usually outperform the model trained with raw acoustic features.

4. Subspace-based adaptation [27]

   In subspace-based approach, a speaker subspace is obtained. To adapt to a new speaker, the adapted model is approximated by a linear combination of the bases in the subspace.

5. Regularization-based adaptation [152]

   In the regularization-based approach, a KL divergence penalty is used in addition to the cross entropy training. The KL divergence term forces the senone distribution estimated from the adapted model to be close to the senone distribution from the unadapted model.

6. Feature augmentation based adaptation [122, 2, 1, 149]

   Under the feature augmentation approach, a speaker-specific feature is learned. Typically, i-vectors and other speaker codes are used. These features are appended to the original acoustic feature vectors as inputs. Compared to the raw acoustic feature vectors, these augmented features explicitly incorporate the speaker information into the DNN training.

The neural graph embedding approach falls into the last category. The difference between the graph embedding features and other speaker-dependent features such as i-vectors and speaker codes is the feature integration. In i-vectors and speaker codes approaches [122, 2, 1, 149], the speaker-specific features are learned at the speaker level. For each speaker or conversation side, we extract one specific feature, and duplicate it to every frame. In our approach, we are learning frame-level speaker-specific embedding features. In other work [95, 156], the acoustic features are also augmented by additional information such as prosodic features and signal-to-noise ratios at the frame-level. Nevertheless, our proposed approach is the first approach that uses manifold information as DNN inputs.

Among i-vectors, speaker codes and other additional side information, our neural graph embeddings bear similarity to i-vectors [25, 122]. I-vectors are usually extracted using a special GMM, which is referred to as the universal background model (UBM-GMM). I-vectors are used to measure how an UBM-GMM should be adapted in an affine subspace in order to capture the variability of the underlying speech segments. I-vectors can be extracted at either utterance or speaker level. Similar to the graph embedding approach, the i-vector extractor needs to compute a set of UBM centroids, which is analogous to the landmark set we are using for graph embedding features. The difference is that i-vectors capture speaker variability at the speaker/utterance level; whereas the graph embedding features capture the manifold information at the frame level. The feature extractor is also different: for the i-vector extractor, we find a linear subspace; for the graph embedding feature extractor, we use an autoencoder that finds a nonlinear embedding.

6.8 Summary

In this chapter, we have described a novel neural graph embedding framework for DNN-based acoustic modeling. We have demonstrated improved scalability and better optimization, compared to the standard graph-based semi-supervised learning approach. We have proposed two different graph embedding frameworks that encode information at both the local neighborhood level (neighborhood graph embeddings) and the global manifold level (similarity graph embeddings). We have evaluated the efficacy of the framework on two LVCSR tasks and have shown that the graph embedding features can improve the WER in both SI-DNN and SD-DNN system. In addition, we have demonstrated that the similarity graph embeddings consistently yield superior performance than the neighborhood graph embeddings. Finally, we have shown that the similarity graph embeddings can be used as rapid adaptation techniques in a novel LSTM-CTC-based ASR system.

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Contributions

In this thesis, we address how to use graph-based SSL for acoustic modeling. In retrospect, the contributions of this thesis can be summarized as follows:

1. **Prior-regularized Measure Propagation (pMP) Algorithm**

   We propose a new graph-based semi-supervised learning algorithm based on [130]. The proposed algorithm allows us to explicitly inject prior knowledge in the learning objective. We show that the pMP algorithm significantly improve the phonetic classification tasks compared to other state-of-the-art graph-based SSL algorithms including label propagation, measure propagation and modified adsorption.

2. **Graph-based SSL with Variable-length Inputs**

   In conventional graph-based SSL, each node is associated with a fixed-length feature vector. We address the problem where each node is associated with a variable-length speech segment. To deploy graph-based SSL in such scenarios, we propose several graph construction techniques that map the variable-length speech segments into fixed-length feature vectors. We show strong performance on segment-level phonetic classification tasks with different graph construction techniques.

3. **Efficient Feature Selection for Speech Processing**

   We address a computational challenge that arises from variable-length input vectors. To extend the graph-based learner to variable-length speech inputs, we use the Fisher

kernel method, which maps a variable-length input vector into a fixed-length, but high-dimensional acoustic feature vector. We propose a novel scalable feature selection method based on submodular function optimization. We show that instead of using hundreds of thousands of features in the Fisher score vectors, we can reduce the feature space by 3 orders of magnitude without compromising the performance of the graph-based SSL.

4. **Improved Graph Construction for DNN-based System**

   We explore different feature representations for graph construction in DNN-based acoustic models. The features we extract are directly relevant to the DNN system. We show that several better feature representations can be obtained which result in significantly better frame-level accuracy, as well as a reduced word error rate (WER), compared to the standard input acoustic features.

5. **Lattice-based Integration Framework for Graph-based SSL in Acoustic Modeling**

   We show a lattice-based framework that integrates graph-based SSL into a fully-fledged ASR system. To our knowledge, all previous work on graph-based SSL for acoustic modeling has been limited to simple artificial tasks such as TIMIT phoneme classification using a weak GMM-based model. It is yet to discover if the graph-based SSL method can improve a DNN-based acoustic model, and it is even more critical to design an integration framework that allows graph-based SSL to be deployed in a fully-fledged ASR system. We demonstrate the performance of the framework on phone recognition as well as continuous word recognition tasks.

6. **Neural Graph Embedding Approach to Acoustic Modeling in LVCSR Systems**

   We address a couple of major caveats of graph-based SSL which hampers the appli-

cation of graph-based SSL methods in acoustic modeling for LVCSR tasks: 1) computational challenge and 2) lack of joint optimization with other system components. We propose a novel neural graph embedding approaches. Neural graph embedding features are produced by an autoencoder that maps graph structures defined over the speech signal to a continuous vector space. The resulting compact feature representation is then used to augment the original acoustic features. We propose two different neural graph embedding methods, one based on a local neighborhood graph encoding, and another based on global similarity graph encoding. We evaluate the graph embedding approach in DNN-HMM-based and LSTM-CTC-based ASR systems on two medium-to-large vocabulary conversation speech recognition tasks and show significant improvements in word error rates.

7.2 Future Work

In this thesis, we propose graph-based SSL method for acoustic modeling. To integrate the graph-based learning technique with a state-of-the-art DNN-based system, we introduce two different integration methods. The first method is a lattice-based integration framework which allows graph-based learning to be combined with the DNN system at the *output* level. The second method is a neural graph embedding approach which extracts a compact feature representation that encodes the graph neighborhood information, and this feature is combined with the original acoustic feature at the *input* level. In the future, we plan to investigate the following directions.

First, we plan to extend the lattice-based framework for graph-based SSL to under-resourced ASR tasks. Although the lattice-based framework entails expensive modeling procedures for large-scale acoustic modeling such as LVCSR system, the framework can be potentially very useful in under-resourced scenarios. In under-resourced scenarios, the amount of training data can vary from 30 minutes to a few hours of speech. The graph construction and inference can be generally finished in a reasonable amount of time. More importantly, the acoustic model trained using a limited amount of labeled data cannot be

easily generalized to unseen test data. The information from the manifold can be beneficial to improve the supervised classifier.

Second, we would like to investigate the performance of neural graph embedding approach in some targeted applications, where the conditions of the test data differ drastically from the training data. These applications include accented speech recognition, speech recognition under novel environment conditions.

Finally, we would like to study how to dynamically update the landmarks for neural graph embedding approach and how to update the feature extractor (autoencoder) in an online approach.

BIBLIOGRAPHY

[1] Ossama Abdel-Hamid and Hui Jiang. Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[2] Ossama Abdel-Hamid and Hui Jiang. Rapid and effective speaker adaptation of convolutional neural network based models for speech recognition. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2013.

[3] Andrei Alexandrescu and Katrin Kirchhoff. Graph-based learning for phonetic classification. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2007.

[4] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata*, pages 11–23. Springer, 2007.

[5] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

[6] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements, 2012.

[7] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, pages 164–171, 1970.

[8] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. Label propagation and quadratic criterion. pages 193–216. The MIT Press, 2006.

[9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[10] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. In *Proceedings of Neural Information Processing System (NIPS)*, 2007.

[11] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[12] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of International Conference on Machine Learning (ICML)*, 2006.

[13] Jeff Bilmes. Dynamic graphical models. *IEEE Signal Processing Magazine*, 27(6):2942, 2010.

[14] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[15] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[16] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2009.

[17] Jie Chen, Haw-ren Fang, and Yousef Saad. Fast approximate kNN graph construction for high dimensional data via recursive lanczos bisection. *The Journal of Machine Learning Research*, 10:1989–2012, 2009.

[18] Kevin Walker Christopher Cieri, David Miller. The Fisher corpus: a resource for the next-generation speech-to-text. In *Proceedings of LREC*, 2004.

[19] Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Lattice kernels for spoken-dialog classification. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2003.

[20] Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062, 2004.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[22] Fabio Gagliardi Cozman, Ira Cohen, Marcelo Cesar Cirelo, et al. Semi-supervised learning of mixture models. In *Proceedings of International Conference on Machine Learning (ICML)*, 2003.

[23] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[24] Samuel I Daitch, Jonathan A Kelner, and Daniel A Spielman. Fitting a graph to vector data. In *Proceedings of International Conference on Machine Learning (ICML)*, 2009.

[25] Najim Dehak, Patrick Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19(4):788–798, 2011.

[26] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586. ACM, 2011.

[27] Stéphane Dupont and Leila Cheboub. Fast speaker adaptation of artificial neural networks for automatic speech recognition. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2000.

[28] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[29] Andrew Errity and John McKenna. An investigation of manifold learning for speech analysis. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2006.

[30] Jonathan T Foote. Content-based retrieval of music and audio. In *Voice, Video, and Data Communications*, pages 138–147. International Society for Optics and Photonics, 1997.

[31] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[32] Satoru Fujishige. *Submodular Functions and Optimization*. Number 58 in Annals of Discrete Mathematics. Elsevier Science, 2nd edition, 2005.

[33] Mark JF Gales. Maximum likelihood linear transformations for hmm-based speech recognition. *Computer speech & language*, 12(2):75–98, 1998.

[34] Mark JF Gales. Semi-tied covariance matrices for hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 7(3):272–281, 1999.

[35] John S Garofolo, Lori F Lamel, William M Fisher, Jonathon G Fiscus, and David S Pallett. DARPA TIMIT acoustic-phonetic continous speech corpus. *NASA STI/Recon Technical Report N*, 93:27403, 1993.

[36] Roberto Gemello, Franco Mana, Stefano Scanzio, Pietro Laface, and Renato De Mori. Linear hidden transformations for adaptation of hybrid ANN/HMM models. *Speech Communication*, 49(10):827–835, 2007.

[37] John J Godfrey, Edward C Holliman, and Jane McDaniel. Switchboard: Telephone speech corpus for research and development. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1992.

[38] Vaibhava Goel and William J Byrne. Minimum bayes-risk automatic speech recognition. *Computer Speech & Language*, 14(2):115–135, 2000.

[39] Andrew B Goldberg and Xiaojin Zhu. Seeing stars when there aren't many stars: graph-based semi-supervised learning for sentiment categorization. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, pages 45–52. Association for Computational Linguistics, 2006.

[40] Ramesh A Gopinath. Maximum likelihood modeling with Gaussian distributions for classification. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1998.

[41] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Proceedings of Neural Information Processing System (NIPS)*, 2005.

[42] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[43] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2006.

[44] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2014.

[45] Frantiseli Grezl and Martin Karafiát. Semi-supervised bootstrapping approach for neural network feature extractor training. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2013.

[46] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[47] Joachim Hagenauer and Peter Hoeher. A viterbi algorithm with soft-decision outputs and its applications. In *Global Telecommunications Conference and Exhibition'Communications Technology for the 1990s and Beyond (GLOBECOM)*, 1989.

[48] Dilek Hakkani-Tur, Giuseppe Riccardi, and Allen Gorin. Active learning for automatic speech recognition. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2002.

[49] Andrew K Halberstadt. *Heterogeneous acoustic measurements and multiple classifiers for speech recognition*. PhD thesis, Massachusetts Institute of Technology, 1998.

[50] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[51] Luheng He, Jennifer Gillenwater, and Ben Taskar. Graph-based posterior regularization for semi-supervised structured prediction. In *Proceedings of Computational Natural Language Learning (CoNLL)*, 2013.

[52] Larry Heck and Hongzhao Huang. Deep learning of knowledge graph embeddings for semantic parsing of Twitter dialogs. In *Proceedings of GlobalSIP*, 2014.

[53] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

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

[55] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[56] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[57] Bert C Huang and Tony Jebara. Fast b-matching via sufficient selection belief propagation. In *International Conference on Artificial Intelligence and Statistics (AISTAT)*, 2011.

[58] Jui Ting Huang. *Semi-supervised learning for acoustic and prosodic modeling in speech applications*. PhD thesis, University of Illinois at Urbana-Champaign, 2012.

[59] Jui-Ting Huang and Mark Hasegawa-Johnson. Semi-supervised training of Gaussian mixture models by conditional entropy minimization. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2010.

[60] Yan Huang, Dong Yu, Yifan Gong, and Chaojun Liu. Semi-supervised GMM and DNN acoustic model training with multi-system combination and confidence re-calibration. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2013.

[61] Rishabh Iyer and Jeff Bilmes. Algorithms for approximate minimization of the difference between submodular functions, with applications. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, 2012.

[62] Rishabh Iyer and Jeff Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *Proceedings of Neural Information Processing System (NIPS)*, 2013.

[63] Tommi S Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Proceedings of Neural Information Processing System (NIPS)*, 1999.

[64] Aren Jansen and Partha Niyogi. A geometric perspective on speech sounds. Technical report, University of Chicago, 2005.

[65] Aren Jansen and Partha Niyogi. Intrinsic fourier analysis on the manifold of speech sounds. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2006.

[66] Tony Jebara, Jun Wang, and Shih-Fu Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of International Conference on Machine Learning (ICML)*, 2009.

[67] Frederick Jelinek. *Statistical methods for speech recognition*. MIT press, 1997.

[68] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of International Conference on Machine Learning (ICML)*, 1999.

[69] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[70] Biing-Hwang Juang, Wu Hou, and Chin-Hui Lee. Minimum classification error rate methods for speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 5(3):257–265, 1997.

[71] Kelley Kilanski, Jon Malkin, Xiao Li, Richard Wright, and Jeff Bilmes. The Vocal Joystick data collection effort and vowel corpus. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2006.

[72] Simon King, Chris Bartels, and Jeff Bilmes. SVitchboard 1: Small Vocabulary Tasks from Switchboard. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2005.

[73] Katrin Kirchhoff and Andrei Alexandrescu. Phonetic classification using controlled random walks. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2011.

[74] Katrin Kirchhoff, Jeff Bilmes, Kai Wei, Yuzong Liu, Arindam Mandal, and Chris Bartels. A submodularity framework for data subset selection. Technical report, DTIC Document, 2013.

[75] Katrin Kirchhoff, Yuzong Liu, and Jeff Bilmes. Classification of developmental disorders from speech signals using submodular feature selection. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2013.

[76] Lori Lamel, Jean-Luc Gauvain, and Gilles Adda. Investigating lightly supervised acoustic model training. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2001.

[77] Lori Lamel, Jean-Luc Gauvain, and Gilles Adda. Unsupervised acoustic model training. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2002.

[78] Kai-Fu Lee. Context-dependent phonetic hidden Markov models for speaker-independent continuous speech recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 38(4):599–609, 1990.

[79] Christopher J Leggetter and Philip C Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech & Language*, 9(2):171–185, 1995.

[80] Bo Li and Khe Chai Sim. Comparison of discriminative input and output transformations for speaker adaptation in the hybrid NN/HMM systems. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2010.

[81] Xiao Li and Jeff Bilmes. Regularized adaptation of discriminative classifiers. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2006.

[82] Hank Liao, Erik McDermott, and Andrew Senior. Large scale deep neural network acoustic modeling with semi-supervised training data for YouTube video transcription. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2013.

[83] Hui Lin and Jeff Bilmes. An application of the submodular principal partition to training data subset selection. In *NIPS Workshop on Discrete Optimization in Machine Learning: Submodularity, Sparsity & Polyhedra*, 2010.

[84] Hui Lin and Jeff A. Bilmes. Optimal selection of limited vocabulary speech corpora. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2011.

[85] Ting Liu, Andrew W Moore, Ke Yang, and Alexander G Gray. An investigation of practical approximate nearest neighbor algorithms. In *Proceedings of Neural Information Processing System (NIPS)*, 2004.

[86] Wei Liu, Junfeng He, and Shih-Fu Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of International Conference on Machine Learning (ICML)*, 2010.

[87] Yuzong Liu, Rishabh Iyer, Katrin Kirchhoff, and Jeff Bilmes. SVitchboard II and FiSVer I: High-quality limited-complexity corpora of conversational English speech. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2015.

[88] Yuzong Liu and Katrin Kirchhoff. A comparison of graph construction and learning algorithms for graph-based phonetic classification. Technical Report UWEETR-2012-0005, University of Washington, Seattle, 2012.

[89] Yuzong Liu and Katrin Kirchhoff. Graph-based semi-supervised learning for phone and segment classification. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2013.

[90] Yuzong Liu and Katrin Kirchhoff. Graph-based semi-supervised acoustic modeling in dnn based speech recognition. In *Proceedings of IEEE Spoken Language Technology Workshop (SLT)*, 2014.

[91] Yuzong Liu and Katrin Kirchhoff. Acoustic modeling with neural graph embeddings. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2015.

[92] Yuzong Liu, Mohit Sharma, Charles Gaona, Jonathan Breshears, Jarod Roland, Zachary Freudenburg, Eric Leuthardt, and Kilian Q Weinberger. Decoding ipsilateral finger movements from ecog signals in humans. In *Proceedings of Neural Information Processing System (NIPS)*, 2010.

[93] Yuzong Liu, Kai Wei, Katrin Kirchhoff, Yisong Song, and Jeff Bilmes. Submodular feature selection for high-dimensional acoustic score space. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[94] Jonathan Malkin, Amar Subramanya, and Jeff A. Bilmes. On the semi-supervised learning of multi-layered perceptrons. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2009.

[95] Florian Metze, Zaid Sheikh, Alex Waibel, Jonas Gehring, Kevin Kilgour, Quoc Bao Nguyen, Van Huy Nguyen, et al. Models of tone for tonal and non-tonal languages. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2013.

[96] Yajie Miao, Mohammad Gowayyed, and Florian Metze. Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2015.

[97] Yajie Miao, Mohammad Gowayyed, Xingyu Na, Tom Ko, Florian Metze, and Alexander Waibel. An empirical exploration of CTC acoustic models. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2016.

[98] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2010.

[99] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243. Springer, 1978.

[100] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[101] Pedro J Moreno and Ryan Rifkin. Using the fisher kernel method for web audio classification. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2000.

[102] Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

[103] Joao Neto, Luís Almeida, Mike Hochberg, Ciro Martins, Luis Nunes, Steve Renals, and Tony Robinson. Speaker-adaptation for hybrid HMM-ANN continuous speech recognition system. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 1995.

[104] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine learning*, 39(2-3):103–134, 2000.

[105] Scott Novotney, Richard Schwartz, and Jeff Ma. Unsupervised acoustic and language model training with small amounts of labelled data. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2009.

[106] Matan Orbach and Koby Crammer. Transductive phoneme classification using local scaling and confidence. In *Proceedings of IEEE 27th Convention of Electric and Electronics Engineers in Israel*, 2012.

[107] Kohei Ozaki, Masashi Shimbo, Mamoru Komachi, and Yuji Matsumoto. Using the mutual k-nearest neighbor graphs for semi-supervised classification of natural language data. In *Proceedings of Computational Natural Language Learning (CoNLL)*, 2011.

[108] Hanchuan Peng, Fulmi Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[109] Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for image categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[110] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier. Large-scale image retrieval with compressed Fisher vectors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[111] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, et al. The Kaldi speech recognition toolkit. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2011.

[112] Daniel Povey, Dimitri Kanevsky, Brian Kingsbury, Bhuvana Ramabhadran, George Saon, and Karthik Visweswariah. Boosted mmi for model and feature-space discriminative training. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

[113] Daniel Povey and Philip C Woodland. Minimum phone error and i-smoothing for improved discriminative training. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2002.

[114] Patti Price, William M Fisher, Jared Bernstein, and David S Pallett. The DARPA 1000-word resource management database for continuous speech recognition. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1988.

[115] Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[116] Juho Rousu and John Shawe-Taylor. Efficient computation of gapped substring kernels on large alphabets. pages 1323–1344, 2005.

[117] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[118] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[119] Hasim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2014.

[120] Hasim Sak, Oriol Vinyals, Georg Heigold, Andrew Senior, Erik McDermott, Rajat Monga, and Mark Mao. Sequence discriminative distributed training of long short-term memory recurrent neural networks. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2014.

[121] George Saon, Hong-Kwang J Kuo, Steven Rennie, and Michael Picheny. The IBM 2015 English conversational telephone speech recognition system. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2015.

[122] George Saon, Hagen Soltau, David Nahamoo, and Michael Picheny. Speaker adaptation of neural network acoustic models using i-vectors. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2013.

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

[124] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2011.

[125] Izhak Shafran, Michael Riley, and Mehryar Mohri. Voice signatures. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2003.

[126] Vin De Silva and Joshua B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Proceedings of Neural Information Processing System (NIPS)*, 2002.

[127] Andreas Stolcke et al. SRILM-an extensible language modeling toolkit. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2002.

[128] A. Subramanya and J. Bilmes. Semi-supervised learning with measure propagation. Technical Report UWEE-TR-2010-0004, University of Washington, 2010.

[129] Amar Subramanya and Jeff Bilmes. Soft-supervised learning for text classification. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[130] Amarnag Subramanya and Jeff Bilmes. Entropic graph regularization in non-parametric semi-supervised classification. In *Proceedings of Neural Information Processing System (NIPS)*, 2009.

[131] Amarnag Subramanya and Jeff Bilmes. The semi-supervised Switchboard transcription project. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2009.

[132] Partha Pratim Talukdar and Koby Crammer. New regularized algorithms for transductive learning. In *Proceedings of ECML-PKDD*, 2009.

[133] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[134] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[135] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[136] Karel Veselỳ, Mirko Hannemann, and Lukas Burget. Semi-supervised training of deep neural networks. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2013.

[137] Vincent Wan and Steve Renals. Speaker verification using sequence discriminant support vector machines. *IEEE Transactions on Speech and Audio Processing*, 13(2):203–210, 2005.

[138] Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2008.

[139] Wei Wang, Yan Huang, Yizhou Wang, and Liang Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2014.

[140] Kai Wei, Yuzong Liu, Katrin Kirchhoff, Chris Bartels, and Jeff Bilmes. Submodular subset selection for large-scale speech training data. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014.

[141] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *Proceedings of NAACL-HLT*, 2013.

[142] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Unsupervised submodular subset selection for speech data. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014.

[143] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Proceedings of Neural Information Processing System (NIPS)*, 2005.

[144] Kilian Q Weinberger, Fei Sha, and Lawrence K Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2004.

[145] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[146] Frank Wessel and Hermann Ney. Unsupervised training of acoustic models for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 13(1):23–31, 2005.

[147] Eric P Xing, Michael I Jordan, Stuart Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *Proceedings of Neural Information Processing System (NIPS)*, 2002.

[148] Zhixiang Xu, Gao Huang, Kilian Q Weinberger, and Alice X Zheng. Gradient boosted feature selection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 522–531. ACM, 2014.

[149] Shaofei Xue, Ossama Abdel-Hamid, Hui Jiang, and Lirong Dai. Direct adaptation of hybrid DNN/HMM model for fast speaker adaptation in LVCSR based on speaker code. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014.

[150] Kaisheng Yao, Dong Yu, Frank Seide, Hang Su, Li Deng, and Yifan Gong. Adaptation of context-dependent deep neural networks for automatic speech recognition. In *Proceedings of IEEE Spoken Language Technology Workshop (SLT)*, 2012.

[151] Dong Yu, Balakrishnan Varadarajan, Li Deng, and Alex Acero. Active learning and semi-supervised learning for speech recognition: A unified framework using the global entropy reduction maximization criterion. *Computer Speech & Language*, 24(3):433–444, 2010.

[152] Dong Yu, Kaisheng Yao, Hang Su, Gang Li, and Frank Seide. KL-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[153] Kai Yu, Mark Gales, Lan Wang, and Philip C Woodland. Unsupervised training and directed manual transcription for lvcsr. *Speech Communication*, 52(7):652–663, 2010.

[154] Xiaohui Zhang, Jan Trmal, Dan Povey, and Sanjeev Khudanpur. Improving deep neural network acoustic models using generalized maxout networks. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014.

[155] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. Fast kNN graph construction with locality sensitive hashing. In *Machine Learning and Knowledge Discovery in Databases*, pages 660–674. Springer, 2013.

[156] Rui Zhao, Jinyu Li, and Yifan Gong. Variable-activation and variable-input deep neural network for robust speech recognition. In *Proceedings of IEEE Spoken Language Technology Workshop (SLT)*, 2014.

[157] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Proceedings of Neural Information Processing System (NIPS)*, 2004.

[158] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, CMU-CALD-02, 2002.

[159] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[160] Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of International Conference on Machine Learning (ICML)*, 2003.

Appendix A

SVITCHBOARD-II & FISVER-I: HIGH-QUALITY LOW-COMPLEXITY CONVERSATIONAL ENGLISH SPEECH CORPORA

In this chapter, we describe the datasets used in this thesis. We first explain the motivations of creating a new benchmark dataset to evaluate the performance of acoustic models for LVCSR systems in Section A.1. We then describe the overall objective and general submodular optimization framework in Section A.2. In Section A.3, we describe the algorithms and function instantiations used in creating the new benchmark datasets. In Section A.4, we show the detailed information on the final selected corpora.

A.1 Motivation

Speech recognition is one of the most challenging tasks in applied machine learning, and one that requires enormous amounts of rich training data. Among different aspects of a speech recognition system, training acoustic models for LVCSR system is the most challenging tasks.

To train good acoustic models for LVCSR system, we need to use datasets that are diverse in acoustic characteristics. This is due to the fact that the acoustic characteristics (variability in pronunciation, speaker, and environment) of conversational speech are more diverse than those of carefully read speech. Also, conversational speech recognition involves large vocabularies. Finally, novel acoustic models, especially the deep models, require long training and experimental turnaround time. While most research groups in industry [53, 23, 118, 42, 119, 120, 121] have the computational resource and large amount of training data, it is much more difficult for researchers with limited computational resources in academia. The complexity of acoustic model training is usually linear in n (i.e., $O(n)$), where n is the

number of tokens in the training data. For very large n and computationally demanding models like DNNs, it can take weeks to train just one system even on GPUs. Such long experimental turnaround time makes large-scale speech recognition impractical, particularly in academia where most researchers and students have limited computational resources.

To address this issue, we craft a new set of benchmark datasets with high quality but limited complexity corpora for LVCSR system. We create these new corpora from two of the most commonly used conversational speech corpora: Switchboard-I [37] and Fisher [18] datasets, both of which are large in terms of vocabulary size and number of training samples. Our goal is 1) to produce useful but acoustically rich and challenging subsets of Switchboard and Fisher, 2) to establish baselines performance numbers, and 3) to release the corpora definitions for free to the community. We refer to the resulting corpora as SVitchboard-II (SVB-II), and FiSVer-I, where in each case "SV" stands for "small vocabulary." By doing so, we hope to provide researchers with smaller but still challenging speech corpora, thus facilitating faster experimental throughput for testing novel acoustic modeling and machine learning methods.

A.2 Subset Selection Objective

The basic goal of **high-quality limited-complexity corpus selection** is to choose a large subset X of a *ground set* V of speech utterances (e.g., the entire 309-hour Switchboard-I dataset) that has limited complexity but is similar to the original dataset in some way. That is, we wish to choose a subset $X \subseteq V$ that have the following two properties:

1. **high quality:** That is, X being high quality may mean that the utterances X constitute a large amount of speech, a large number of tokens, or be acoustically diverse and/or confusable in some way. We construct a function $g(X)$ that measures the quality of X, and we choose X such that $g(X)$ is maximized.

2. **low complexity:** Complexity may correspond to computational cost of running an ASR system, so an obvious complexity measure is the vocabulary size in X (i.e., the

number of distinct types in X). We define a function $f(X)$ that measures the complexity of X, and choose X such that $f(X)$ is minimized.

Selecting such subset is a difficult combinatorial optimization task as it involves exploring an exponential amount of possible subsets ($2^{|V|}$), where V is the size of the ground set. In previous work [72], a heuristic method was proposed to select different subsets of Switchboard (with vocabulary size of 10, 25, 50, 100, 250, and 500 words). In general, the heuristic approach in [72] greedily selected the most frequent words in the transcripts until the vocabulary size constraint were met, a procedure that can have unboundedly poor performance [84]. In this work, we investigate a principled approach based on [83, 84, 62, 61] to data selection using submodular function [32] optimization.

A.3 Subset Selection using Submodular Function Optimization

A.3.1 Algorithm

In this section, we apply three submodular function optimization algorithms for the subset selection task. The first algorithm we evaluate is proposed in [62] and referred to as "Submodular Cost Submodular Cover (SCSC)", and "Submodular Cost Submodular Knapsack" (SCSK), respectively:

$$\text{Problem 1 (SCSC): } \min\{f(X) \,|\, g(X) \geq c\}, \text{and} \tag{A.1}$$

$$\text{Problem 2 (SCSK): } \max\{g(X) \,|\, f(X) \leq b\}, \tag{A.2}$$

where both $g : 2^V \rightarrow \mathbb{R}_+$ and $f : 2^V \rightarrow \mathbb{R}_+$ are polymatroid (non-negative monotone-nondecreasing submodular) functions.

This addresses exactly the problem we wish to solve. In particular, we can use the formulation of Problem 2 and directly enforce constraints on the vocabulary size while maximizing the quality. Unlike submodular function minimization, however, this problem is NP-hard [62]. Several of the algorithms proposed in [62], however, are scalable and admit bounded approximation guarantees.

The second algorithm is the difference of submodular functions optimization, or "DS" in short [61]. DS algorithm minimizes the difference between submodular functions, defined as follows:

$$\text{Problem 3 (DS): } \min_{X \subseteq V} v(X) \tag{A.3}$$

where $v(X) = \lambda f(X) - g(X)$ is a difference of two submodular functions. Similar to SCSC/SCSK, this method addresses the underlying problem; different values of λ will amount to different vocabulary sizes. Unfortunately, unlike SCSC and SCSK, we do not have explicit control over the vocabulary size and we instead need to tune λ to obtain the right solution. Like SCSC/SCSK this problem is NP-hard, but the algorithms proposed in [61] are scalable and work well in practice.

Finally, we evaluate the submodular function minimization algorithms, or "SFM" in short. In SFM, we minimize the following objective:

$$\text{Problem 4 (SFM): } \min_{X \subseteq V} h(X) \tag{A.4}$$

where $h(X) = g(V \setminus X) + \lambda f(X)$ is a submodular function. We minimize $g(V \setminus X)$, the quality of X's complement $V \setminus X$, rather than maximizing the quality of X. With these new algorithms, therefore, we can directly address the problem of high quality limited complexity corpus selection. In the following section, we describe different function instantiations for the quality function $g(\cdot)$ and complexity function $f(\cdot)$.

A.3.2 Function Instantiations

We start with four different modular functions as quality functions g. All are normalized so that $g(\emptyset) = 0$ and $g(V) = 1$.

- **Utterance count**: $g_1(X) = |X|/|V|$. This defines high quality as containing a large percentage of utterances in V. Each utterance (short or long) is given equal weight.

- **Amount of speech**: $g_2(X) = w_V(X)/w_V(V)$ where $w_V(X) = \sum_{v \in X} w_V(v)$ and $w_V(v)$ measures how much speech (excluding silence) is in utterance v.

- **Number of tokens**: $g_3(X) = w_V(X)/w_V(V)$ where $w_V(X) = \sum_{v \in X} w_V(v)$ and $w_V(v)$ measures how many tokens are contained in the transcription of utterance v.

- **Intra-utterance acoustic dispersion/diversity**. $g_4(X) = w_V(X)/w_V(V)$ where $w_V(X) = \sum_{v \in X} w_V(v)$ and $w_V(v)$ measures the "acoustic dispersion" of utterance v. If $x^v = (x_1^v, x_2^v, \ldots, x_T^v)$ is a sequence of MFCC vectors for utterance v, then we can measure acoustic dispersion via:

$$w_V(v) = \frac{1}{T^2} \left| \sum_{i=1}^{T} \sum_{j=1}^{T} (x_i^v - x_j^v)(x_i^v - x_j^v)^\intercal \right| \tag{A.5}$$

  Hence, we prefer an utterance if it is internally acoustically diverse.

In addition to the above modular quality functions, we also consider a submodular quality function, or the so-called "feature-based function" defined in [140]. The feature-based function has the following formulation:

$$g_5(X) = \sum_{u \in \mathcal{U}} w_u \phi(m_u(X)) \tag{A.6}$$

where $\phi(\cdot)$ is a non-negative monotone non-decreasing concave function, $\mathcal{U}$ is a set of features, and $m_u(S) = \sum_{j \in S} m_u(j)$ is a non-negative score for feature u in set S, with $m_u(j)$ measuring the degree to which utterance $j \in S$ possesses feature u. w_u is a non-negative weight for feature u. We note that Equation (A.6) is a sum of concave functions over modular functions, and is hence submodular.

Each term in the sum is based on a "feature" of the objects being scored. The feature based submodular functions are convenient for applications in speech processing since speech objects can often be described by a variety of phonetic or prosodic feature labels (e.g. phonemes, triphones, words, syllables, tones, etc.). Feature-based submodular functions, therefore, have the ability to leverage much of the important work on both knowledge- and data-driven feature engineering that has been available in speech processing.

In [140], $\mathcal{U}$ is the set of triphones over frame labels that are derived from the word transcriptions via forced alignments using a trained system. The function $\phi()$ is the square

root function. The score $m_u(s)$ is the count of feature u in element s, normalized by term frequency-inverse document frequency (TF-IDF), i.e., $m_u(s) = \text{TF}_u(s) \times \text{IDF}_u(s)$, where $\text{TF}_u(s)$ is the count of feature u in s, and $\text{IDF}_u = \log(\frac{|V|}{d(u)})$ is the inverse document count of the feature u with $d(u)$ being the number of utterances that contain the feature u (each utterance is considered a "document"). The choice of w_u affects the attribute of the subset. When $w_u = 1$ for all $u \in \mathcal{U}$, maximizing this objective naturally encourages diverse coverage of all features within X. When w_u is set to the prior distribution of feature u in the ground set V, maximizing this objective would result in a subset X that is acoustically representative of V. In this work, we choose the latter option for the feature weights.

A.4 Corpora Selection

In the previous section we proposed three different algorithms with different $f(\cdot)$ and $g(\cdot)$ instantiations. Our goal here is to create subsets using different submodular optimization algorithms as well as function instantiations. For Switchboard I and Fisher we first remove utterances containing the disfluencies and fillers. For example, we remove utterances that contain only word fragments (e.g. sim[ilar]-), *uh*, [noise], *yeah*, [laughter], *huh*, *hm*, [laughter-*], *uh-huh*, *um-hum hum*, *huh-uh*, *um*. The size of the resulting ground sets V for Switchboard I and Fisher is 93312, and 1.7 million, respectively. For Switchboard I, we use a combination of different algorithms and function instantiations for a given target vocabulary size. For Fisher, we use the SCSK algorithm with g_5 as the quality function because of scalability and the computational efficiency of the SCSK algorithm.

To choose the best resulting corpus for a particular target vocabulary size, we run each of the optimization methods which gives us a relatively small number of corpora to choose from. We then compute a set of statistics on each of the resulting corpora such as the actual vocabulary size, average number of phonemes per word, the number of utterances and tokens, the speech durations, etc. We also compute the value g_5 for each subset; note that the g_5 function measures the representativeness of the subsets. We believe this value is a good indicator of corpus diversity. To show a corpus's phonetic balance, we compute

the normalized entropy of the phoneme distribution $\frac{H(p)}{\log(43)}$ and the normalized entropy of the non-silence phoneme distribution $\frac{H(p)}{\log(42)}$, where we use 43 phones in the lexicon with 42 non-silence phones. $H(p)$ is the entropy of the probability distribution over phonemes in the selected subset.

In order to have the best final corpus for the current vocabulary size, we make the final selection by visual inspection of these statistics (i.e., by hand). Table A.1 and Table A.2 shows the statistics of our chosen corpora, comprising both SVitchboard-II and FiSVer-I. Table A.3 shows the specific algorithm and function instantiations we used to create each subset in Svitchboard-II.

SVitchboard-II Dataset									
Task	Vocab Size	Avg. Phone	# Utts	# Tokens	Speech (hrs)	g-value	# conv.	norm. ent 1	norm. ent 2
50	50	3.32	24033	38154	4.01	4.13054e9	4491	0.4688	0.3916
100	100	3.28	27228	51254	4.93	4.8425e9	4571	0.4998	0.4203
500	500	3.95	39694	131815	10.30	7.70767e9	4749	0.6122	0.5243
1000	1001	4.50	48445	230876	16.81	9.70981e9	4801	0.6831	0.5911
5000	5003	5.55	74162	668261	46.28	1.49496e10	4867	0.78340	0.6911
10000	9983	5.97	84636	883710	61.19	1.68402e10	4871	0.8059	0.7152
All	30021	6.22	262473	3109768	224.11 (310 total)	1.0244404e11	4876	0.8016	0.7108

Table A.1: Statistics of SVitchboard-II datasets. Vocab size: actual vocabulary size; Avg. Phone: average number of phonemes per word; #Utts: number of utterances; #Tokens: number of tokens; Speech: hours of speech (excluding the silence parts); g-value: the function value of $g_5(X)$; # conv.: number of conversation sides; norm. ent 1: normalized entropy of phoneme distribution; norm. ent 2: normalized entropy of non-silence phoneme distribution

A.5 Downloads

The corpora can be downloaded from: `https://bitbucket.org/melodi/hqlc-speechcorpora`.

FiSVer-I Dataset									
Task	Vocab Size	Avg. Phone	# Utts	# Tokens	Speech (hrs)	g-value	# conv.	norm. ent 1	norm. ent 2
10	10	4.6	64998	73650	9.96	3.21993e10	15561	0.4740	0.3815
50	50	5.92	115512	144906	17.95	6.91023e10	21052	0.5609	0.4678
100	100	5.6	138722	191156	22.01	9.56028e10	22062	0.5891	0.4958
500	500	5.34	258307	653847	55.32	2.25651e11	23111	0.7129	0.6175
1000	1000	5.43	352261	1299566	99.06	3.23534e11	23214	0.7744	0.5911
All	42154	6.36	1.7M	17M	1242.5 (1593 total)	1.30739e12	23300	0.8596	0.7737

Table A.2: Statistics of FiSVer-I datasets. Vocab size: actual vocabulary size; Avg. Phone: average number of phonemes per word; #Utts: number of utterances; #Tokens: number of tokens; Speech: hours of speech (excluding the silence parts); g-value: the function value of $g_5(X)$; # conv.: number of conversation sides; norm. ent 1: normalized entropy of phoneme distribution; norm. ent 2: normalized entropy of non-silence phoneme distribution

Task	Algorithm and Function
50	DS, g_5
100	SFM, g_2
500	DS, g_5
1000	DS, g_5
5000	SFM, g_2
10000	SFM, g_2

Table A.3: Selected datasets for Svitchboard-II and the corresponding algorithms and functions.

VITA

Yuzong Liu

Degrees

B.Eng. Electrical and Information Engineering, Southeast University, Nanjing, China, June 2009

M.S. Computer Science, Washington University, St. Louis, MO, May 2011

Ph.D. Electrical Engineering, University of Washington, Seattle, WA, June 2016

Publications

Yuzong Liu, Katrin Kirchhoff. Acoustic Modeling with Neural Graph Embeddings. ASRU 2015.

Yuzong Liu, Rishabh Iyer, Katrin Kirchhoff, Jeff Bilmes. SVitchboard II and FiSVer I: High-Quality Limited-Complexity Corpora of Conversational English Speech. Interspeech 2015.

Yuzong Liu, Katrin Kirchhoff. Graph-based Semi-supervised Learning in DNN-based Speech Recognition. SLT 2014.

Kai Wei, Yuzong Liu, Katrin Kirchhoff, Chris Bartels, Jeff Bilmes. Submodular subset selection for large-scale speech training data. ICASSP 2014.

Kai Wei, Yuzong Liu, Katrin Kirchhoff, Jeff Bilmes. Unsupervised submodular subset selection for speech data. ICASSP 2014.

Yuzong Liu, Katrin Kirchhoff. Graph-based Semi-supervised Learning for Phone and Segment Classification. Interspeech 2013.

Katrin Kirchhoff , Yuzong Liu, Jeff Bilmes. Classification of Developmental Disorders from Speech Signals Using Submodular Feature Selection. Interspeech 2013.

Kai Wei*, Yuzong Liu*, Katrin Kirchhoff, Jeff Bilmes. Using Document Summarization Techniques for Speech Data Subset Selection. NAACL 2013. (* Joint first authors)

Yuzong Liu, Kai Wei, Katrin Kirchhoff, Yisong Song, Jeff Bilmes. Submodular Feature Selection for High-Dimensional Acoustic Score Space. ICASSP 2013

Yuzong Liu, Katrin Kirchhoff. A comparison of graph construction and learning algorithms for graph-based phonetic classification. UWEE Technical Report 2012

Yuzong Liu, Mohit Sharma, Charles M. Gaona, Jonathan D. Breshears, Jarod Roland, Zachary V. Freudenburg, Kilian Q. Weinberger, and Eric C. Leuthardt. (2010). Decoding Ipsilateral Finger Movements from ECoG Signals in Humans, NIPS 2010.