SEQUENCE PREDICTION WITH NEURAL SEGMENTAL MODELS

BY

HAO TANG

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

Doctor of Philosophy in Computer Science

at the

TOYOTA TECHNOLOGICAL INSTITUTE AT CHICAGO
Chicago, Illinois

September, 2017

Thesis Committee:
Karen Livescu (Thesis Advisor)
Kevin Gimpel
David McAllester
Eric Fosler-Lussier
James Glass

SEQUENCE PREDICTION
WITH NEURAL SEGMENTAL MODELS

A thesis presented

by

HAO TANG

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

Toyota Technological Institute at Chicago

Chicago, Illinois

September, 2017

— Thesis Committee —

Kevin Gimpel	Kevin Gimpel	9-9-17
Committee member (print/type)	Signature	Date
David McAllester	David McAllester	9/7/17
Committee member (print/type)	Signature	Date
Eric Fosler-Lussier	J Eric Fosler-Lussier	9/11/17
Committee member (print/type)	Signature	Date
James Glass	James Glass	9/7/17
Committee member (print/type)	Signature	Date
Karen Livescu	Karen Livescu	9/6/17
Thesis/Research Advisor (print/type)	Signature	Date
Avrim Blum	Avrim Blum	9/11/17
Chief Academic Officer (print/type)	Signature	Date

Sequence Prediction with Neural Segmental Models

by
Hao Tang

Abstract

Segments that span contiguous parts of inputs, such as phonemes in speech, named entities in sentences, actions in videos, occur frequently in sequence prediction problems. Segmental models, a class of models that explicitly hypothesizes segments, have allowed the exploration of rich segment features for sequence prediction. However, segmental models suffer from slow decoding, hampering the use of computationally expensive features. In this thesis, we introduce discriminative segmental cascades, a multi-pass inference framework that allows us to improve accuracy by adding higher-order features and neural segmental features while maintaining efficiency.

Segmental models, similarly to conventional speech recognizers, are typically trained in multiple stages. In the first stage, a frame classifier is trained, and in the second stage, segmental models are trained with the outputs of the frame classifier. Both training stages require manual alignments, and obtaining manual alignments are time-consuming and expensive. We explore end-to-end training for segmental models with various loss functions, and show how end-to-end training with marginal log loss can eliminate the need for detailed manual alignments.

We draw the connections between the marginal log loss and a popular end-to-end training approach called connectionist temporal classification, and present a unifying framework for various end-to-end graph search-based models, such as hidden Markov models, connectionist temporal classification, and segmental models. Finally, we discuss possible extensions of segmental models to large-vocabulary sequence prediction tasks.

Thesis Supervisor: Karen Livescu
Title: Associate Professor

Acknowledgements

First, I would like to thank my advisor, Karen Livescu. Karen has always been very patient, letting me make mistakes after mistakes, mistakes that she can foresee, mistakes that I have made many times already, mistakes that have no easy fix, so that I can learn from them. She taught me when to explore novel ideas and when to persist and complete tedious tasks. I have learned to think critically and argue constructively, while at the same time to be honest and critical to my own work. There are countless other things I have learned from her including phonetics, spectrogram reading, and English word usage, to name a few. I can not express how much I appreciate her guidance and mentorship. I am really fortunate to be her first PhD student.

Next, I want to thank my committee members, Kevin Gimpel, David McAllester, Eric Fosler-Lussier, and Jim Glass. David's comments and questions always encourage me to view my work in a broader context and in the most general form possible. I have also greatly benefited from his mathematical rigor through his talks, his lectures, and interactions with him, and have developed the habit of asking whether a mathematical expression is type checked or not. I am grateful to Eric for his insightful comments. I have benefited a lot from conversations we had in workshops and conferences. I enjoyed his humorous and joyful attitude, and him pushing me towards the finish line harder than my advisor does. I thank Jim for keeping the speech science aspect of my research in mind. My research is heavily influenced by his prior work, and it is very special to work on something that he has been working on for decades.

I am deeply grateful to Kevin. He is like my second advisor, caring and helpful in many ways. I have benefited from his expertise in natural language processing (NLP), and discovered many similar approaches developed in parallel in both the NLP and the speech community. Besides the technical content, he taught me to be humble yet to be bold when necessary. I have also enjoyed his humorous comments on my writing before stressful deadlines. It has been a great pleasure to have his company during my PhD.

I was fortunate to join Mark Hasegawa-Johnson's team in the second Jelinek summer workshop. I have benefited from Mark's extensive knowledge in both speech science and machine learning. It is fair to say that I always learn something new when talking to him. I also thank him for his guidance after the workshop. It has been a great pleasure to work with the people on the team. In particular, I have benefited from interactions with Chunxi Liu, Preethi Jyothi, Amit Das, Vimal Manohar, Paul Hager, Tyler Kekona, and Rose Sloan. People I met during the workshop, including Luan Yi, Yangfeng Ji, Lingpeng Kong, Guoguo Chen, and Trang Tran, also made the overall experience fun and pleasant.

In 2013, I worked as an intern with Shinji Watanabe at the Mitsubishi Electric Research Laboratories (MERL). I have benefited from the interactions with him and other people at MERL, including John Hershey, Jonathan Le Roux, and Tim Marks. In particular, I thank Shinji for taking care of me when I had Bell's palsy during the internship. The fellow interns at MERL, including Niao He and Lingling Tao, also made the internship experience fun and rewarding.

My first research experience related to speech and language processing was acquired from

Lin-Shan Lee's guidance in National Taiwan University back in 2009. Lin-Shan, even with his always busy schedule, has been very caring and helpful. It would not have been possible for me to pursue a PhD without his help and encouragement.

I thank the people in Toyota Technological Institute at Chicago (TTIC) for making my PhD journey wonderful. In particular, I have benefited from Julia Chuzhoy's and Madhur Tulsiani's lectures, acquiring the necessary vocabulary to talk to theory people. I thank Nati Srebro for grilling me during my qualifying exam, forcing me to always keep the precise language and mathematical terms in mind. I enjoyed the interactions with Greg Shakhnarovich and his humor. I thank Joseph Keshet for his guidance and mentorship during my first few years of PhD. I have benefited interactions and collaboration with Liang Lu. I thank the fellow students, including Behnam Tavakoli, Somaye Hashemifar, Taehwan Kim, Shubham Toshniwal, Shane Settle, Qingming Tang, Lifu Tu, Hai Wang, Jian Yao, Jianzhu Ma, Xing Xu, Avleen Bijral, Payman Yadollahpour, Feng Zhao, Zhiyong Wang, Karthik Sridharan, and Peng Jian, for the interactions, cookie breaks, random interruptions, and random questions. I will never regret choosing a cubicle that invites so many interruptions. I am grateful to Weiran Wang. It is fair to say that much of my work would not be possible without his help. I have also greatly benefited from interactions with past post-docs, including Raman Arora and Herman Kamper. I thank visiting students, Taiki Kawano and Takayuki Yamabe, from Toyota Technological Institute in Japan, for the fun time we had in Chicago and in Tokyo. Thanks to Chrissy Novak, Adam Bohlander, and the rest of the staff for making my PhD life easy and smooth. Lastly, I thank the president of TTIC, Sadaoki Furui, for his guidance, and for pushing me to exercise and to practice my tennis skills with him.

Thanks to my family in Taiwan for their support over the years. Thanks to Hsin-Yu, for everything. And thanks to my parents, whom I cannot possibly thank in words.

Contents

Chapter 1

Introduction

Segmental models, the subject of this thesis, are a family of models that predict sequences of segments. Segmental models are designed for a wide range of sequence prediction tasks, such as named entity recognition (Sarawagi and Cohen, 2005), speech recognition (Ostendorf et al., 1996; Glass, 2003), and action recognition (Simon et al., 2010; Hoai et al., 2011). Examples of these tasks are shown in Figure 1.1. The input of a sequence prediction task is commonly represented as a sequence of real-valued vectors, and the output is a sequence of discrete labels. The goal is to find a function that maps the input vectors to the output labels. Every output label in the output sequence, such as a named entity, a phoneme, or an action, typically has a corresponding chunk of contiguous input vectors, called a segment. Segments come in varying lengths. As a consequence, the lengths of the output sequences typically do not match the lengths of the input sequences.

The predominant approach to solving these tasks is to break the varying-length segments into pieces so that the lengths of the input sequences match the lengths of the output sequences. These smaller pieces are then assembled back into varying-length segments with an additional post-processing step. Each individual element in the input sequence is referred to as a **frame**. Models, such as hidden Markov models (Rabiner, 1989; Jelinek, 1998), linear-chain conditional random fields (Gunawardana et al., 2005; Morris and Fosler-Lussier, 2009), and recurrent neural networks (Graves and Schmidhuber, 2005), that map input sequences to output sequences of the same lengths, are called **frame-based models** (Ostendorf et al., 1996; Glass, 2003). To include a wider context, it is common to use windows of frames instead of individual frames for prediction. Nevertheless, the number of windowed frames is still the same as the number of output labels. Because frame-based models are conceptually simple and computationally cheap, they have enjoyed much success and received much attention in the past few decades.

Model accuracies for prediction tasks are heavily influenced by the set of features used to make prediction, where a feature is a real-value function of the input that correlates well with a label or a subset of labels. Breaking up varying-length segments is an extremely successful heuristic, but it also makes extracting certain features from segments difficult. For example, computing durations is difficult for frame-based models. Variants of frame-based models, such as variable-duration HMMs (Ferguson, 1980), are proposed in order to

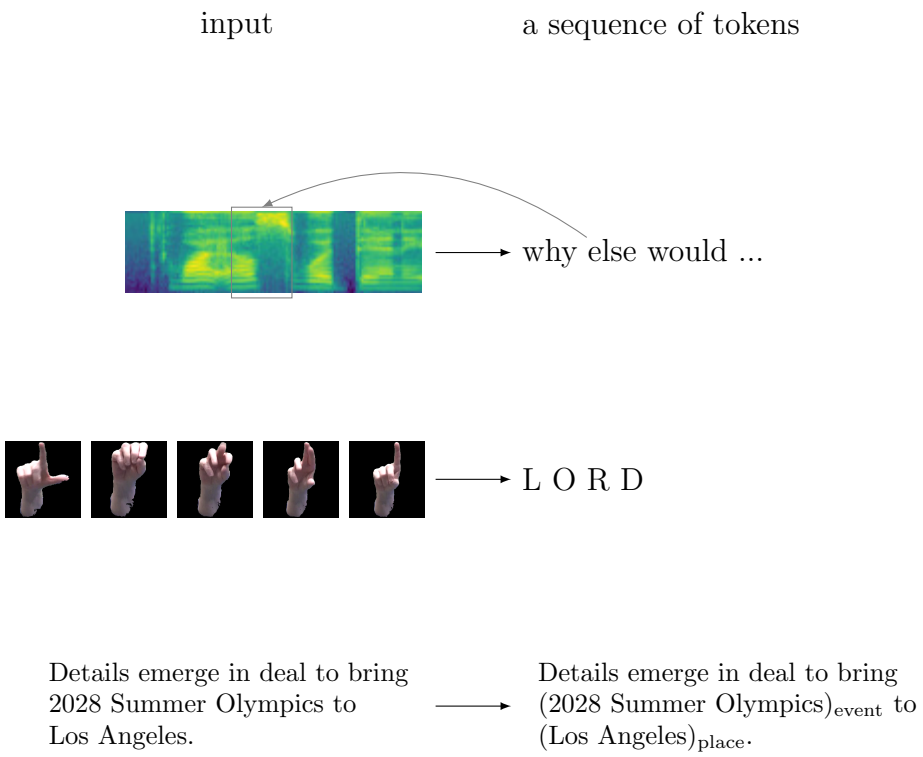
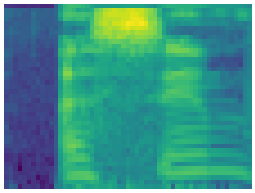
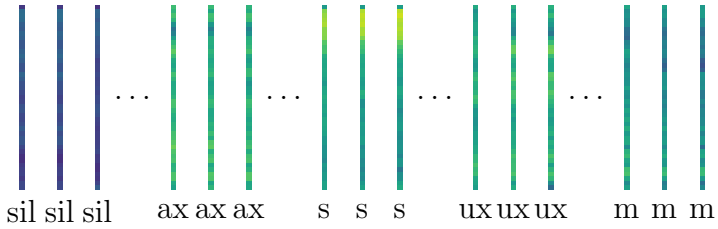


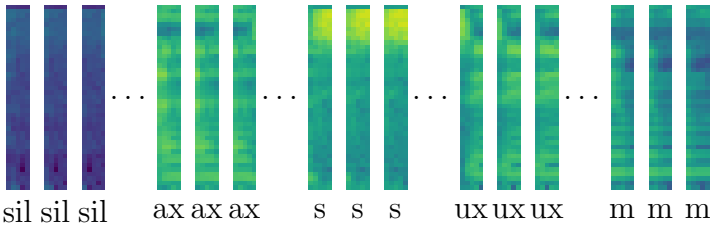
Figure 1.1: Examples of sequence prediction tasks. The first example is speech recognition, where the input is a sequence of acoustic signals and the output is a sequence of words. The second example is American Sign Language fingerspelling recognition, where the input is a sequence of images and the output is a sequence of letters. The third example is named entity recognition, where the input is a sequence of words and the output is the same sequence of words with named entities in parentheses. In the example of speech recognition, the gray arrow connects the word "else" to its corresponding contiguous part in the input sequence.



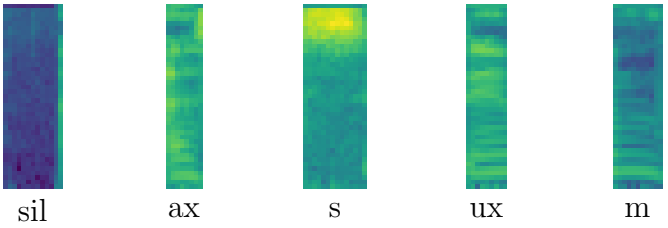
(a) A sequence of input frames.



(b) A frame-based approach with a label sequence of the same length as the input frames.



(c) A frame-based approach with a label sequence of the same the length as the windowed frames.



(d) A segment-based approach with a label sequence aligned to varying-length segments in the input frames.

Figure 1.2: Frame-based approaches and segment-based approaches for sequence prediction.

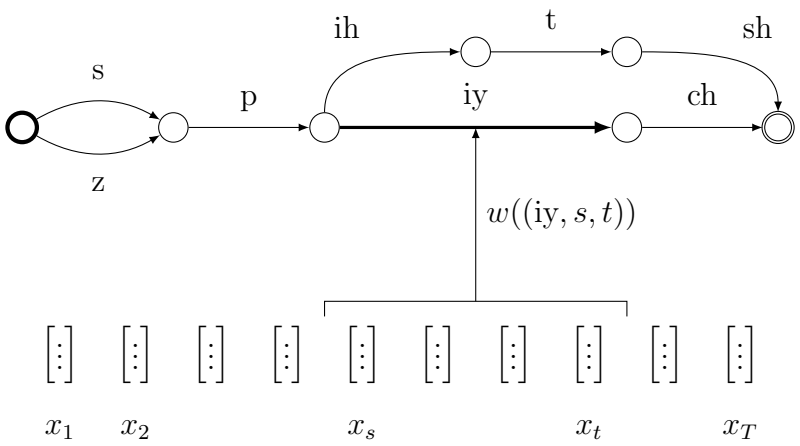


Figure 1.3: An example of a segmental model. A search space is built based on the input frames. Each edge (segment) has a start time, an end time, and a label, which the weight function can make use of. Once the weights of edges are computed, decoding is the problem of finding the maximum-weight path.

overcome this difficulty, but it is impractical to construct a model for every new type of feature. In contrast, segment-based approaches do not assume anything about the segments, so the users are free to use arbitrary information about the segments during prediction. For example, given a segment with a precise start time and end time, computing its duration is trivial. For speech recognition, energy distribution at the left and right boundaries are useful features. For named entity recognition, we can check whether a phrase (segment) is in the dictionary. The flexibility makes segment-based models appealing, and many promising results have been reported in the past (Zue et al., 1989; Ostendorf et al., 1996; Glass, 2003). A comparison of the two approaches is shown in Figure 1.2.

More formally, a segment is a tuple of a start time, an end time, and possibly a label. At a high level, segmental models make predictions by first creating a search space consisting of connected segments, or paths. Segments are given weights based on the start time, end time, and label. The weight values reflect how well the labels match the input. Finding the sequence of segments that best matches the input is equivalent to finding the maximum-weight path. The prediction process is also referred to as **decoding**. Note that the segment weights can be computed by any function as long as it only makes use of the start time, the end time, the label, and the input sequence. An example is shown in Figure 1.3.

Despite their success and attractive properties, segmental models still fall behind frame-based models on many tasks, such as speech recognition (Zweig, 2012). There are many possible reasons behind this. The choice of features extracted from the segments might play a role. However, complex features are typically computationally expensive. The large number of segments we need to consider prohibits the use of computationally expensive segment features. The central goal of this thesis is to improve the efficiency of segmental models, allowing us to explore more computationally expensive features and to improve the

performance of segmental models.

This thesis studies segmental models along three directions: segment features, inference, and learning. We introduce discriminative segmental cascades to speed up inference, allowing us to explore computationally expensive segment representations, such as ones computed with neural networks. We study how segmental models perform when they are trained with various loss functions. Different loss functions have different training requirements, some of them allowing us to train segmental models without manual alignments. We also study whether it is possible to train segmental models end to end from random initialization. We focus on two tasks, phonetic recognition and American Sign Language fingerspelling recognition.

1.1 Preliminaries

In this section, we briefly describe automatic speech recognition (ASR), a task for which segmental models have been studied extensively. We describe a few features that were shown to be useful for disambiguating phonemes, the basic sound units that distinguish words. We then describe why it is hard to incorporate these features in frame-based models, and some of the past efforts to overcome this difficulty.

1.1.1 Automatic speech recognition

In brief, human speech production is the process of mapping a sequence of discrete linguistic units, such as words or phonemes, into waveforms, and automatic speech recognition is about inverting this process, mapping speech waveforms to sequences of discrete linguistic units. On one side of the process, we have speech waveforms, commonly represented as sequences of real-valued vectors called frames. On the other side of the process, we have the discrete units, namely segments, representing phonemes or words.

A waveform can be represented as a sequence of real-valued vectors in multiple ways. One common representation is the amount of sinusoids at different frequencies appearing in the signal. A waveform is first converted into small overlapping chunks. Typically the size of the chunk is 25ms, and a chunk is created every 10ms. Each chunk of waveform is represented as a linear combination of different sinusoids at different frequencies. In particular, a frame in this representation is the vector of coefficients of the linear combination. This representation is known as the **spectrogram**. Examples of spectrograms are shown in Figure 1.2.

Given a sequence of frames, the goal of ASR is to predict what is being said in the waveform, in the form of a sequence of discrete linguistic units, with words and phonemes being the two most common discrete units. Phonemes are defined as the sound units that distinguish words (Ladefoged, 2005), and a sequence of phonemes can be converted into words by looking up the pronunciations of words in a lexicon. In most settings, the set of phonemes and the lexicon are assumed to be given.

A standard evaluation metric for ASR is the **Levenshtein distance** between the predicted label sequence and the ground-truth sequence. Levenshstein distance measures the

minimum number of edits that needs to be performed to transform the predicted label sequence to the ground-truth sequence; hence it is also called the **edit distance**. Formally, the edit distance of two label sequences y and $\hat{y}$ is defined as

$$\text{edit}(\hat{y}, y) = \frac{I + D + S}{|y|}, \tag{1.1}$$

where $|y|$ is the length of y, I, D, and S are the number of insertion, deletion, and substitution, respectively.

ASR is a difficult task due to the wide range of variabilities in the process of speech production. Even in the most controlled setting, the same isolated word can hardly be pronounced the same way twice by the same speaker. Speech production is a highly context-dependent process. Phonemes are pronounced differently when the neighboring phonemes are different (Ladefoged, 2005). Similarly, words are pronounced differently in different contexts. Different speakers pronounce words and phonemes differently due to the differences in their speech organs (Leggetter and Woodland, 1995). Different speaking styles also affect pronunciations. For example, in conversational speech, words are seldom pronounced in the canonical way presented in the lexicon due to the casual speaking style (Livescu, 2005). All of the above variabilities make ASR difficult. It is even more difficult when the speech signals are degraded by noise (Hirsch and Pearce, 2000).

1.1.2 Segment features

Since the goal of speech recognition is to predict words based on their pronunciations, much work has been dedicated to finding features, sometimes referred to as acoustic cues, that identify phonemes and differentiate phonemes. See (Hasegawa-Johnson et al., 2005) and citations therein. Some features are correlated with how humans identify phonemes (Miller and Nicely, 1955). In general, there are many such features that are useful for speech recognition. Each feature alone is probably not enough to identify a phoneme, but a combination of features might be able to (Hasegawa-Johnson et al., 2005). Below we describe two features as examples.

Duration is one of the features that differentiate phonemes. For example, the duration of the long vowel /iy/[1] in the word "seat" is typically longer than short vowels /ih/ in the word "sit" (Peterson and Lehiste, 1960). In this case, duration serves as a good feature to differentiate /iy/ from /ih/. Unvoiced fricatives, such as /f/, /s/, and /sh/, typically have longer duration than voiced fricatives, such /v/, /z/, and /zh/ (Umeda, 1977). Durations have also been shown to help improve recognition results (Anastasakos et al., 1995).

Perhaps the most salient feature for distinguishing phonemes by looking at the spectrogram is the distribution of energy (Ladefoged, 2005). For example, fricatives, such as /s/ and /f/, have evenly spread energy across frequency. Stops, such as /t/ and /p/, have sudden sharp bursts of energy. Bands of high energy in spectrograms, commonly referred to as formants, are another type of energy patterns useful for identifying vowels (Ladefoged,

[1]The phonemes are written in ARPAbet.

2005). The first and second formants (counting from low frequencies) are commonly used to differentiate vowels (Hillenbrand et al., 1995). Formants are also useful for speech recognition (Holmes et al., 1997).

Integrating arbitrary segment features into frame-based models has always been considered a difficult task. Integrating even just the duration requires nontrivial modification to the models (Levinson, 1986). For example, we can expand the label set with durations, but this approach only works for discrete features and generates unnecessarily large search spaces.

Integrating these segment features is also one of the driving forces behind the use and development of segmental models (Zue et al., 1989). While hidden Markov models assume that frame labels follow a Markov process, hidden semi-Markov models assume that segments follow a Markov process. Semi-Markov processes have been used to incorporate duration in frame-based models (Russell and Moore, 1985). Hidden semi-Markov models have been applied to speech recognition with the same reason in mind (Levinson, 1986; Russell and Cook, 1987). The segment-based speech recognizer SUMMIT, is designed with the goal of integrating arbitrary segment features (Zue et al., 1989).

1.1.3 Other sequence prediction tasks

Segmental models have also been applied to other sequence prediction tasks, such as named entity recognition (Sarawagi and Cohen, 2005), American Sign Language fingerspelling recognition (Kim et al., 2013), and action recognition (Simon et al., 2010; Hoai et al., 2011). For named entity recognition, the input is a sequence of words, and the output is a sequence of named entity tags, such as person name, organization name, and location name. For American Sign Language fingerspelling recognition, the input sequence is a sequence of video frames, and the output is a sequence of letters. For action unit detection, the input sequence is a sequence of video frames, and the output is a sequence of actions, such as walk, jump, sit, and stand. In general, these tasks follow a left-to-right (or right-to-left) order in both the input sequence and the output sequence.

A task is said to be linear or monotonic if the task has a left-to-right (or right-to-left) order. One sequence prediction task that is not monotonic is translation (Chiang, 2005). Depending on the source and the target languages, the order of words in the source language can be different from the order in the target language. In this thesis, we only focus on segmental models for monotonic sequence prediction tasks.

1.2 Motivations

We have seen why segmental models are favored over frame-based models when incorporating complex segment features is needed. However, the flexibility of segmental models comes with a price. Enumerating segments of different lengths and of different labels can be time-consuming. Consider an input sequence of length 300, and a label set size of 50. The number of possible segments of length up to 30 is around $30 \times 50 \times 300 = 450000$. If we

make only a single decision at every time point as for frame-based models, we only need to consider $50 \times 300 = 15000$ different decisions. The hypothesis space for segmental models is considerably larger, and as a consequence learning and inference for segmental models are significantly slower than for frame-based models.

Many researchers have attacked the problem of large hypothesis spaces either implicitly or explicitly. The most common approach is to use frame-based models to generate a set of high-confidence segments, and then use segmental models to rescore the segments with segment features (Glass, 2003; Zweig and Nguyen, 2009). Okanohara et al. (2006) explicitly train a model for pruning segments, while Vinh et al. (2011) reuse the computation of features whenever possible. The first approach needs to have two separate models and the second approach depends on the exact form of the features. We would like to design segmental models that do not depend on other external models, and can handle large hypothesis spaces in a feature-oblivious manner.

Designing and implementing state-of-the-art frame-based models requires a significant amount of engineering effort. Phonemes are modeled with three-state hidden Markov models (Schwartz et al., 1984). Since phonemes are influenced heavily by nearby phonemes, phoneme labels that include the previous and the next phonemes, referred to as triphones, are introduced (Schwartz et al., 1984). The large number of triphones makes estimating their probabilities difficult, so triphone parameters are shared using with decision trees (Young et al., 1994). These design decisions in model structure makes engineering difficult. In contrast to frame-based models, the engineering effort in segmental models is put into designing segment features, while the model structure remains the same.

In sum, we aim to design segmental models that perform well on sequence prediction tasks, have a clean and modular mathematical definition, are efficient to train and to decode, and do not depend on other models.

1.3 Previous work

The concept of segmental models was not explicitly defined before the late 1980s, and little work has had segmental models as the primary focus. Weinstein et al. (1975) were one of the early attempts in the 1970s to build speech recognizers based on segment features. In the early 1980s, Cole et al. (1983) developed a system for recognizing isolated English letters based on segment features. Isolated digits as segments were considered in (Kopec and Bush, 1985). Zue et al. (1989) introduced SUMMIT, a segment-based speech recognizer that can handle arbitrary segment features. While the SUMMIT system was later formulated into a probabilistic framework (Glass et al., 1996), others were not probabilistic, and have very few parameters to be estimated.

The probabilistic view of segmental models can also be traced back to the 1980s. As mentioned earlier, Russell and Moore (1985) introduced semi-Markov processes and the follow-up (Russell and Cook, 1987) introduced hidden semi-Markov models to the speech recognition community. These two studies were mainly motivated by the need to include duration in frame-based models. Ostendorf and Roukos (1989) formalized segmental models

as a probabilistic framework to include arbitrary features, but the authors only used frame-based probabilities with one additional duration feature. Russell (1993) and Gales and Young (1993a) further proposed segmental hidden Markov models, but the general form stayed mostly the same. Segmental models proposed in these studies were generative, and were mostly restricted to frame-based Gaussian distributions. This line of work has been summarized in (Ostendorf et al., 1996).

Into the 2000s, studies of segmental models shifted from generative to discriminative. Sarawagi and Cohen (2005) proposed semi-Markov conditional random fields (CRF), the first type of discriminative segmental models. Semi-Markov CRFs require the use of manual alignments to train the models, making them less applicable to speech recognition. Zweig and Nguyen (2009) proposed to use a different training loss for training semi-Markov CRFs, marginalizing over all possible segmentations. This approach alleviated the need for manual alignments to train segmental models. Later, Zweig et al. (2010) used segmental models as second-pass rescoring models for word recognition, showcasing the flexibility of segmental models for incorporating a wide variety of features.

A segmental model is said to be a first-pass model if it exhaustively searches over the entire hypothesis space. Before (Zweig, 2012), all of the segmental models for speech recognition were not first-pass. The early version of SUMMIT (Zue et al., 1989) used a heuristic approach to estimate phoneme boundaries, instead of searching over the entire search space. A more recent version of SUMMIT (Glass et al., 1996) avoided searching exhaustively by using a frame-based model to generate high confidence hypotheses. Semi-Markov CRFs are first-pass segmental models, but have only been applied to natural language tasks, where sequences are typically short and label set size is small.

Zweig (2012) showed that it is feasible to use segmental models as first-pass speech recognizers. Since then, studies of segmental models have moved on to first-pass recognition. At the same time, advances in neural networks have allowed us to learn generic feature representations, so many studies have been devoted to finding better segment representations to replace hand-crafted features. He and Fosler-Lussier (2012) improved upon (Zweig, 2012) by using a 3-layer multilayer perceptron to produce phoneme probabilities. Abdel-Hamid et al. (2013) further improved the results by using a deep convolutional neural network, and it was also the first to train segmental models and neural networks end to end. Along the same line, Lu et al. (2016) proposed end-to-end segmental models with segment representations computed with long short-term memory networks.

1.4 Contributions

In this thesis, we make the following contributions to the development of segmental models.

- We introduce discriminative segmental cascades, allowing us to improve sequence prediction performance using rich features while maintaining efficiency.
- We develop improved understanding of training approaches and training requirements for segmental models. We compare segmental models trained end to end and ones

trained in multiple stages, and compare segmental models trained with and without manual alignments

- Along the way, we present segmental models in a general, modular fashion. We present a unifying framework that encompasses many end-to-end models, such as hidden Markov models, connectionist temporal classification, as special cases.
- We use phonetic recognition and American Sign Language fingerspelling recognition as test beds for comparing segmental models and frame-based models. Segmental models are able to outperform frame-based models in many settings for both sequence prediction tasks.

1.5 Thesis outline

In Chapter 2, we review finite-state transducers (FST), an important tool for describing the hypothesis spaces of segmental models. For the second part of Chapter 2, we review the essential components in speech recognizers, such as language models, lexicons, and hidden Markov models, represented with FSTs. In Chapter 3, we formally define segmental models. The definition is modular and covers many existing segmental models as special cases. In Chapter 4, we introduce discriminative segmental cascades, which allow us to explore various segment representations while maintaining efficiency. In Chapter 5, we study segmental models trained in different training conditions, comparing end-to-end training and multi-stage training. In Chapter 6, we review several variants of segmental models, and describe how they relate to our definition of segmental models. In Chapter 7, we propose a unified framework encompassing many end-to-end frame-based models and segmental models. Finally, in Chapter 8, we discuss possible ways to extend segmental models.

Chapter 2

Background

This chapter describes finite-state transducers (FST), a useful tool for representing and manipulating the search space of a sequence prediction task. This chapter also includes a complete example of a standard speech recognizer based on hidden Markov models represented as FSTs. See (Mohri, 2009) for a comprehensive review of FSTs and their algorithms, and (Mohri et al., 1996) for their applications to speech recognition.

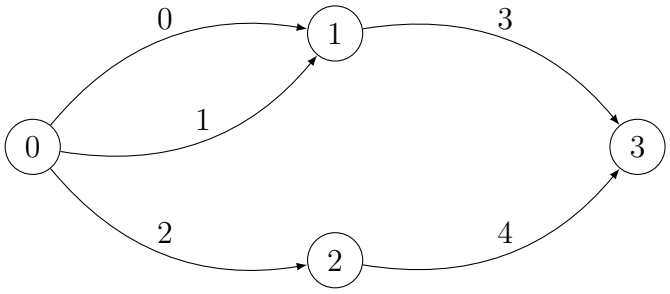
2.1 Finite-State Transducers

We define FSTs based on multigraphs (graphs that allow multiple edges between any two vertices) instead of regular graphs. A **multigraph** G is a tuple $(V, E, \text{tail}, \text{head})$, where V is a set of vertices, E is a set of edges, $\text{tail} : E \to V$ is a function that associates an edge to its tail vertex, and $\text{head} : E \to V$ is a function that associates an edge to its head vertex. Note that E is not a subset of $V \times V$ but a general set, because we allow many edges to have the same tail and head, and a pair of vertices is not enough to uniquely identify an edge. An example of a multigraph is shown in Figure 2.1. We say that $e_1 \in E$ and $e_2 \in E$ are connected if $\text{head}(e_1) = \text{tail}(e_2)$. In addition, we define a **path** of length n as a sequence of connected edges $(e_1, e_2, \ldots, e_n)$ where $\text{head}(e_i) = \text{tail}(e_{i+1})$ for $i = 1, \ldots, n$. A sub-path is a sequence of connected edges within a path. For example, (e_3, e_4, e_5) is a sub-path of $(e_1, e_2, \ldots, e_7, e_8)$.

A **finite-state transducer** is a tuple $(G, \Sigma, \Lambda, I, F, i, o, w)$, where G is a multigraph, Σ is a set of input symbols, Λ is a set of output symbols, $I \subseteq V$ is a set of initial vertices, $F \subseteq V$ is a set of final vertices, $i : E \to \Sigma$ is a function that associates an edge to its input symbol, $o : E \to \Lambda$ is a function that associates an edge to its output symbol, and $w : E \to \mathbb{R}$ is a function that puts weights on edges. An example is shown in Figure 2.2.

An FST can be seen as a function that maps strings to strings, where each path in the graph defines an input-output pair. Specifically, a path $(e_1, e_2, \ldots, e_n)$ associates the input string $i(e_1)i(e_2)\cdots i(e_n)$ with the output string $o(e_1)o(e_2)\cdots o(e_n)$. For example, the FST shown in Figure 2.2 defines the mapping $\{(\texttt{ad}, \texttt{AD}), (\texttt{bd}, \texttt{BD}), (\texttt{ce}, \texttt{CE})\}$.

There is a special symbol called the empty symbol, denoted ϵ. Any string concatenated



e	tail	head
0	0	1
1	0	1
2	0	2
3	1	3
4	2	3

Figure 2.1: A multigraph, where $V = \{0, 1, 2, 3\}$ and $E = \{0, 1, 2, 3, 4\}$. The functions of tail and head are shown in the table on the right.

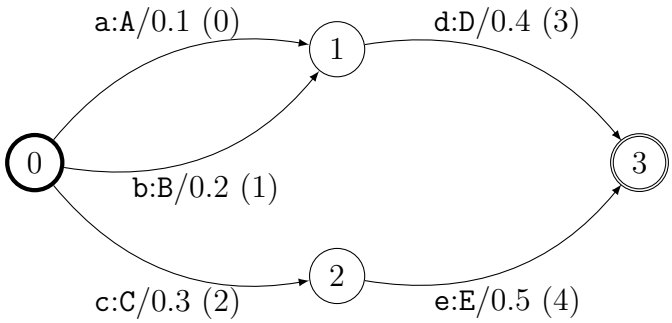


Figure 2.2: An FST based on the multigraph in Figure 2.1, with $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}$ and $\Lambda = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}\}$. The initial vertex is shown in bold, and the final vertex is shown with a doubled circle, i.e., $I = \{0\}$ and $F = \{3\}$. We use σ:λ/w to denote an edge with input symbol σ, output symbol λ, and weight w. The number in parentheses is an identifier of an edge.

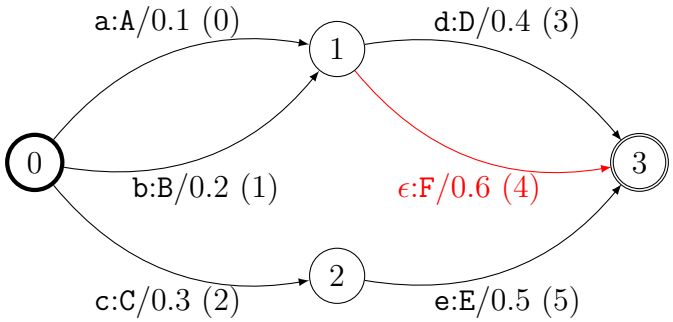


Figure 2.3: An FST with empty symbols. The edge with the empty symbol is highlighted in red.

with an empty symbol is itself, i.e., $\sigma_1\sigma_2\cdots\sigma_n\epsilon = \epsilon\sigma_1\sigma_2\cdots\sigma_n = \sigma_1\sigma_2\cdots\sigma_n$. If the input symbol of an edge is the empty symbol, then we can traverse the edge without consuming any input. If the output symbol of an edge is the empty symbol, then we do not produce any output when traversing the edge. As an example, the FST in Figure 2.3 defines the mapping $\{(\texttt{ad}, \texttt{AD}), (\texttt{a}, \texttt{AF}), (\texttt{bd}, \texttt{BD}), (\texttt{b}, \texttt{BF}), (\texttt{ce}, \texttt{CE})\}$.

We assume there is one unique start vertex and one unique end vertex, i.e., $|I| = |F| = 1$. If an FST has more than one initial vertices, we can always create an additional vertex and connect all initial vertices to the additional vertex with empty symbols (and possibly zero weights) without affecting the string mapping of the FST. For convenience, we also define $\text{in}(v) = \{e \in E : \text{head}(e) = v\}$ and $\text{out}(v) = \{e \in E : \text{tail}(e) = v\}$.

We have reviewed FSTs as graphs and string functions. The third view of FSTs is as state machines. Take the FST in Figure 2.3 for example. To get the output `AD` from the input `ad`, we first feed the character `a` and traverse from vertex 0 to vertex 1, and then feed the character `d` and traverse from vertex 1 to vertex 3. Similarly, to get `BF` from `b`, we first feed the character `b` and traverse from vertex 0 to vertex 1. Due to the empty symbol ϵ, we get `F` and traverse from vertex 1 to vertex 3 without feeding any character. The view of state machines is especially useful when we talk about FST compositions.

2.1.1 Semiring

Other than just manipulating strings, we are also interested in the weights on the edges, especially when paths are considered. We introduce two operators $\otimes$ and $\oplus$, where $\otimes$ is for combining edge weights and $\oplus$ is for combining path weights. For example, by traversing the path $(0, 3)$ that produces `AD` in Figure 2.3, we collect the weight 0.1 from edge 0 and 0.4 from edge 3, and the weight of the path $(0, 3)$ is defined as $w(0) \otimes w(3) = 0.1 \otimes 0.4$.

Suppose in general S is the set of weights. It is desirable to have certain properties, such as commutativity and associativity, for the two operators. For $a \in S$, $b \in S$, and $c \in S$, we say that an operator $*$ is **commutative** if $a*b = b*a$, and that an operator $*$ is **associative** if $(a * b) * c = a * (b * c)$. The element $0 \in S$ is an **identity** with respect to the operator $*$

if $0 * a = a * 0 = a$. The element $0 \in S$ annihilates S with respect to $*$ if $0 * a = a * 0 = 0$. We say that the operator $*$ distributes over the operator $+$ if $a * (b + c) = a * b + a * c$ and $(a + b) * c = a * c + b * c$.

The tuple $(S, \oplus, \otimes)$ is a **semiring** (Mohri, 2002) if $\oplus$ is associative and commutative with identity $0 \in S$, $\otimes$ is associative with identity $1 \in S$, $\otimes$ distributes over $\oplus$, and the element $0 \in S$ annihilates S with respect to $\otimes$. An example of semiring is $(\mathbb{R}, +, \times)$ with 0 being the identity of $+$, and 1 being the identity of $\times$. Another example is the **tropical semiring** $(\mathbb{R} \cup \{\infty\}, \min, +)$ with ∞ being the identity of min, and 0 being the identity of $+$. Yet another example is the **log semiring** $(\mathbb{R} \cup \{-\infty\}, \text{logadd}, +)$ where $\text{logadd}(a, b) = \log(\exp(a) + \exp(b))$, $-\infty$ is the identity of logadd, and 0 is the identity of $+$.

Defining the operators for combining weights in a general way encourages algorithm reuse. As we will see in later chapters, FST algorithms tend to look very similar, and typically the only difference is how the weights are combined. By using general operators, we get a family of algorithms. In other words, we create algorithms almost for free by choosing a proper semiring.

2.1.2 Shortest-path algorithm

Given an FST, we are interested in finding a shortest path from the initial vertex to the final vertex. For example, if the weights are negative log probabilities traversing from one vertex to another, then a shortest path corresponds to one of the most likely paths. We assume there are multiple shortest paths, but finding one of them is enough. We also assume the underlying graph is acyclic, meaning that every path can only traverse a vertex at most once. For acyclic graphs, we have a nice necessary condition for shortest paths—every sub-path within a shortest path is also a shortest path. The argument for the necessary condition is simple. If a sub-path is not a shortest path, then we can always substitute the sub-path with another shorter sub-path to create a shorter path overall.

The necessary condition can be viewed as a recursive condition. Let $\mathcal{P}(u, v)$ be the set of paths from vertex u to vertex v, and let $w(p)$ be a shorthand of $\sum_{e \in p} w(e)$. Suppose we want to compute a shortest path from the vertex u to vertex v. We examine the set of edges $\text{in}(v) = \{e \in E : \text{head}(e) = v\}$ leading into vertex v. If an edge $e \in \text{in}(v)$ is part of the shortest path, then by the necessary condition every path from u to $\text{tail}(e)$ should also be a shortest path. In other words,

$$\min_{p \in \mathcal{P}(u,v)} w(p) = \min_{e \in \text{in}(v)} \min_{p' \in \mathcal{P}(u, \text{tail}(e))} [w(e) + w(p')] \tag{2.1}$$

$$= \min_{e \in \text{in}(v)} \left[w(e) + \min_{p' \in \mathcal{P}(u, \text{tail}(e))} w(p') \right] \tag{2.2}$$

The recursive condition can be computed efficiently with dynamic programming if we store the shortest distance from vertex u to every other vertex. Specifically, let $d(v)$ be the shortest distance from u to v, i.e.,

$$d(v) = \min_{p \in \mathcal{P}(u,v)} w(p). \tag{2.3}$$

By the recursive condition, we have

$$d(v) = \min_{e \in \text{in}(v)} [w(e) + d(\text{tail}(e))]. \tag{2.4}$$

Before computing the minimization in (2.4), we need to make sure the shortest sub-paths are computed. Formally, to compute $d(v)$ for vertex v, we need to make sure $d(\text{tail}(e))$ are computed for $e \in \text{in}(v)$. In other words, if there is a path from vertex w to vertex v, then $d(w)$ should be computed before $d(v)$. We are looking for an order in which w should come before v if there is a path from w to v. An order in which w comes before v if there is a path from w to v is called a **topological order**. To be precise, a topological order is a function $f : V \to \{1, 2, \ldots, |V|\}$ such that $f(w) < f(v)$ if there is a path from w to v. Given an directed acyclic graph, there exist many topological orders. Any one of them would suffice for our dynamic programming.

One way to find a topological order is to run **depth-first search** (DFS) on the reversed graph. The depth-first search algorithm is shown in Algorithm 1. We maintain a stack S. We say that a vertex is traversed if it has been put on S, and we use a set T to track the traversed vertices. When a vertex v is put on stack, we also put a variable indicating whether we have tried to put its neighbors on the stack, where the set of neighbors is defined by $\text{in}(\cdot)$. We say that a vertex is expanded if we have tried to put its neighbors on the stack. It is not hard to see that every vertex is put on the stack twice, once before it is expanded and once after. When a vertex v popped out from the stack is expanded, all vertices that can be traversed from v are also expanded. Since we put a vertex v in O when v and all vertices traversable from v are expanded, the order O is exactly a topological order.

Algorithm 1 is an instance of DFS specifically for computing a topological order by defining the set of neighbors with $\text{in}(\cdot)$. The algorithm can be modified for other purposes, for example, letting the set of neighbors be $\text{out}(\cdot)$ (and changing $\text{tail}(\cdot)$ to $\text{head}(\cdot)$ accordingly) if we want to search the graph in a forward direction instead of backwards. Due to the use of a stack, DFS prefers to follow one path until none of the vertices can be expanded; hence the name depth-first search.

Up to this point, we are interested in finding a path that is shortest. The weight of a path is defined to be the sum of the edge weights, and path weights are combined with the min operator. These operators are the ones used in the tropical semiring, and can be generalized to other semirings, where edge weights are combined with $\otimes$ and path weights are combined with $\oplus$. To be precise, the weight of a path p is defined as $w(p) = \bigotimes_{e \in p} w(e)$, and the distance can be written as

$$d(v) = \bigoplus_{p \in \mathcal{P}(u,v)} w(p) = \bigoplus_{p \in \mathcal{P}(u,v)} \bigotimes_{e \in p} w(e) = \bigoplus_{e \in \text{in}(v)} \bigoplus_{p' \in \mathcal{P}(u, \text{tail}(e))} [w(e) \otimes w(p')] \tag{2.5}$$

$$= \bigoplus_{e \in \text{in}(v)} \left[w(e) \otimes \bigoplus_{p' \in \mathcal{P}(u, \text{tail}(e))} w(p') \right] \tag{2.6}$$

$$= \bigoplus_{e \in \text{in}(v)} [w(e) \otimes d(\text{tail}(e))]. \tag{2.7}$$

Algorithm 1 Depth-First Search (DFS)

Require: Let r be the vertex we start to search.
Ensure: A list of vertices O is returned in topological order.

```
S = {(r, false)}, T = {r}, O = ()
while S is not empty do
    pop (v, z) from S
    if z then
        append v to O
    else
        push (v, true) to S
        for e ∈ in(v) do                                  ▷ Expand v.
            if tail(e) ∉ T then
                push (tail(e), false) to S                ▷ tail(e) is traversed.
                add v to T
            end if
        end for
    end if
end while
```

Algorithm 2 Shortest-Path Algorithm for Directed Acyclic Graphs

Require: Let t be the ending vertex of all paths.
Ensure: The map d contains the shortest distances to v for all vertices.

```
Let O = (o_1, o_2, ..., o_n) be a topological order by running DFS starting from t.
for i = 1, ..., n do
    d(o_i) = min_{e∈in(o_i)} [w(e) + d(tail(e))]
    π(o_i) = argmin_{e∈in(o_i)} [w(e) + d(tail(e))]
end for
```

Algorithm 3 Backtracking

Require: Let s be the initial vertex and t be the final vertex.
Ensure: A shortest path p from s to t.

```
v ← t
p = ()
while v ≠ s do
    add π(v) to p
    v ← tail(π(v))
end while
reverse p
```

Algorithm 4 Generalized Shortest-Distance Algorithm for Directed Acyclic Graphs

Require: Let t be the ending vertex of all paths.
Ensure: The map d contains the shortest distances to v for all vertices.
  Let $O = (o_1, o_2, \ldots, o_n)$ be a topological order by running DFS starting from t.
  for $i = 1, \ldots, n$ **do**
    $d(o_i) = \bigoplus_{e \in \text{in}(o_i)} [w(e) \otimes d(\text{tail}(e))]$
  end for

The generalized shortest-distance algorithm is shown in Algorithm 4. See (Mohri, 2002) for a general treatment of shortest-distance algorithms.

The recursive condition (2.4) only gives us the distance to the final vertex. To obtain the path, we can record down the edge that achieves the minimum while computing the distances. Specifically, let $\pi(v)$ be the edge that achieves the minimum, i.e.,

$$\pi(v) = \underset{e \in \text{in}(v)}{\text{argmin}} [w(e) + d(\text{tail}(e))]. \tag{2.8}$$

We can iteratively collect the optimal edge with π. Since the algorithm goes from the final vertex to the initial vertex, it is commonly called backtracking. The final shortest-path algorithm and the backtracking algorithm are shown in Algorithm 2 and 3.

2.1.3 Composition

Recall that FSTs can be regarded as functions that map strings to strings. It is natural to have multiple FSTs with one FST consuming the outputs of another FST, similar to function composition. Let $T(x, y)$ be the weight of a path in the FST T consuming x as input and producing y as output. For any two FSTs T_1 and T_2, the weight of a path with input x and output y in the composed FST $T_1 \circ T_2$ is defined as

$$(T_1 \circ T_2)(x, y) = \bigoplus_{z \in \mathcal{Z}(x)} T_1(x, z) \otimes T_2(z, y), \tag{2.9}$$

where $\mathcal{Z}(x)$ is the set of output strings that can be produced by feeding x to T_1 (Allauzen and Mohri, 2009). The weight is undefined if $\mathcal{Z}(x)$ is an empty set or there is no path with input z and output y in T_2.

The definition of (2.9) does not provide an explicit structure of the composed FST. In this thesis, we prefer another approach to composing FSTs which we term **structured composition** (Tang et al., 2015). Formally, the structured composition (or σ-composition) of two FSTs $T_1 = (G_1, \Sigma_1, \Lambda_1, I_1, F_1, i_1, o_1, w_1)$ with $G_1 = (V_1, E_1, \text{tail}_1, \text{head}_1)$ and $T_2 = (G_2, \Sigma_2, \Lambda_2, I_2, F_2, i_2, o_2, w_2)$ with $G_2 = (V_2, E_2, \text{tail}_2, \text{head}_2)$ is defined as $T_1 \circ_\sigma T_2 = (G, \Sigma, \Lambda, I, F, i, o, w)$ where $G = (V, E, \text{tail}, \text{head})$ with the following constraints.

$$V = V_1 \times V_2 \qquad E = \left\{ \langle e_1, e_2 \rangle \in E_1 \times E_2 : o_1(e_1) = i_2(e_2) \right\} \tag{2.10}$$

$$\Sigma = \Sigma_1 \qquad i(\langle e_1, e_2 \rangle) = i_1(e_1) \tag{2.11}$$

$$\Lambda = \Lambda_2 \qquad o(\langle e_1, e_2 \rangle) = o_2(e_2) \tag{2.12}$$

$$I = I_1 \times I_2 \qquad \text{tail}(\langle e_1, e_2 \rangle) = \langle \text{tail}_1(e_1), \text{tail}_2(e_2) \rangle \tag{2.13}$$

$$F = F_1 \times F_2 \qquad \text{head}(\langle e_1, e_2 \rangle) = \langle \text{head}_1(e_1), \text{head}_2(e_2) \rangle \tag{2.14}$$

The σ-composed graph is defined over pairs of vertices and pairs of edges. Due to the edge constraints (2.10), the outputs from T_1 have to match the inputs to T_2. An example is shown in Figure 2.4. Note that the definition of the weight function is left to the users. One possible definition is to let

$$w(\langle e_1, e_2 \rangle) = w_1(e_1) \otimes w_2(e_2). \tag{2.15}$$

As a consequence of (2.10), the outgoing edges and incoming edges of a vertex in the σ-composed FST can be computed locally with $\text{out}_1(\cdot)$ and $\text{out}_2(\cdot)$. Specifically,

$$\text{out}(\langle v_1, v_2 \rangle) = \Big\{ \langle e_1, e_2 \rangle : e_1 \in \text{out}_1(v_1), e_2 \in \text{out}_2(v_2), o_1(e_1) = i_2(e_2) \Big\}. \tag{2.16}$$

Since computing $\text{out}(\langle v_1, v_2 \rangle)$ only requires access to edges connected to v_1 and v_2, traversing the composed FST, for example with DFS (Algorithm 1), only needs to keep track of the vertex pairs. The edges can be computed on the fly and do not need to be stored in memory. In general, composing FSTs without explicitly storing the entire graph is commonly known as on-the-fly composition or lazy composition.

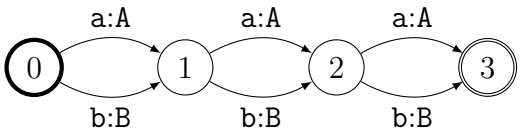
Another way to compute structured composition is to view FSTs as state machines. Take Figure 2.4 for example. We start from vertex 0 in both T_1 and T_2. To get to vertex 1 in T_1, we can either produce `A` or produce `B` as output. In T_2, to go from vertex 0 to vertex 1, we only have the choice to consume `A`. Therefore, in the composed FST $T_1 \circ_\sigma T_2$ we have the edge from $(0, 0)$ to $(1, 1)$ with input `a` and output α. By walking through the vertices one by one, we are essentially performing a search on both FSTs while computing (2.16) on the fly.

2.2 FST-Based Speech Recognizers

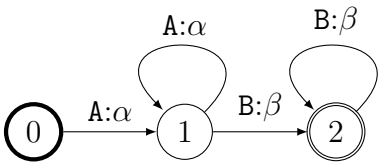
We now have all the necessary tools to describe a basic speech recognizer based on FSTs. Modern speech recognizers are based on a series of string function compositions. Following (Mohri et al., 2002), we represent the string functions as FSTs, and the compositions are realized with structured composition. The first FST U takes acoustic frames as inputs and produces a sequence of frame labels. The second FST H takes in frame labels and converts them into phonemes. The third FST L takes phonemes as inputs and produces words as outputs. The fourth FST G takes words as inputs and produces words as outputs. Finally, the four FSTs are σ-composed into

$$D = U \circ_\sigma H \circ_\sigma L \circ_\sigma G. \tag{2.17}$$

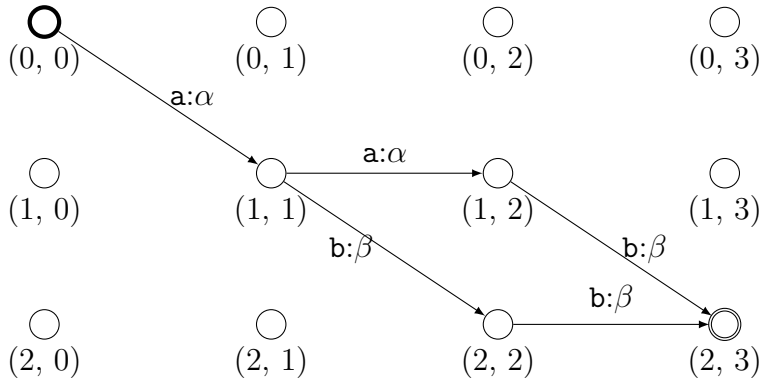
The path weights in D are typically negative log probabilities, so the maximum-weight path is the most likely path based on the probabilities. To find the maximum-weight path, we



(a) FST T_1



(b) FST T_2



(c) FST $T_1 \circ_\sigma T_2$

Figure 2.4: An example of structured composition. Vertices in $T_1 \circ_\sigma T_2$ are labeled with pairs of vertices from T_1 and T_2.

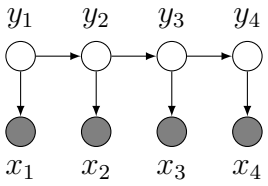


Figure 2.5: A hidden Markov model for 4 frames x_1, x_2, x_3, x_4 with hidden labels y_1, y_2, y_3, y_4. Note that this is a graphical model where vertices are random variables and edges represent conditional dependencies, not an FST. The observed variables are shaded, and the unobserved are not.

simply negate the weights and find the shortest path. The tropical semiring is used to combine the negated weights. In sum, to predict a sequence from a sequence of frames, we create the FST D and find the maximum-weight path in D. Below we describe the four FSTs U, H, L and G in detail.

2.2.1 Hidden Markov models

The dynamics of speech are commonly modeled by hidden Markov models (HMM). The generative story of a hidden Markov model is as follows. We first generate a state y_1 based on a prior distribution and generate a frame x_1 given y_1. A state y_2 is generated given y_1, and a frame x_2 is generated given y_2. In general, for some $t = 2, \ldots, T$, the state y_t is generated given y_{t-1}, and x_t is generated given y_t. For a sequence of T frames $x_1, \ldots, x_T$, the probability distribution defined by an HMM is

$$p(x_{1:T}, y_{1:T}) = p(y_1) \prod_{t=2}^{T} p(y_t|y_{t-1}) \prod_{t=1}^{T} p(x_t|y_t), \tag{2.18}$$

where $x_{1:T}$ is a shorthand for $x_1, \ldots, x_T$, $x_t \in \mathbb{R}^d$ for some $d \in \mathbb{N}$ and $y_t \in \{1, \ldots, S\}$ for some $S \in \mathbb{N}$ for $t = 1, \ldots, T$. An example of a four-frame HMM is shown in Figure 2.5. Note that Figure 2.5 is not an FST but a graphical model where vertices are random variables and edges represent conditional dependencies. The learnable parameters in an HMM are $p(y_1)$, $p(y_t|y_{t-1})$, and $p(x_t|y_t)$ for $t = 1, \ldots, T$. The probabilities $p(y_t|y_{t-1})$ are commonly known as transition probabilities, and $p(x_t|y_t)$ emission probabilities.

The transition probabilities $p(y_t|y_{t-1})$ can be represented as a square matrix A of size $S \times S$ in which $A_{ij} \in [0, 1]$ is the probability of transitioning from state i to state j satisfying $\sum_{j=1}^{S} A_{ij} = 1$. To forbid the transition from state i to state j, we simply let $A_{ij} = 0$. It is common to model a phoneme as a 3-state HMM, and parameterize A as

$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \tag{2.19}$$

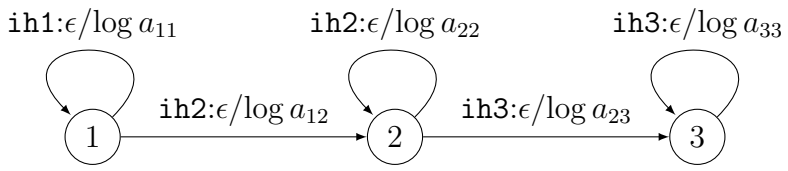


Figure 2.6: A 3-state FST for the phoneme ih, where $\log a_{ij}$ is the log transition probability from state i to state j.

where we are only allowed to either stay in the current state or move to the next state. The allowed transitions can be conveniently represented as an FST, where vertices are states and edges are transitions associated with transition probabilities. An example of the allowed state transitions is shown in Figure 2.6.

To allow transitions from phonemes to phonemes, we connect the FSTs, such as the ones in Figure 2.6, in parallel. An example is shown in Figure 2.7. An additional start state and an additional end state are added. A backward edge from the end state to the start state is also added to allow a sequence of phonemes with indefinite length. The prior distribution $p(y_1)$ entering the phoneme is also added from the start state to each of the phoneme FSTs. Note that when entering one of the phoneme FSTs, the edge consumes a phoneme state and produces a phoneme, and the phoneme FSTs only consume phoneme states and do not produce any output. Silences are modeled with five states rather than three, because they are allowed to be longer than other phonemes. This finishes the construction of the FST H that converts phoneme states to phonemes.

The emission probabilities $p(x_t|y_t)$ in HMM are typically modeled by Gaussian mixture models (GMM). Specifically,

$$p(x_t|y_t) = \sum_{i=1}^{C} \pi_{i,y_t} p(x_t|\mu_{i,y_t}, \sigma^2_{i,y_t}) \tag{2.20}$$

where there are C components for each state in $\{1, \ldots, S\}$, each component $p(x_t|\mu_{i,y}, \sigma^2_{i,y})$ is a Gaussian distribution with mean $\mu_{i,y}$ and variance $\sigma^2_{i,y}$, and $\pi_{i,y}$ is the probability of selecting the i-th component. For a sequence of T frames, the emission probability for each frame can be independently evaluated. Since each state has C Gaussian components, there are $S \times C$ evaluations for each frame. We can build an FST that has $S \times C$ edges for every frame. Each edge has a weight computed from its Gaussian component and has a phone state as the output symbol. An example is shown in Figure 2.8. Note that the FST is constructed by following the generative story of Gaussian mixture models, evaluating a single Gaussian component (selected based on π) for every frame rather than evaluating the weighted average of all Gaussian components.

By σ-composing U and H, we construct an FST where each path in the FST is a sample drawn from the generative story and the weight of each path is the corresponding log probability. To better see the connection, we can traverse the FST U and H synchronously as the way we compute $U \circ_\sigma H$. First when we traverse an edge in U, we select one phoneme state

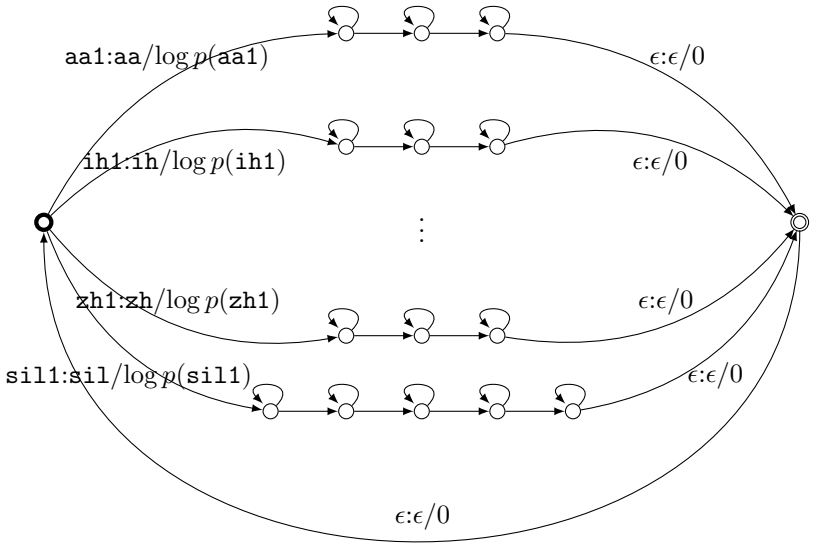


Figure 2.7: An FST H that converts phoneme states to phonemes.

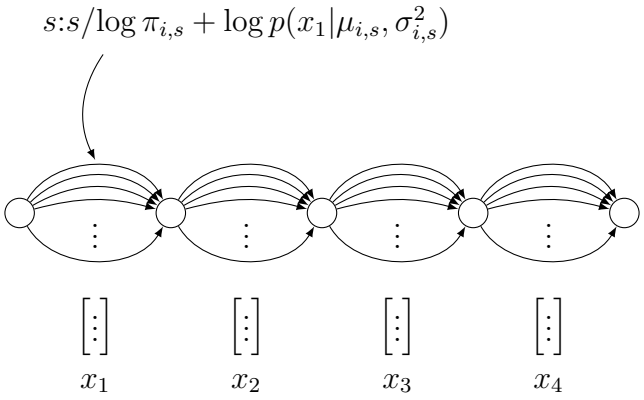


Figure 2.8: An FST U for 4 frames with GMM probabilities. For example, if each phoneme has 3 states, and each state has 64 Gaussian components, then $s \in \{\texttt{aa1}, \texttt{aa2}, \texttt{aa3}, \ldots, \texttt{zh1}, \texttt{zh2}, \texttt{zh3}\}$, $i \in \{1, \ldots, C = 64\}$. If there are L phonemes (i.e., $S = 3L$), then there are $3L \times 64$ (i.e., $S \times C$) edges between each pair of vertices.

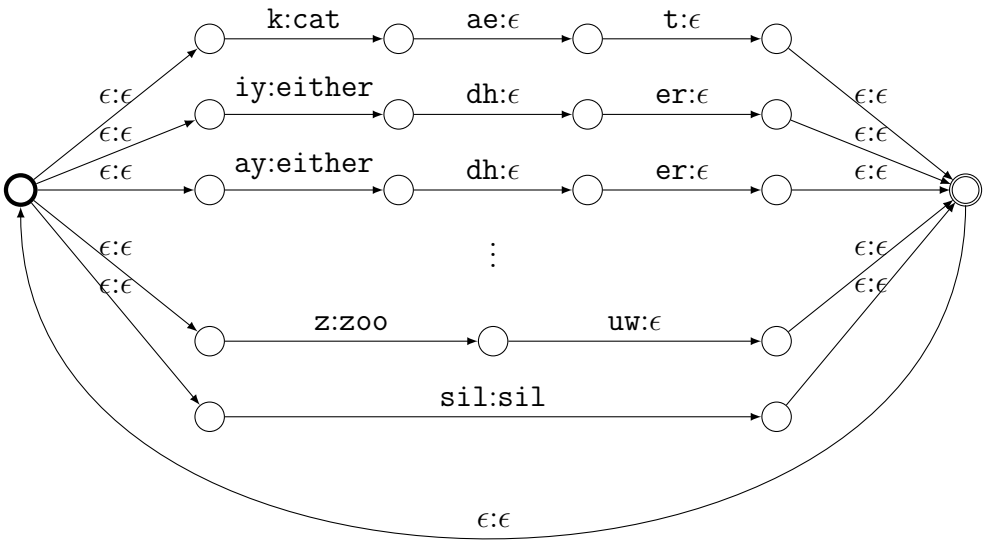


Figure 2.9: An FST L representing a pronunciation dictionary.

and one Gaussian component, collecting the log probability and producing a phoneme state as output. Then the FST H receives the phoneme state and waits for the next phoneme state, collecting transition probabilities and producing a phoneme when necessary. This completes the construction of U and H.

2.2.2 Pronunciation dictionary

The pronunciation dictionary, or lexicon, is a mapping from a word to its pronunciation. To construct an FST representing the lexicon, we create, for each entry in the lexicon, a path consuming a sequence of phonemes and producing a word. Similar to the FST H, we have a start vertex and an end vertex that joins the pronunciation paths, and we also have a backward edge that allows indefinite amount of words to be produced. An example is shown in Figure 2.9. Note that the weights on the edges can either be 0 or the probability of a chosen pronunciation. The FST also allows a word to have multiple pronunciations. Silences are considered as words and are included in the FST. The size of the FST can be significantly reduced if we maintain a prefix tree of the phonemes instead of having parallel pronunciations for every word.

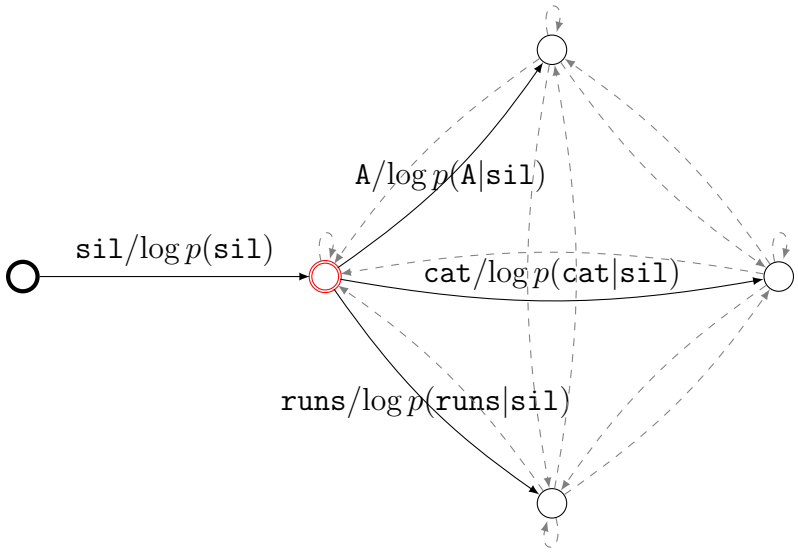


Figure 2.10: An FST G representing a bigram language model for the vocabulary {`sil`, `A`, `cat`, `runs`}. Some edges are dashed to avoid clutter. The output symbols are the same as the input symbols for every edge, hence ignored. The vertex in red is the silence state.

2.2.3 Language models

Language models assign probabilities to word sequences. For any $k \in \mathbb{N}$, a word sequence $w_{1:k} = w_1, \ldots, w_k$ of length k has probability

$$p(w_{1:k}) = \prod_{i=1}^{k} p(w_i | w_{1:i-1}). \tag{2.21}$$

As the sequence gets longer, i.e., as i gets larger, $p(w_i|w_{1:i-1})$ depends on more words, which makes estimating the probabilities difficult. A simple approach to remedy this problem is to introduce the Markov assumption

$$p(w_i|w_{1:i-1}) = p(w_i|w_{i-n+1:i-1}). \tag{2.22}$$

A language model that gives a probability for the current word based on the previous $n-1$ words is commonly known as an n-gram language model. To construct an FST representing a language model, we create vertices that corresponds to history words $w_{i-n+1:i-1}$. For each vertex, the outgoing edges produce the next word w_i with the weight $\log p(w_i|w_{i-n+1:i-1})$. A path in this FST consumes a sequence of words and has the probability of the word sequence assigned by the language model. There is a start vertex representing the state of having no history words. We assume the utterance starts and ends with silences, so there is only one edge going out from the start state expecting a silence. An example is shown in Figure 2.10. Back-off language models can also be approximated with FSTs (Allauzen et al., 2003).

2.3 Summary

In this chapter, we have reviewed the definition of finite-state transducers (FST) and constructed a basic speech recognizer by σ-composing an HMM emission FST U, an HMM transition FST H, a lexicon L, and a language model G. Since $H \circ_\sigma L \circ_\sigma G$ is shared across all utterances, it is typically σ-composed and saved. Each individual FST and the composed FST can also be determinized and minimized for efficiency, which we do not cover. Interested readers should refer to (Mohri et al., 2002) for further details.

Chapter 3

Discriminative Segmental Models

The problem of prediction in general can be considered as a search problem. Given an input x, we first construct a set of possible outputs $\mathcal{Y}(x)$, referred to as the search space. For every output hypothesis y in the search space $\mathcal{Y}(x)$, we measure how well the output matches the input, assigning a weight to each pair (x, y). Prediction can be considered as finding the hypothesis $\hat{y}$ such that the weight of $(x, \hat{y})$ is larger than any other pairs.

Extending the above paradigm, the problem of sequence prediction, such as speech recognition as we have seen in Chapter 2, can be considered as a search problem. Given an input sequence, the search space in this case is a set of sequences of connected segments. The number of such sequences is exponential in the number of possible segments. Fortunately, we do not need to store exponentially many such sequences, and can represent the search space compactly as an finite-state transducer (FST). Consider an FST with vertices corresponding to time points, and edges corresponding to segments. A path in the FST corresponds to a sequence of connected segments, and the set of all paths defined by the FST is the search space. The FST is weighted, and the weights assigned to the edges are based on how well the segments match the input. Prediction can be considered as finding the maximum-weight path, because the maximum-weight path, by definition, is a path that best matches the input.

In this chapter, we formally define segmental models, including search spaces constructed from sequences of input vectors, weight functions that measure how well a segment matches the acoustic signals, and loss functions used for training segmental models.

3.1 Preliminaries

Let $\mathcal{X}$ be the input space, the set of all sequences of real-valued vectors, e.g., log mel filter bank features or mel frequency cepstral coefficients (MFCCs). Specifically, for a sequence of T vectors $x = (x_1, \ldots, x_T) \in \mathcal{X}$, each $x_t \in \mathbb{R}^d$ is a d-dimensional vector, also referred to as a **frame**, for $t \in \{1, \ldots, T\}$. Let $\mathcal{Y}$ be the output space, the set of all label sequences, where each label in a label sequence comes from a label set L, e.g., a phoneme set in the case of phoneme recognition. Given any T frames, a **segmentation** of length K is a sequence of time

points $((1 = s_1, t_1), \ldots, (s_K, t_K = T))$, where $s_k \leq t_k$ and $t_k + 1 = s_{k+1}$ for $k \in \{2, \ldots, K\}$. A **segment** (typically denoted e in later sections) is a tuple (ℓ, s, t) where $\ell \in L$ is its label, s is the start time, and t is the end time.

A **segmental model** is a tuple (Θ, w) where Θ is a set of parameters, and $w : \mathcal{X} \times E \to \mathbb{R}$ is a weight function parameterized by Θ and E is the set of all segment tuples (ℓ, s, t). A sequence of segments forms a **path**. Specifically, a path of length K is a sequence of segments $(e_1, \ldots, e_K)$, where $e_k \in E$ for $k \in \{1, \ldots, K\}$. Let $\mathcal{P}$ be the set of all paths. For any path p, we overload w such that $w(x, p) = \sum_{e \in p} w(x, e)$. We will also abbreviate $w(x, e)$ and $w(x, p)$ as $w(e)$ and $w(p)$ respectively when the context is clear. The concrete form of the weight function will be defined in later sections.

Given an input $x \in \mathcal{X}$, segmental models aim to solve sequence prediction by reducing it to finding the maximum-weight path

$$\operatorname*{argmax}_{p \in \mathcal{P}} w(x, p). \tag{3.1}$$

The set of paths $\mathcal{P}$, also referred to as the **search space**, can be compactly represented as an FST. Once we have the search space FST, we can simply invoke Algorithm 2 to find the maximum-weight path. We will describe how the search space FST is constructed in the next section.

A segmental model can be trained by finding a set of parameters that minimizes a loss function. We emphasize that the model definition is not tied to any loss function, allowing us to study the behavior of segmental models under different loss functions. We define various loss functions for training segmental models in Section 3.4, and discuss how the properties of the losses affect the training requirement.

3.2 Search Space

To represent the set of paths $\mathcal{P}$ as an FST, we place a vertex at every time point and connect vertices based on the set of segments. Suppose we have T frames. The set of segments E is an exhaustive enumeration of tuples (ℓ, s, t) for all $\ell \in L$ and $1 \leq s \leq t \leq T$. We create a set of vertices $V = \{v_0, v_1, \ldots, v_T\}$ and a time function $\tau : V \to \mathbb{N}$ such that $\tau(v_t) = t$ for $t \in \{0, 1, \ldots, T\}$. For every segment $(\ell, s, t) \in E$, we create an edge e such that $i(e) = o(e) = \ell$, $\text{tail}(e) = v_{s-1}$, and $\text{head}(e) = v_t$. In other words, for any $e \in E$, the corresponding segment $(\ell, s, t) = (o(e), \tau(\text{tail}(e)), \tau(\text{head}(e)))$. As a result, we will use $w(e)$ and $w((\ell, s, t))$ interchangeably. We set $\Sigma = \Lambda = L$, $I = \{v_0\}$, and $F = \{v_T\}$ to complete the construction of the FST given any T frames.

The number of segments in the graph is $O(|L|T^2)$. To reduce the size of the search space, a maximum duration constraint is typically imposed while creating the search space. Specifically, we only create segments (ℓ, s, t) with $t - s + 1 \leq D$, for some maximum duration D. Adding such constraint makes the number of segments $O(|L|TD)$. An example of a search space is shown in Figure 3.1.

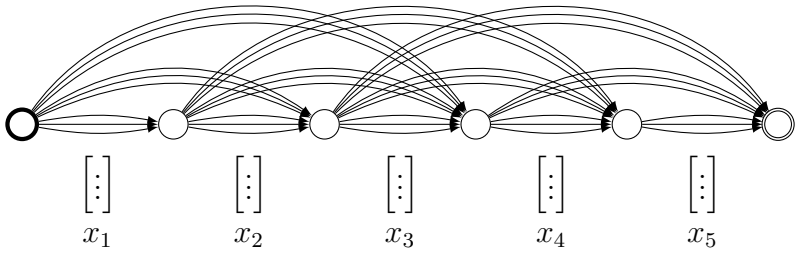


Figure 3.1: An example search space for a five-frame input sequence with a label set L of size three and maximum segment duration D of two frames, i.e., $T = 5$, $|L| = 3$, and $D = 3$. The three edges between any two nodes are associated with the three labels.

3.3 Weight Functions

Here we detail two types of weight functions based on prior work by ourselves and others (He and Fosler-Lussier, 2012; Tang et al., 2015; Abdel-Hamid et al., 2013; Lu et al., 2016). We will compare the two weight functions and in Chapter 5. The term feature function is often used in the literature to denote the function $\phi : \mathcal{X} \times E \to \mathbb{R}^m$ for some m, when the weight function $w(x, e)$ is of the form $\theta^\top \phi(x, e)$ for some parameter vector $\theta \in \mathbb{R}^m$. In general, the weight function need not be a dot product, but can be any (sub-)differentiable real-valued function.

A weight function is typically a composition of two steps: first the input vectors are converted into an intermediate representation, and second the intermediate representation is converted into segment weights. The function in the first step that converts input vectors to an intermediate representation are called an **encoder**. Specifically, an encoder is a function that takes in T frames $x_1, \ldots, x_T$ and outputs $\tilde{T}$ feature vectors $h_1, \ldots, h_{\tilde{T}}$. The number of output vectors $\tilde{T}$ can be different from the number of input vectors T depending on the encoder architecture. We defer the actual implementation of encoders in the experimental sections. Here we define weight functions based on $h_1, \ldots, h_{\tilde{T}}$. We use Θ_{enc} to denote the parameters for the encoder, and let Θ_{dec} be the remaining parameters in the weight function. Note that $\Theta = \Theta_{\text{enc}} \cup \Theta_{\text{dec}}$.

3.3.1 FC weight function

The following weight function, termed FC weight, is based on a frame classifier and is similar to weight functions used in a variety of prior work (He and Fosler-Lussier, 2012; Tang et al., 2015; Abdel-Hamid et al., 2013). The frame classifier takes in the output $h_1, \ldots, h_{\tilde{T}}$ from the encoder, and produces a sequence of log probability vectors over the labels

$$z_i = \text{logsoftmax}(W h_i + b) \tag{3.2}$$

where $z_i \in \mathbb{R}^{|L|}$ and W and b are the parameters, for $i \in \{1, \ldots \tilde{T}\}$. Based on these posterior vectors, we define several functions that summarize the posteriors over a segment:

frame average The average of transformed log probabilities

$$w_{\text{avg}}((\ell, s, t)) = \frac{1}{t-s+1} \sum_{i=s}^{t} (u_i)_\ell, \tag{3.3}$$

where $u_i = W_{\text{avg}} z_i$ for $i \in \{1, \ldots, \tilde{T}\}$.

frame samples A sample of transformed log probabilities

$$w_{\text{spl-}j}((\ell, s, t)) = (u_j)_\ell \tag{3.4}$$

at time $j \in \{(t-s)/6, (t-s)/2, 5(t-s)/6\}$, where $u_i = W_{\text{spl}} z_i$ for $i \in \{1, \ldots, \tilde{T}\}$.

boundary The average of transformed log probabilities around the left boundary (start) of the segment

$$w_{\text{left}-k}((\ell, s, t)) = (u_{i-k})_\ell \tag{3.5}$$

and around the right boundary (end)

$$w_{\text{right}-k}((\ell, s, t)) = (u'_{i+k})_\ell \tag{3.6}$$

where $u_i = W_{\text{left}} z_i$ and $u'_i = W_{\text{right}} z_i$ for $i \in \{1, \ldots, \tilde{T}\}$, and $k = 1, 2, 3$.

duration The label-dependent duration weight

$$w_{\text{dur}}((\ell, s, t)) = d_{\ell, t-s}. \tag{3.7}$$

bias A label-dependent bias

$$w_{\text{bias}}((\ell, s, t)) = b'_\ell. \tag{3.8}$$

The final FC weight function is the sum of all the above weight functions. When the FC weight function is used, Θ_{dec} is $\{W_{\text{avg}}, W_{\text{spl}}, W_{\text{left}}, W_{\text{right}}, d, b'\}$. Note that $\{W, b\}$ are considered parameters of the encoders.

3.3.2 MLP weight function

The MLP weight function based on a multi-layer perceptron (MLP) is inspired by (Tang et al., 2015; Kong et al., 2016; Lu et al., 2016). Two hidden layers

$$z^{(1)}_{\ell,s,t} = \text{ReLU}(W_1[h_s; h_t; c_\ell; d_{\lfloor \log_{1.6}(t-s) \rfloor}] + b_1) \tag{3.9}$$

$$z^{(2)}_{\ell,s,t} = \tanh(W_2 z^{(1)}_{\ell,s,t} + b_2) \tag{3.10}$$

are computed directly from the outputs $h_1, \ldots, h_{\tilde{T}}$ of the encoder before computing the final weight, where c_ℓ is a label embedding vector for the label ℓ, d_k is a duration embedding vector

for the duration k in log scale[1], with $k = \lfloor \log_{1.6}(t - s + 1) \rfloor$, and $\text{ReLU}(x) = \max(x, 0)$. The final weight for the segment is defined as

$$w((\ell, s, t)) = \theta^\top z^{(2)}_{\ell,s,t}. \tag{3.11}$$

When the MLP weight function is used, Θ_{dec} is $\{W_1, b_1, W_2, b_2, \theta\}$. Although the MLP weight function is conceptually simple, it is more expensive to compute than the FC weight function. In (Kong et al., 2016; Lu et al., 2016), an LSTM is created for each segment consuming the outputs of the encoder, followed by an MLP taking the output vectors of the per segment LSTM at the segment boundary. In order to reduce the computation, we discard the per segment LSTM, and use the vectors produced by the encoder at the segment boundary as input to the MLP.

3.4 Losses

Recall that a path $p = ((\ell_1, s_1, t_1), \ldots, (\ell_K, s_K, t_K))$ consists of a label sequence $y = (\ell_1, \ldots, \ell_K)$ and a segmentation $z = ((s_1, t_1), \ldots, (s_K, t_K))$. In the following, we will use (y, z) and p interchangeably. We will also denote the space of all segmentations $\mathcal{Z}$.

Training a segmental model aims to find a set of parameters Θ that minimizes the expected task loss, for example, the expected edit distance

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[\text{edit}(h_\Theta(x), y)] \tag{3.12}$$

where h is the inference algorithm (Algorithm 2) parameterized with Θ, edit computes the edit distance between two sequences, and the expectation is taken over samples $(x, y) \in \mathcal{X} \times \mathcal{Y}$ drawn from a distribution $\mathcal{D}$. The expectation can be decomposed into two steps

$$\mathbb{E}_{x\sim\mathcal{D}(x)}\mathbb{E}_{y\sim\mathcal{D}(y|x)}[\text{edit}(h_\Theta(x), y)], \tag{3.13}$$

first sampling x and then sampling y. To obtain a good discriminator h, it suffices to optimize the inner expectation

$$\mathbb{E}_{y\sim\mathcal{D}(y|x)}[\text{edit}(h_\Theta(x), y)], \tag{3.14}$$

for any x drawn from $\mathcal{D}(x)$. However, the edit distance, due to its discrete nature, is difficult to optimize, so instead we minimize the expected loss

$$\mathbb{E}_{y\sim\mathcal{D}(y|x)}[\mathcal{L}(\Theta; x, y)], \tag{3.15}$$

where $\mathcal{L}$ is a surrogate loss function. If segmentations are considered in the loss function, then we can minimize

$$\mathbb{E}_{(y,z)\sim\mathcal{D}'(y,z|x)}[\mathcal{L}(\Theta; x, y, z)], \tag{3.16}$$

where $\mathcal{D}'$ is a conditional distribution over $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$.

[1]We use 5 different duration embedding vectors for our experiments. The base 1.6 is chosen such that $k \in \{0, \ldots, 4\}$.

Since the distribution $\mathcal{D}$ is unknown, we use a tranining set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of size n to approximate the expectation and instead minimize

$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(\Theta; x_i, y_i). \tag{3.17}$$

If segmentations are considered, we use a tranining set $S = \{(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)\}$ of size n to approximate $\mathcal{D}'$ and minimize

$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(\Theta; x_i, y_i, z_i). \tag{3.18}$$

The connection between the surrogate loss $\mathcal{L}$ and the edit distance depends on the choice of loss. We will optimize the loss functions with first-order methods, such as stochastic gradient descent. When listing the function definitions along with reasons for using them, we will also list the (sub-)gradients with respect to the weight $w(e)$ for some edge e. We assume the weight function w is (sub-)differentiable and the (sub-)gradients with respect to the parameters can be obtained with back-propagation.

3.4.1 Hinge loss

Given an utterance x and a ground truth path $p = (y, z)$, the hinge loss is defined as

$$\mathcal{L}_{\text{hinge}}(\Theta; x, p) = \max_{p' \in \mathcal{P}} \left[\text{cost}(p', p) - w(p) + w(p')\right] \tag{3.19}$$

where cost is a user-defined cost function. The connection between the hinge loss and the task loss is through the cost function. Suppose $\hat{p} = \text{argmax}_{p \in \mathcal{P}}\, w(p)$ is the maximum-weight path found by Algorithm 2. The cost of the inferred path $\hat{p}$ against the ground truth p can be upper-bounded by the hinge loss:

$$\text{cost}(\hat{p}, p) \leq \text{cost}(\hat{p}, p) - w(p) + w(\hat{p}) \leq \mathcal{L}_{\text{hinge}}(\Theta; x, p). \tag{3.20}$$

When the cost function is the edit distance, minimizing the hinge loss is minimizing an upper bound on the edit distance of the predicted sequence.

The hinge loss itself is difficult to optimize when the cost function is the edit distance. In practice, the cost function is assumed to be decomposable

$$\text{cost}(p', p) = \sum_{e' \in p'} \text{cost}(e', p) \tag{3.21}$$

to allow efficient dynamic programming. When the cost is decomposable, the hinge loss can be written as

$$\mathcal{L}_{\text{hinge}}(\Theta; x, p) = \max_{p' \in \mathcal{P}} \left[\sum_{e' \in p'} \text{cost}(e', p) - \sum_{e \in p} w(e) + \sum_{e' \in p'} w(e')\right] \tag{3.22}$$

$$= \max_{p' \in \mathcal{P}} \sum_{e' \in p'} \left[\text{cost}(e', p) + w(e')\right] - \sum_{e \in p} w(e), \tag{3.23}$$

and the max operator in the first term can be solved with Algorithm 2 by adding the costs to the weights for all segments.

A subgradient of the hinge loss with respect to $w(e)$ is

$$\frac{\partial \mathcal{L}_{\text{hinge}}(\Theta; x, p)}{\partial w(e)} = -\mathbb{1}_{e \in p} + \mathbb{1}_{e \in \tilde{p}} \tag{3.24}$$

where

$$\tilde{p} = \operatorname*{argmax}_{p' \in \mathcal{P}} [\text{cost}(p', p) + w(p')], \tag{3.25}$$

which is the path that maximizes the first term in the hinge loss, and can be obtained with Algorithm 2 with the cost added.

Linear models are called support vector machines (SVM) when trained with the hinge loss, and are referred to as structured SVMs when used for solving structured prediction problems, e.g., sequence prediction in our case. A hinge loss with cost zero is also called perceptron loss (LeCun and Huang, 2005). Segmental models trained with the hinge loss have been studied in (Zhang and Gales, 2013; Tang et al., 2014, 2015).

3.4.2 Log loss

Segmental models can be treated as probabilistic models by defining probability distributions on the set of all paths $\mathcal{P}$. Specifically, the probability of a path $p = (y, z)$ is defined as

$$P(y, z|x) = P(p|x) = \frac{1}{Z(x)} \exp(w(x, p)) \tag{3.26}$$

where

$$Z(x) = \sum_{p' \in \mathcal{P}} \exp(w(x, p')) \tag{3.27}$$

is the partition function. Given an input x and a ground truth path p, the log loss is defined as

$$\mathcal{L}_{\log}(\Theta; x, p) = -\log P(p|x). \tag{3.28}$$

Minimizing the log loss is equivalent to maximizing the conditional likelihood. In addition, the conditional likelihood can be written as

$$P(y, z|x) = \mathbb{E}_{(y', z') \sim P(y', z'|x)}[\mathbb{1}_{(y', z') = (y, z)}] \tag{3.29}$$

$$= 1 - \mathbb{E}_{(y', z') \sim P(y', z'|x)}[\mathbb{1}_{(y', z') \neq (y, z)}]. \tag{3.30}$$

Therefore, maximizing the conditional likelihood is equivalent to minimizing the expected zero-one loss

$$\mathbb{E}_{(y', z') \sim P(y', z'|x)}[\mathbb{1}_{(y', z') \neq (y, z)}], \tag{3.31}$$

where $P(y, z|x)$ is used to approximate $\mathcal{D}'(y, z|x)$ and the zero-one loss $\mathbb{1}_{(y', z') \neq (y, z)}$ is used. The use of the log loss is justified because the expectation above can be seen as an approximation of (3.16). Segmental models trained with log loss have been referred to as semi-Markov

CRFs (Sarawagi and Cohen, 2005). Minimizing log loss is equivalent to maximizing mutual information, and is commonly referred to as the MMI criterion (Bahl et al., 1986).

Since the weight for the ground truth path p can be efficiently computed, we are left with the problem of computing the partition function $Z(x)$. The partition function can also be computed efficiently with the following dynamic programming algorithm. Let $\mathcal{P}(u, v)$ be the set of paths that start at vertex u and end at vertex v. For any vertex v, define the forward marginal as

$$\alpha(v) = \log \sum_{p' \in \mathcal{P}(v_0, v)} \exp(w(p')). \tag{3.32}$$

By expanding the edges ending at v, we have

$$\begin{aligned}
\alpha(v) &= \log \sum_{p' \in \mathcal{P}(v_0, v)} \exp\left(\sum_{e \in p'} w(e)\right) && (3.33)\\
&= \log \sum_{e \in \text{in}(v)} \sum_{p' \in \mathcal{P}(v_0, \text{tail}(e))} \exp\left(w(e) + \sum_{e' \in p'} w(e')\right) && (3.34)\\
&= \log \sum_{e \in \text{in}(v)} \exp\left(w(e) + \alpha(\text{tail}(e))\right) && (3.35)
\end{aligned}$$

Similarly, the backward marginal at v is defined as

$$\beta(v) = \log \sum_{p' \in \mathcal{P}(v, v_T)} \exp(w(p')), \tag{3.36}$$

and has similar recursive structure. The complete algorithm is shown in Algorithm 5. Once all entries in α and β are computed, the log partition function is

$$\log Z(x) = \alpha(v_T) = \beta(v_0). \tag{3.37}$$

Note that we store all of the entries in log space to maintain numerical stability.

The gradient of the log loss with respect to $w(e)$ is

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\log}(\Theta; x, p)}{\partial w(e)} &= -\mathbb{1}_{e \in p} + \frac{1}{Z(x)} \sum_{p' \ni e} \exp(w(p')) && (3.38)\\
&= -\mathbb{1}_{e \in p} + \exp\Big[\alpha(\text{tail}(e)) + w(e) + \beta(\text{head}(e)) - \log Z(x)\Big], && (3.39)
\end{aligned}$$

which can also be efficiently computed once the marginals are computed.

Note that the form of Algorithm 5 is very similar to that of Algorithm 2. Recall that to get the standard shortest-path algorithm (Algorithm 2), we simply use the tropical semiring when running Algorithm 4. If the tropical semiring is replaced by the log semiring, we arrive at Algorithm 5, demonstrating the generality of FST algorithms and semirings.

Algorithm 5 Computing forward and backward marginals

$\alpha(v_0) = 0$
$\beta(v_T) = 0$
$\text{logadd}(a, b) = \log(\exp(a) + \exp(b))$
for $v = v_0, v_1, \ldots, v_T$ **do**
  $\alpha(v) = \text{logadd}_{e \in \text{in}(v)} \Big[\alpha(\text{tail}(e)) + w(e)\Big]$
end for
for $v = v_T, v_{T-1}, \ldots, v_0$ **do**
  $\beta(v) = \text{logadd}_{e \in \text{out}(v)} \Big[\beta(\text{head}(e)) + w(e)\Big]$
end for

Log loss can be modified to include a cost function. Speficically, we define

$$P_p^+(p'|x) = \frac{1}{Z(x,p)} \exp(w(x,p') + \text{cost}(p',p)) \tag{3.40}$$

where p is the ground-truth path and

$$Z(x,p) = \sum_{p'' \in \mathcal{P}} \exp(w(x,p'') + \text{cost}(p'',p)). \tag{3.41}$$

Unlike P, the distribution P^+ considers both the weights and the costs. The modified log loss, also known as the boosted MMI criterion (Povey et al., 2008), is defined as

$$\mathcal{L}_{\text{bMMI}}(\Theta; x, p) = -\log P_p^+(p|x). \tag{3.42}$$

3.4.3 Marginal log loss

Given an input x and a label sequence y, the marginal log loss is defined as

$$\mathcal{L}_{\text{mll}}(\Theta; x, y) = -\log P(y|x) = -\log \sum_{z \in \mathcal{Z}} P(y, z|x) \tag{3.43}$$

where the segmentation is marginalized compared to log loss; hence the name. Following the same argument as for the log loss, the marginal distribution can be written as

$$P(y|x) = 1 - \mathbb{E}_{y' \sim P(y'|x)}[\mathbb{1}_{y \neq y'}], \tag{3.44}$$

and maximizing the marginal distribution is equivalent to minimizing the expected zero-one loss

$$\mathbb{E}_{y' \sim P(y'|x)}[\mathbb{1}_{y \neq y'}], \tag{3.45}$$

where $P(y|x)$ is used to approximate $\mathcal{D}(y|x)$. Note that the zero-one loss $\mathbb{1}_{y \neq y'}$ only depends on the label sequence. While the log loss has a connection to (3.16), the marginal log loss justifies its use by directly approximating (3.15) with the above expected zero-one loss.

Note that both the hinge loss and the log loss depend on the ground-truth path, or more specifically, the ground-truth segmentation. The marginal log loss marginalizes over the segmentations, so it only depends on the ground-truth label sequence and not the segmentation. The lack of reliance on the ground-truth segmentation makes the marginal log loss attractive for tasks such as speech recognition, because collecting ground-truth segmentations for phonemes or words is time-consuming and/or expensive. In addition, the boundaries of phonemes and words tend to be ambiguous, so it can be preferable to leave the decision to the model. Segmental models trained with the marginal log loss have been referred to as segmental CRFs (Zweig and Nguyen, 2009).

To compute the marginal log loss, we can rewrite it as

$$\mathcal{L}_{\text{mll}}(\Theta; x, y) = -\log \sum_{z \in \mathcal{Z}} P(y, z|x) \tag{3.46}$$

$$= -\log \sum_{z \in \mathcal{Z}} \exp(w(x, (y, z))) + \log Z(x) \tag{3.47}$$

$$= \underbrace{-\log \sum_{p' : \Gamma(p') = y} \exp(w(x, p'))}_{\log Z(x,y)} + \log Z(x) \tag{3.48}$$

where Γ extracts the label sequence from a path, i.e., for $p' = (y', z')$, $\Gamma(p') = y'$. Since the partition function can be efficiently computed from Algorithm 5, we only need to compute $\log Z(x, y)$. Since the term $\log Z(x, y)$ is identical to $\log Z(x)$ except that it involves a constrained search space considering all paths with the same label sequence y, the strategy is to construct the constrained search space with an FST and run Algorithm 5 on the FST. Let F be a chain FST that represents y, with edges $\{e_1, \ldots, e_{|y|}\}$, where $i(e_k) = o(e_k) = y_k$ for all $k \in \{1, \ldots, |y|\}$. Let G be the search space consisting of all paths in $\mathcal{P}$. The term $\log Z(x, y)$ can be efficiently computed by running Algorithm 5 on the σ-composition of G and F, i.e., $G \circ_\sigma F$. Note that after σ-composing G and F, we only allow paths in G that can produce output sequences that F accepts. Since F only accepts y, the paths in $G \circ_\sigma F$ are all the paths in G that produce y as desired. Let the forward and backward marginals computed on $G \circ_\sigma F$ be α' and β'. We have $\log Z(x, y) = \alpha'(v_T) = \beta'(v_0)$.

The gradient of the marginal log loss with respect to $w(e)$ is

$$\frac{\partial \mathcal{L}_{\text{mll}}(\Theta; x, y)}{\partial w(e)} = -\frac{1}{Z(x, y)} \sum_{\substack{p' \ni e \\ \Gamma(p') = y}} \exp(w(p')) + \frac{1}{Z(x)} \sum_{p' \ni e} \exp(w(p')) \tag{3.49}$$

$$= -\exp\Big[\alpha'(\text{tail}(e)) + w(e) + \beta'(\text{head}(e)) - \log Z(x, y)\Big] + \exp\Big[\alpha(\text{tail}(e)) + w(e) + \beta(\text{head}(e)) - \log Z(x)\Big]. \tag{3.50}$$

and can be efficiently computed once all of the marginals are computed.

3.4.4 Empirical Bayes risk

The empirical Bayes risk (EBR) (Smith and Eisner, 2006) is defined as

$$\mathcal{L}_{\text{ebr}}(\Theta; x, p) = \mathbb{E}_{p' \sim P(p'|x)}[\text{cost}(p', p)]. \tag{3.51}$$

The intuition behind EBR is that if P is a good approximation for $\mathcal{D}'$, then EBR is a good approximation for

$$\mathbb{E}_{p' \sim \mathcal{D}'(p'|x)}[\text{cost}(p', p)].$$

The goal is to find P that achieves a low expected cost. Empirical Bayes risk is also referred to as minimum phone error (MPE) when the cost is based on phone errors, and is referred to as minimum word error (MWE) when the cost is based on word errors respectively (Povey and Woodland, 2002). A boosted version of EBR can be obtained by replacing P with P^+ (McDermott and Nakamura, 2008).

Since $Z(x)$ can be computed through the forward and backward marginals α and β, we are left with $\sum_{p'} \exp(w(p'))\text{cost}(p', p)$. Similar to the forward and backward marginals, we have

$$\alpha''(v) = \log \sum_{p' \in \mathcal{P}(v_0, v)} \exp(w(p') + \log \text{cost}(p', p)) \tag{3.52}$$

$$= \log \sum_{e' \in \text{in}(v)} \exp(w(e') + \log \text{cost}(e', p)) \sum_{p' \in \mathcal{P}(v_0, \text{tail}(e))} \exp(w(p') + \log \text{cost}(p', p)) \tag{3.53}$$

$$= \log \sum_{e' \in \text{in}(v)} \exp(w(e') + \log \text{cost}(e', p) + \alpha''(\text{tail}(e))) \tag{3.54}$$

$$\beta''(v) = \log \sum_{p' \in \mathcal{P}(v, v_T)} \exp(w(p') + \log \text{cost}(p', p)) \tag{3.55}$$

$$= \log \sum_{e' \in \text{out}(v)} \exp(w(e') + \log \text{cost}(e', p)) \sum_{p' \in \mathcal{P}(\text{head}(e), v_T)} \exp(w(p') + \log \text{cost}(p', p)) \tag{3.56}$$

$$= \log \sum_{e' \in \text{out}(v)} \exp(w(e') + \log \text{cost}(e', p) + \beta''(\text{head}(e))) \tag{3.57}$$

where cost is assumed to be non-negative, and the marginals are stored in log space to maintain numerical stability. Given the cost-augmented marginals, we have

$$\mathbb{E}_{p' \sim P(p'|x)}[\text{cost}(p', p)] = \exp(\alpha''(v_T) - \log Z(x)) = \exp(\beta''(v_0) - \log Z(x)). \tag{3.58}$$

The gradient of EBR with respect to $w(e)$ of some edge e is

$$\frac{\mathcal{L}_{\text{ebr}}(\Theta; x, p)}{w(e)} = \frac{\sum_{p' \ni e} \exp(w(p'))\text{cost}(p', p)}{Z(x)} - \frac{\left[\sum_{p'} \exp(w(p'))\text{cost}(p', p)\right]\left[\sum_{p' \ni e} \exp(w(p'))\right]}{(Z(x))^2} \tag{3.59}$$

$$= \exp(\alpha''(\text{tail}(e)) + w(e) + \log \text{cost}(e, p) + \beta''(\text{head}(e)) - \log Z(x)) - \mathcal{L}_{\text{ebr}}(\Theta; x, p)[\exp(\alpha(\text{tail}(e)) + w(e) + \beta(\text{head}(e)) - \log Z(x))] \tag{3.60}$$

It is also interesting to note that, in general,

$$\frac{\partial}{\partial w(e)}\mathbb{E}_{p'\sim P(p'|x)}[f(p')] = \mathbb{E}_{p'\sim P(p'|x)}[\mathbb{1}_{p'\ni e}f(p')] - \mathbb{E}_{p'\sim P(p'|x)}[\mathbb{1}_{p'\ni e}]\mathbb{E}_{p'\sim P(p'|x)}[f(p')] \quad (3.61)$$

$$= \mathrm{Cov}(\mathbb{1}_{p'\ni e}, f(p')), \quad (3.62)$$

for any function $f : \mathcal{P} \to \mathbb{R}$.

3.4.5 Ramp loss

The ramp loss (Collobert et al., 2006) is defined as

$$\mathcal{L}_{\text{ramp}}(\Theta; x, p) = \max_{p'}[\text{cost}(p', p) + w(p') - \max_{p''} w(p'')]. \quad (3.63)$$

It is easy to see that

$$\mathcal{L}_{\text{hinge}}(\Theta; x, p) = \max_{p'}[\text{cost}(p', p) + w(p') - w(p)] \quad (3.64)$$

$$\geq \max_{p'}[\text{cost}(p', p) + w(p') - \max_{p''} w(p'')] = \mathcal{L}_{\text{ramp}}(\Theta; x, p). \quad (3.65)$$

In addition, for $\hat{p} = \text{argmax}_{p'} w(p')$,

$$\text{cost}(\hat{p}, p) = \text{cost}(\hat{p}, p) + w(\hat{p}) - w(\hat{p}) \quad (3.66)$$

$$\leq \max_{p'}[\text{cost}(p', p) + w(p')] - w(\hat{p}) \quad (3.67)$$

$$= \max_{p'}[\text{cost}(p', p) + w(p')] - \max_{p'} w(p') = \mathcal{L}_{\text{ramp}}(\Theta; x, p). \quad (3.68)$$

Therefore, we have

$$\mathcal{L}_{\text{hinge}}(\Theta; x, p) \geq \mathcal{L}_{\text{ramp}}(\Theta; x, p) \geq \text{cost}(\hat{p}, p). \quad (3.69)$$

In other words, ramp loss is also an upper bound on the cost of the predicted sequence, but is tighter than hinge loss. However, optimizing the ramp loss is more complicated. Ramp loss can be minimized with the concave-convex procedure (CCCP) (Yuille and Rangarajan, 2003). See (Gimpel and Smith, 2012) for a detailed implementation of CCCP for solving ramp loss.

3.4.6 Connections between losses

Log loss augmented with a cost function, or boosted MMI, can be seen as a soft version of hinge loss. By the definition of boosted MMI, where

$$\mathcal{L}_{\text{bMMI}}(\Theta; x, p) = -\log P(p|x) = -w(p) + \log \sum_{p'} \exp(w(p') + \text{cost}(p', p)), \quad (3.70)$$

we can see that the second term acts as a soft-max instead of a max function. To see this, note that

$$\log(e^{x_1} + e^{x_2} + \cdots + e^{x_n}) = \log e^{x_{\max}}(e^{x_1 - x_{\max}} + \cdots + e^{x_n - x_{\max}}) \tag{3.71}$$

$$= x_{\max} + \log(e^{x_1 - x_{\max}} + \cdots + e^{x_n - x_{\max}}), \tag{3.72}$$

where $x_{\max} = \max_i x_i$. The second term in (3.72) is small when the difference between $x_{\max}$ and other x_i's is large. In fact, we can introduce an addition factor ϵ to control this effect. Specifically,

$$\epsilon \log(e^{x_1/\epsilon} + e^{x_2/\epsilon} + \cdots + e^{x_n\epsilon}) = x_{\max} + \epsilon \log(e^{(x_1 - x_{\max})/\epsilon} + \cdots + e^{(x_n - x_{\max})/\epsilon}). \tag{3.73}$$

When $\epsilon \to 0$, the second term vanishes and (3.73) acts exactly like a max function. We can add the additional ϵ factor to the cost-augmented log loss and get

$$\mathcal{L}^{\epsilon}_{\text{bMMI}}(\Theta; x, p) = -\log P(p|x) = -w(p) + \epsilon \log \sum_{p'} \exp\left(\frac{1}{\epsilon}(w(p') + \text{cost}(p', p))\right). \tag{3.74}$$

As $\epsilon \to 0$, $\mathcal{L}^{\epsilon}_{\text{bMMI}} \to \mathcal{L}_{\text{hinge}}$. This connection was first presented in (Heigold et al., 2008). Gimpel and Smith (2010) independently proposed softmax-margin, the equivalence of cost-augmented log loss.

As we have noted when we introduced log loss,

$$P(p|x) = \mathbb{E}_{p' \sim P(p'|x)}[\mathbb{1}_{p'=p}] = 1 - \mathbb{E}_{p' \sim P(p'|x)}[\mathbb{1}_{p' \neq p}]. \tag{3.75}$$

Therefore, minimizing log loss is equivalent to minimizing EBR with $\text{cost}(p', p) = \mathbb{1}_{p' \neq p}$.

Another connection between cost-augmented log loss and EBR is the following. Let

$$\mathcal{L}_{\lambda,\text{bMMI}}(\Theta; x, p) = -\log P(p|x) = -w(p) + \log \sum_{p'} \exp(w(p') + \lambda \text{cost}(p', p)), \tag{3.76}$$

where we introduce a scaling factor λ for the cost function in cost-augmented log loss. If we take the derivative of cost-augmented log loss with respect to λ, we have

$$\frac{\partial}{\partial \lambda} \mathcal{L}_{\lambda,\text{bMMI}} = \frac{\sum_{p'} [\exp(w(p') + \lambda \text{cost}(p', p)) \text{cost}(p', p)]}{\sum_{p'} \exp(w(p') + \lambda \text{cost}(p', p))} \tag{3.77}$$

$$= \mathbb{E}_{p' \sim P^{+}_{\lambda}(p'|x)}[\text{cost}(p', p)], \tag{3.78}$$

where

$$P^{+}_{\lambda}(p'|x) = \frac{\exp(w(p') + \lambda \text{cost}(p', p))}{\sum_{p''} \exp(w(p'') + \lambda \text{cost}(p'', p))} \tag{3.79}$$

Therefore, the derivative of cost-augmented log loss with respect to λ is cost-augmented EBR, or sometimes referred to as boosted MPE. This connection was first presented in (McDermott et al., 2009).

3.5 Preliminary Experiments

In this section[2], we describe experiments comparing segmental models trained with various loss functions and two cost functions. Instead of using the entire search space, we follow (Zweig and Nguyen, 2009; Zweig et al., 2010) and use a baseline recognizer to generate sparse search spaces, commonly known as **lattices**. Segmental models are then applied on these lattices. The weight function used in these experiments is a linear function $w(x, e) = \theta^\top \phi(x, e)$ where θ is a parameter vector and $\phi(x, e)$ is a feature vector for some edge e. The feature function ϕ is task-dependent, and is described in detail later. Hinge loss and ramp loss are sensitive to the scale of the cost, so the scale of the cost function is tuned. Specifically, we introduce the parameter μ in the hinge loss

$$\mathcal{L}_{\text{hinge}} = \max_{p'}[\mu \cdot \text{cost}(p', p) - w(p) + w(p')], \tag{3.80}$$

and parameters μ_1 and μ_2 for the ramp loss

$$\mathcal{L}_{\text{ramp}} = \max_{p'}[\mu_1 \cdot \text{cost}(p', p) + w(p')] - \max_{p'}[\mu_2 \cdot \text{cost}(p', p) + w(p')]. \tag{3.81}$$

Note the additional cost term in the above ramp loss. This version of ramp loss is a generalization of the standard ramp loss (Chiang, 2012; Gimpel and Smith, 2012).

3.5.1 Cost functions

One cost function was proposed in (Povey and Woodland, 2002) in the context of MPE/MWE training, and we refer to it as **MPE cost**. The cost of an edge is the duration of the non-overlapping part of a matching ground-truth edge that gives the lowest error, where the error is one if the label is correct and 0.5 otherwise. Formally, for any hypothesized edge e', we define

$$\text{cost}_{\text{MPE}}(e', p) = 1 - \max_{e \in p} \left[\mathbb{1}_{o(e)=o(e')} \frac{|e \cap e'|}{|e|} + \frac{1}{2} \mathbb{1}_{o(e)\neq o(e')} \frac{|e \cap e'|}{|e|} \right], \tag{3.82}$$

and

$$\text{cost}_{\text{MPE}}(p', p) = \sum_{e' \in p'} \text{cost}_{\text{MPE}}(e', p), \tag{3.83}$$

where $|e|$ denotes the length of segment e.

The MPE cost only penalizes false negatives and does not account for false positives. Therefore, we propose an alternative that we refer to as the **overlap cost**:

$$\text{cost}_{\text{overlap}}(e', p) = 1 - \mathbb{1}_{o(e)=o(\tilde{e})} \frac{|e \cap \tilde{e}|}{|e \cup \tilde{e}|}, \tag{3.84}$$

[2]These results were published in (Tang et al., 2014)

Table 3.1: Letter error rates (%) for a baseline tandem HMM and segmental models trained with various losses and cost functions on the ASL data set.

			Andy	Drucie	Rita	Robin	Avg
Tandem HMM			13.8	7.10	26.1	11.5	14.6
segmental models		log loss	**9.9**	**6.7**	23.9	**10.5**	12.8
	MPE cost	hinge loss	10.5	**6.7**	23.5	10.9	12.9
		EBR	13.8	7.2	27.6	12.6	15.3
		ramp	13.8	7.2	28.6	12.7	15.6
	overlap cost	hinge loss	10.3	6.9	21.9	**10.5**	12.4
		EBR	10.4	6.8	23.5	11.6	13.1
		ramp	12.0	6.8	25.2	11.7	13.9
		tuned ramp	10.1	6.8	**21.5**	10.6	**12.3**

where $\tilde{e} = \text{argmax}_{e \in p} |e' \cap e|$. This cost function finds the most overlapping edge in the ground truth and considers any part of the union of the two edges that is not overlapping to be in error. The cost for the whole path is again

$$\text{cost}_{\text{overlap}}(p', p) = \sum_{e' \in p'} \text{cost}_{\text{overlap}}(e', p). \tag{3.85}$$

3.5.2 Results

We study the various losses and cost functions on two tasks. One is a standard speech recognition task, namely TIMIT (Garofolo et al., 1993) phonetic recognition. The second is a sign language recognition task from video, in particular recognition of fingerspelled letter sequences in American Sign Language (ASL). Both are tasks on which there is prior work using semi-Markov CRFs (Zweig, 2012; Kim et al., 2013), and both are small enough (in terms of data set size and decoding search space) to run many empirical comparisons in a reasonable amount of time. For the ASL task, we use the data and experimental setup of (Kim et al., 2013): We obtain baseline lattices using their tandem HMM-based system, and we use the same set of segmental feature functions. We use forced alignments of the ground-truth transcriptions for training. We train all models from all-zero weights and optimize with Rprop (Riedmiller and Braun, 1993) for 20 epochs. We use L_1 and L_2 regularization, with parameters tuned over the grid $\{0, 10^{-6}, 10^{-5}, \ldots, 0.1, 1\}^2$. For hinge and ramp loss, we use the standard forms without tuning the cost weights (i.e., $\mu = 1$, $\mu_1 = 0$, and $\mu_2 = 1$).

For TIMIT, we use the standard 3696-utterance training set and 192-utterance core test set, plus a random 192 utterances from the full test set (excluding the core test set) as a development set. We collapse the 61 phones in the phone set to 48 for training, and further collapse them to 39 for evaluation (Lee and Hon, 1989).

We use lattices generated by a baseline monophone HMM system with 39-dimensional MFCCs. The resulting lattices have an average density (average number of hypothesized

Table 3.2: Phone error rates (%) for a baseline HMM and segmental models trained with various losses and cost functions on the TIMIT data set.

			test
Tandem HMM			30.7
segmental models		log loss	**30.1**
	MPE cost	hinge loss	30.2
		EBR	30.7
		ramp	30.2
	overlap cost	hinge loss	**30.1**
		EBR	**30.1**
		ramp	30.3

edges per ground truth edge) of 60.1. The oracle phone error rate is 6.3% for the development set and 7.0% for the core test set. We use oracle paths (paths with minimum phone error) from the lattices as ground truth for training. We implement segmental models with various feature functions. The base features are the acoustic and language model score from the baseline recognizer, and a bias (a feature that is always one). We also include a set of features based on spectro-temporal receptive fields implemented as follows. We begin with 40-dimensional log mel filter bank features. For each segment, we divide it evenly into thirds in both time and frequency, resulting in nine patches for each segment. For each patch, we have a 3×13 receptive field of all ones, and convolve it with the patch. The resulting $3 \times 13 \times 9$ numbers are lexicalized to form the final features for the segmental model. Specifically, for any feature vector v, we refer to $v \otimes \mathbf{1}_\ell$ as the lexicalized feature vector, where $\mathbf{1}_\ell$ is a one-hot vector for the label ℓ. We optimize the loss functions using AdaGrad (Duchi et al., 2011), using step size 0.1 for 10 epochs. L_1 and L_2 regularization parameters are tuned over the grid $\{0, 0.001, 0.1, 1\} \times \{0, 0.1, 1, 100\}$.

The results for ASL recognition, averaged over four signers, are shown in Table 3.1. The evaluation metric is the letter error rate, which is the percentage of letters that are substituted, inserted, or deleted. The results for TIMIT are shown in Table 3.2. We observe three consistent conclusions:

- Segmental models perform significantly better than the baseline HMM.
- Across losses, overlap cost is better than MPE cost.
- Hinge loss with overlap cost is the best performer, but this is only by a small margin, and log loss is very competitive even without using an explicit cost function.
- Non-convex losses (ramp and EBR) are difficult to optimize and therefore achieve inconsistent results. We suspect a warm start might be able to remedy this.

For the ASL task, we tuned on a development set the cost weights for ramp loss over the grid $\{-100, -10, -1, -0.1, -0.01\} \times \{0.01, 0.1, 1, 10, 100\}$, using overlap cost. The test result

of tuned ramp slightly improves over hinge loss, confirming that if ramp is tuned carefully, it is able to outperform hinge. However, even though tuned ramp loss achieves very good results, considering the time spent tuning μ_1 and μ_2, we still favor hinge and log loss.

We also conducted experiments to determine how the results are affected by different levels of noise in the feature functions, using simulated phone detector-based features. Similarly to (Zweig et al., 2010), we define a detection event as a (time, phone label) pair, and a feature function that is an indicator of whether a phone detection event occurs in the time span of the edge. If we set a high weight for the phone event that occurs in an edge with the same phone label, then we can exactly recover the oracle path. This allows us to conduct a series of simulated experiments with different amounts of noise added to the oracle phone events, or gold events. For all experiments below, we use the same TIMIT setting except that we only use the acoustic and language model score with the simulated phone detector features, with no regularizer and one epoch of AdaGrad. The ramp cost weights are set to $\mu_1 = 0, \mu_2 = 1$. The cost weight for hinge is tuned over $\{0.01, 0.1, 1, 10, 100\}$ and results are only shown for the best-performing value.

The first set of experiments randomly perturbs the correct phone label of each event to an incorrect label with the corruption probability shown on the x-axis; the event times are not perturbed. The second set of experiments perturbs the time for each event but not the label. We add Gaussian noise with mean set to the time at which the event occurs and with several standard deviations shown on the x-axis. For the third and fourth set of experiments, we randomly include an edge in the lattice as a false positive event, or randomly delete an event from the gold events.

The conclusions are consistent with our previous observation, namely, that hinge is the consistent winner but only by a very small margin, that log loss is very competitive, that non-convex losses are hard to optimize, and that overlap cost is better than MPE cost. As a byproduct, we note that we could achieve 17.7% given a phone detector with any of the following characteristics: up to 50% phone error rate but perfect time information, up to 5-frame time perturbations (in standard deviation) but perfect labels, 1.8 false positives per gold edge, or 20% false negatives.

3.6 Summary

In this chapter, we have described a formal framework for discriminative segmental models. The definition is not tied to any weight function or any loss function, encompassing a large family of segmental models. We have described two weight functions, one based on frame classifiers, and one based on segmental recurrent neural networks. We have also described various loss functions, and have drawn connections between surrogate losses and the true loss that we want to minimize. We have shown how (sub-)gradients of loss functions can be efficiently computed with FST algorithms. The flexibility to use different loss functions allows us to train segmental models in different training settings. Preliminary results have shown that segmental models outperform HMMs by a large margin and that overlap cost is better than MPE cost across loss functions. In terms of loss functions, non-convex losses are

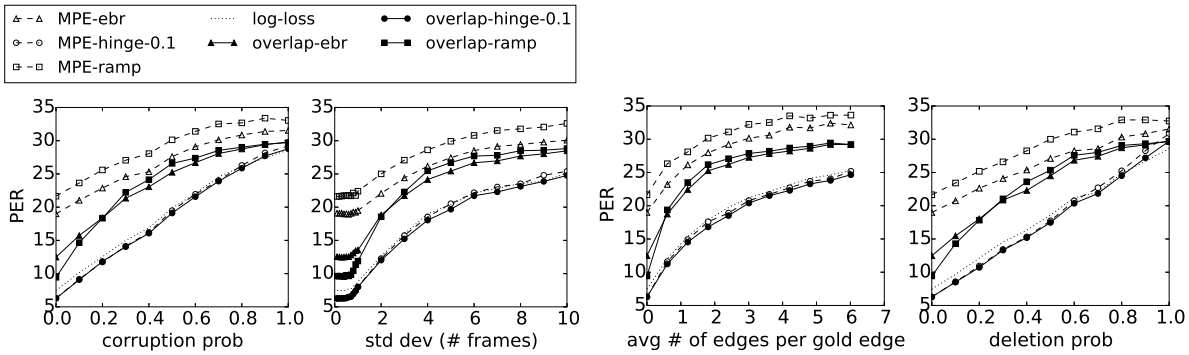


Figure 3.2: Lattice rescoring with various noise added to a perfect phone detector. *From Left to Right*: Perturbing phone labels. Perturbing time. Adding false positive events. Adding false negative events.

more difficult to optimize. Hinge loss is the best performer while log loss is very competitive.

Our final task loss is the edit distance, but none of the surrogate losses can be optimized when the cost function is the edit distance. There are two difficulties for optimizing the edit distance: the edit distance is discrete, and it involves a minimization over all alignments to the ground truth. Using a cost function that considers all alignments to the ground truth without the minimization has been explored in (Heigold et al., 2005; van Dalen and Gales, 2015; Shannon, 2017). Other approaches for directly minimizing task losses (Keshet and McAllester, 2011; Karmon and Keshet, 2015) are also suitable for minimizing the edit distance.

Chapter 4

Discriminative Segmental Cascades

As we have seen in Chapter 3, search spaces defined by segmental models are dense and redundant, in the sense that many segments that have the same label differ only by a few frames (or even just a single frame) and that many segments with the same start time and end time differ only in the labels. In other words, most of the random segments do not look like the ground truths.

Decoding in multiple passes is an approach that exploits dense search spaces. Since the search spaces are dense, a simple weight function is typically enough to significantly reduce the size of search spaces. After the first pass, we may use more complicated weight functions to make predictions or to further reduce the search spaces. Reducing the size of search spaces is commonly known as **pruning**.

Pruning has to be done carefully, because it might hurt the final performance if the ground truth segments are pruned. To measure how pruning affects the search spaces and the quality of predictions, we measure the lowest achievable task loss, the **oracle error rate**, of the search spaces after pruning. A simple experiment on the phonetic recognition task shows that the oracle error rate, i.e., the best edit distance we can achieve, on average stays at zero even if we remove 50% of segments at random. This result shows that the search spaces are indeed dense and redundant.

Since the pruned search space is smaller than the original space, inference and learning using the pruned search space are faster in the second pass. As a result, we can afford to use computationally expensive weight functions for better prediction in the second pass. In general, with the same weight function, multi-pass systems have the potential to decode faster than single-pass systems. In other words, under the same decoding time budget, multi-pass systems can use computationally more expensive weight functions than single-pass systems to obtain better performance.

Since pruning plays an important role, in the following sections, we present and compare several pruning strategies and their consequences. We then develop a multi-pass system, trading between the size of the search space and the computational complexity of weight functions.

4.1 Pruning

In this section, we describe several pruning approaches to reduce search spaces. Note that pruning is a form of inference, and we assume a pruning approach has access to search spaces defined by an FST, and a weight function $w : E \to \mathbb{R}$ parameterized by Θ_{prn}. We discuss the choice of w and how Θ_{prn} is obtained in the experimental sections.

4.1.1 Greedy pruning

A very naive approach, referred to as greedy pruning, is to prune the edges branching out from a vertex based on their edge weights. Specifically, the set of edges branching out from a vertex v is pruned based on a threshold τ. Let

$$S_v = \{e \in \text{out}(v) : w(e) \geq \tau\} \tag{4.1}$$

be the set of vertices that survive pruning. We collect the edges $\bigcup_{v \in V} S_v$ to form the pruned graph. There are two nice properties about this approach. First, the pruning procedure is embarrassingly parallelizable. Second, if we let

$$\tau = \lambda \min_{e \in \text{out}(v)} w(e) + (1 - \lambda) \max_{e \in \text{out}(v)} w(e) \tag{4.2}$$

where $0 \leq \lambda \leq 1$ is a parameter that controls the amount of pruning, the graph is guaranteed to be connected. To see this, note that there is at least one edge surviving for every vertex. In other words, from the start vertex, there is at least one edge for us to traverse for every vertex, and we eventually arrive at the final vertex.

4.1.2 Beam pruning

Beam pruning, or more generally beam search (Lowerre, 1976), is a widely used search and pruning method. The motivation behind beam search is to constrain a search algorithm, such as the shortest-path algorithm (Algorithm 2), with a fixed memory budget. Due to the memory budget, we cannot afford to remember all the paths we have traversed, so we prune the paths that are less likely to have high weights.

Specifically, the beam search algorithm performs the following steps while visiting vertices in topological order. We keep an approximate shortest distance $\hat{d}(u)$ to every vertex u. Suppose vertex v is the current vertex being visited. A set of surviving edges

$$S_v = \{e \in \text{in}(v) : \hat{d}(\text{tail}(e)) \geq \tau\} \tag{4.3}$$

is computed based on a threshold τ, and then the approximate shortest distance for v is updated by setting

$$\hat{d}(v) = \max_{e \in S_v} w(e) + \hat{d}(\text{tail}(e)). \tag{4.4}$$

Note that this equation is identical to the shortest-path recursion, except that the set of incoming edges $\text{in}(v)$ is pruned before updating. To reconstruct the graph after beam pruning, we simply collect the surviving edges $\bigcup_{v \in V} S_v$. We can also let

$$\tau = \lambda \min_{e \in \text{in}(v)} \hat{d}(\text{tail}(e)) + (1 - \lambda) \max_{e \in \text{in}(v)} \hat{d}(\text{tail}(e)) \tag{4.5}$$

where $0 \leq \lambda \leq 1$ is a parameter that controls the amount of pruning.

Note that beam pruning is very similar to greedy pruning. For greedy pruning, edges are pruned based on edge weights, while for beam pruning, edges are pruned based on estimates of shortest distances.

4.1.3 Max-marginal pruning

It is obvious that if an edge is pruned, paths going through the edge are no longer in the search space. Whenever we prune an edge, we discard the maximum-weight path that passes through the edge. Obviously if the maximum-weight path passing through the edge survives pruning, the edge survives pruning. The **max-marginal** for an edge is defined as the weight of the maximum-weight path passing through the edge.

Max-marginal pruning was first proposed by Sixtus and Ortmanns (1999) and later rediscovered by Weiss et al. (2012). Formally, the max-marginal of an edge e is defined as

$$\gamma(e) = \max_{p \ni e} w(p), \tag{4.6}$$

i.e., the weight of the maximum-weight path that passes through e. The pruning procedure is simple: choose a threshold τ and discard edge e if $\gamma(e) < \tau$. One way to set the threshold (Weiss et al., 2012) is

$$\tau_\lambda = \lambda \max_{e \in E} \gamma(e) + (1 - \lambda) \frac{1}{|E|} \sum_{e \in E} \gamma(e), \tag{4.7}$$

where E is the set of segments and $0 \leq \lambda \leq 1$. There are two nice properties about max-marginal pruning with the above threshold: There is at least one path surviving; all paths with weights higher than τ_λ survive the pruning. The first property is true because paths that achieve the maximum weight always survive. The second property can be proved by contradiction. Suppose path p has weight $s > \tau_\lambda$ but is pruned. Then there exists an edge e in p such that $\gamma(e) < \tau_\lambda$. On the other hand, since p passes through e, $\gamma(e)$ is at least s, i.e., $\gamma(e) > s > \tau_\lambda$, a contradiction. Note that we can only guarantee that paths with weights larger than τ_λ survive pruning. It does not imply that all paths that survive pruning have weights larger than τ_λ.

To calculate $\gamma(e)$ for each $e \in E$, note that

$$\gamma(e) = \max_{p \ni e} w(p) = \Big[\max_{p_1 \in \mathcal{P}(s, \text{tail}(e))} w(p_1) \Big] + w(e) + \Big[\max_{p_2 \in \mathcal{P}(\text{head}(e), t)} w(p_2) \Big], \tag{4.8}$$

where $\mathcal{P}(u, v)$ is the set of paths that start at vertex u and end at vertex v. We can construct an algorithm like the shortest-path algorithm to calculate the first and third terms as follows.

$$d_1(v) = \max_{p_1 \in \mathcal{P}(s,v)} w(p_1) = \max_{e \in \text{in}(v)} [d_1(\text{tail}(e)) + w(e)] \tag{4.9}$$

$$d_2(v) = \max_{p_2 \in \mathcal{P}(v,t)} w(p_2) = \max_{e \in \text{out}(v)} [d_2(\text{head}(e)) + w(e)] \tag{4.10}$$

As shown above, the runtime of max-marginal pruning is the same as that of the shortest-path algorithm (Algorithm 2).

Max-marginal pruning can also be applied to vertices. Let $\gamma(v) = \max_{p \ni v} w(p)$ be the max-marginal of the vertex v. We can also obtain a similar recursion

$$\gamma(v) = \max_{p \ni v} w(p) = \Big[\max_{p_1 \in \mathcal{P}(s,v)} w(p_1)\Big] + \Big[\max_{p_2 \in \mathcal{P}(v,t)} w(p_2)\Big] = d_1(v) + d_2(v), \tag{4.11}$$

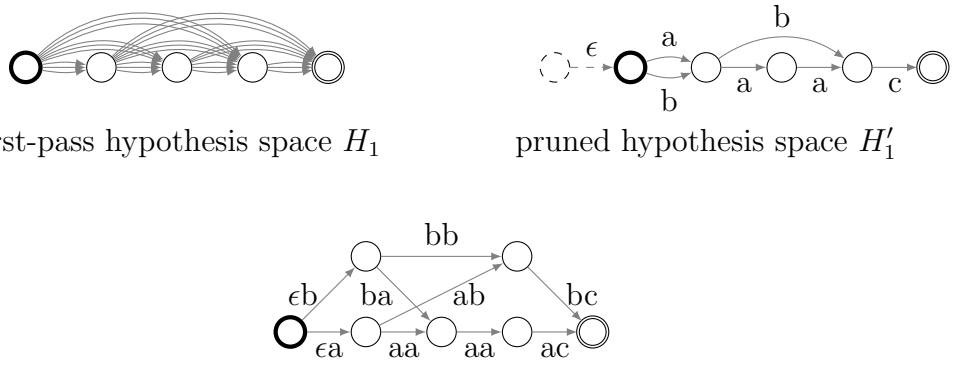
and set a threshold based on the convex combination of $\max_{v \in V} \gamma(v)$ and $\frac{1}{|V|} \sum_{v \in V} \gamma(v)$. Similar guarantees also hold for max-marginal vertex pruning.

4.2 Discriminative Segmental Cascades

Recall that the runtime for finding the shortest path is $O(|E|C)$ where E is the set of segments in the search space and C is the computational cost of evaluating the weight function on a single segment. Suppose we have a fixed time budget. Since the pruned search space is smaller than the original space, we can afford to use a more computationally expensive weight function on the pruned space for better prediction. Since additional pruning can be applied to the search space, we end up with a system having multiple rounds of pruning with increasingly expensive weight functions. Prediction is then done on the pruned space from the final round.

The above approach is also commonly referred to as **rescoring**, because hypotheses are given new scores (weights) after pruning. Models are also named after the pass in which they are used. For example, a first-pass model is one that searches over the entire search space; a second-pass model is a model that rescores the pruned search space produced by a first-pass model. Based on rescoring, Weiss et al. (2012) proposed structured prediction cascades, a type of multi-pass system with multiple rounds of pruning. The complete search space is denoted H_1. At round k, we have hypothesis space H_k, train a pruning model Θ_k on H_k, and prune H_k to get H'_k. The pruned space H'_k is expanded to H_{k+1} by considering more context for every segment, such as label n-grams. The process can be repeated for any number of rounds.

Inspired by (Weiss et al., 2012), we propose **discriminative segmental cascades**, a multi-pass system consisting of segmental models with increasingly expensive weight functions. In our cascades, all search spaces are represented as FSTs. We denote the complete search space H_1. At the k-th pass, based on the search space H_k, we train a segmental model Θ_k, and use it to prune the search space to get H'_k. In our framework, we can decide whether



first-pass hypothesis space H_1

pruned hypothesis space H'_1

second-pass hypothesis space $H_2 = H_1'^{\uparrow}$

Figure 4.1: An example segmental cascade. The search space H_1 is the complete space. The search space H'_1 is pruned from H_1. The search space H_2 is the self-expansion of H'_1. The additional vertex and edge added to H'_1 before self-expansion are shown in dashed lines.

to expand the search space to include more context. Since the search spaces are represented as FSTs, we propose a new FST operation, termed **self-expansion**, to efficiently consider neighboring segments for every segment. If we decide to self-expand, then $H_{k+1} = H_k'^{\uparrow}$, where $H_k'^{\uparrow}$ is the self-expansion of H'_k. Otherwise, $H_{k+1} = H'_k$. Suppose we decide to self-expand. Then the weight function for the subsequent pass w_{k+1} can make use of a wider context than the weight function of the previous pass. Including a wider context is one possible approach to increase the expressiveness of the weight function. Other approaches include using a deeper or more complex neural network, or extracting more sophisticated features such as tracking formants or pitches, both of which can be done without self-expansion. The process can be repeated many times, forming a segmental cascade. An example of a segmental cascade is shown in Figure 4.1.

Segmental models in each of these passes can be trained with the losses described in Section 3.4. Weiss et al. (2012) proposed to train the pruning models with the filtering loss, measuring how well the ground truths are retained after pruning. We decide to train the segmental models in the cascade with losses in Section 3.4 that directly measure the final performance, making sure that we have a model for prediction after each pass. This approach also makes it easier to monitor when to stop stacking the cascade.

4.2.1 Self-expansion

Given a search space represented as an FST H, we can expand the context of a segment by considering its neighbors. Suppose we want to incorporate the previous segment given the current segment. We create a new FST $H^{\uparrow}$ that remembers and keeps track of the previous segment (edge) traversed in H. Vertices in $H^{\uparrow}$ correspond to the possible previous segments in H. Edges in $H^{\uparrow}$ are of the form $\langle e_1, e_2 \rangle$, where e_1 is the edge just tra-

versed and e_2 is the edge to be traversed. We refer to $H^\uparrow$ as the **self-expansion** of H. Formally, let $H = ((V, E, \text{tail}, \text{head}), \Sigma, \Lambda, I, F, i, o)$. The self-expansion of H is defined as $H^\uparrow = ((V^\uparrow, E^\uparrow, \text{tail}^\uparrow, \text{head}^\uparrow), \Sigma^\uparrow, \Lambda^\uparrow, I^\uparrow, F^\uparrow, i^\uparrow, o^\uparrow)$ where

$$V^\uparrow = E \cup \{e_0\} \qquad E^\uparrow = \{\langle e_1, e_2\rangle : \text{head}(e_1) = \text{tail}(e_2)\} \tag{4.12}$$

$$I^\uparrow = \{e_0\} \qquad \text{tail}^\uparrow(\langle e_1, e_2\rangle) = e_1 \tag{4.13}$$

$$F^\uparrow = \{e \in E : \text{head}(e) \in F\} \qquad \text{head}^\uparrow(\langle e_1, e_2\rangle) = e_2 \tag{4.14}$$

$$\Sigma^\uparrow = \Sigma \qquad i^\uparrow(\langle e_1, e_2\rangle) = i(e_2) \tag{4.15}$$

$$\Lambda^\uparrow = \Lambda \qquad o^\uparrow(\langle e_1, e_2\rangle) = o(e_2) \tag{4.16}$$

A vertex v_0 and an e_0 is created in H before self-expansion, where $i(e_0) = o(e_0) = \epsilon$, $\text{tail}(e_0) = v_0$, and $\text{head}(e_0) = v_1$ assuming we only have a single start state v_1, i.e., $I = \{v_1\}$. The edge e_0 is created as a sentinel, corresponding to a vertex in $H^\uparrow$ for not having traversed any edge.

The weight function $w^\uparrow : E^\uparrow \to \mathbb{R}$ is task-dependent, so we do not define it explicitly here. However, since $w^\uparrow$ is defined on $E^\uparrow$, the weight function after taking a edge $\langle e_1, e_2\rangle \in E^\uparrow$ can make use of all the information about e_1 and e_2 to compute the weight, including the start times, end times, and labels. We also observe that

$$\text{out}^\uparrow(e) = \{\langle e, e'\rangle : e' \in \text{out}(\text{head}(e))\} \tag{4.17}$$

$$\text{in}^\uparrow(e) = \{\langle e', e\rangle : e' \in \text{in}(\text{tail}(e))\} \tag{4.18}$$

for $e \in V^\uparrow = E$. Therefore, similarly to σ-composition, self-expansion can be computed lazily (on the fly) while traversing H.

4.3 Experiments

We conduct experiments on the TIMIT data set in the same setting as in Section 3.5.2, except that here we follow (Graves et al., 2013) and use 40-dimensional log mel filter bank outputs instead of 39-dimensional MFCCs. The data set is phonetically transcribed, so we have the option of training the encoders with frame-wise cross entropy based on the ground-truth frame labels.

4.3.1 A segmental baseline

In the following experiments, we explore convolutional neural network (CNN) encoders. The CNN encoders are inspired by (Simonyan and Zisserman, 2015) in that they are deep convolutional neural networks with small 3×3 and 5×5 filters. For frame classification, the input to the network is a window of 15 frames of 40-dimensional log mel filter outputs centered on the current frame. The network has five convolutional layers, with 64 filters of

size 5×5 for the first layer and 128, 128, 256, and 256 filters of size 3×3 for layers two to five respectively, each of which is followed by a rectified linear unit (ReLU) activation (Nair and Hinton, 2010), with max pooling layers after the first and the third ReLU layers. The output of the final ReLU layer is concatenated with a window of 15 frames of 39-dimensional MFCCs centered on the current frame, and the resulting vector is passed through three fully connected ReLU layers with 4096 units each. The network is trained with SGD for 35 epochs with step size 0.001, momentum 0.95, weight decay 0.005, and a mini-batch size of 100 frame predictions. Dropout (Srivastava et al., 2014) are applied to the concatenation layer with rate 20% and to the fully connected layers with rate 50%. This classifier was tuned on the development set and achieves a 22.3% frame error rate (after collapsing to 39 phone labels) on the development set and 23.0% on the test set.

After confirming that the CNN encoder is able to perform well on frame classification, we use the CNN encoder to generate frame posterior probabilities, and train segmental models with the FC weight function and a maximum duration of 30 frames. Hinge loss is optimized with AdaGrad (Duchi et al., 2011) with step size 0.01 and a mini-batch size of 1 utterance for 70 epochs. No explicit regularizer is used except early stopping on the development set. We use a scaled overlap cost for our experiments:

$$\text{cost}(p', p) = \sum_{e' \in p} \text{cost}(e', p) \tag{4.19}$$

$$\text{cost}(e', p) = |e' \cup \tilde{e}| - |e' \cap \tilde{e}| \mathbb{1}_{o(e')=o(\tilde{e})} \tag{4.20}$$

where p' is the hypothesized path, p is the ground-truth path, $|e' \cup e|$ denotes the union of the two intervals defined by e and e', $|e' \cap e|$ denotes the intersection of the two intervals, $o(e)$ is the output label of e, and $\tilde{e} = \text{argmax}_{e \in p} |e' \cap e|$. In words, $\tilde{e}$ is the edge that overlaps the most with e'. The cost is non-negative and is only zero if $e' = e$, and it can be seen as an estimate of the number of incorrectly predicted frames.

The results are shown in Table 4.1. Our first-pass segmental model is on par with Abdel-Hamid et al. (2013), where a CNN encoder is also used, and it is also on par with a baseline HMM-DNN hybrid system that we trained with Kaldi using the standard recipe for TIMIT (Povey et al., 2011).

4.3.2 Pruning comparison

To construct a second-pass segmental model, we first evaluate pruning approaches based on oracle error rates and lattice densities. The **oracle error rate** is the lowest achievable phone error rate of a given lattice. The **lattice density** is defined as the total number of edges divided by the number of ground-truth edges, i.e., the number of hypothesized edges per ground-truth edge. In general, a dense lattice has a higher chance to contain the ground-truth path than a sparse lattice. In fact, the entire search space is a dense lattice that always contains the ground-truth path. A good pruner is expected to produce sparse lattices while maintaining a low oracle error rate.

We compare max-marginal pruning with $\lambda \in \{0.5, 0.6, 0.7, 0.8\}$, greedy pruning with $\lambda \in \{0.95, 0.9, 0.875, 0.85, 0.825\}$, random pruning with probability $\{0.92, 0.94, 0.96, 0.98\}$,

Table 4.1: Phoneme error rates (%) on TIMIT comparing a HMM-DNN hybrid system to discriminative segmental models.

	dev	test
HMM-DNN		21.4
deep segmental neural network (Abdel-Hamid et al., 2013)		21.8
our segmental model	22.2	21.7

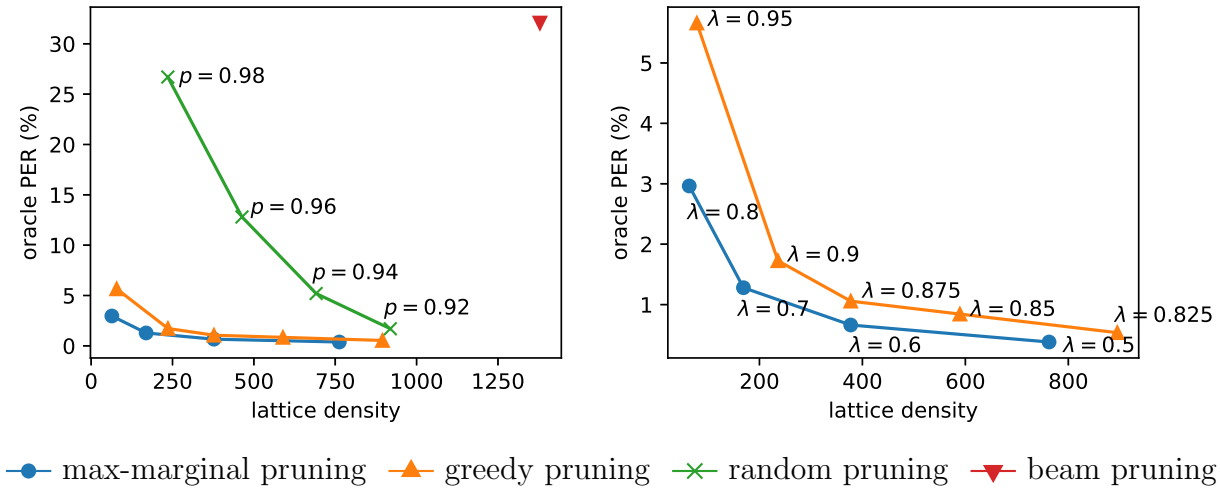


Figure 4.2: *Left*: Oracle error rates vs lattice densities for various pruning approaches. *Right*: A re-scaled version of the left figure focusing on greedy pruning and max-marginal pruning.

and beam pruning with $\lambda = 0.9$. The results are shown in Figure 4.2. For reference, the average density of the complete spaces is 11587.67. We observe that search spaces of segmental models are robust to missing edges—the oracle error rate is only 1.7% when we randomly drop 92% of the edges. We also observe that max-marginal pruning produces significantly sparser lattices than greedy pruning and random pruning while maintaining low oracle error rates. Beam pruning performs the worst. Since greedy pruning works well by just comparing edges with the same tail vertex (i.e., with the same start time), we suspect the bad performance of beam pruning is due to comparing distances at different time points.

4.3.3 Improving prediction

Based on the above pruning comparison, we decide to use the lattices generated by max-marginal pruning with $\lambda = 0.8$. Let H_1 be the entire search space, and H_1' be the pruned

search space. To train a segmental model with a wider segmental context, we first self-expand H_1' into $H_2 = {H_1'}^{\uparrow}$ and introduce the following weight functions. Note that the weight functions can now depend on two segments instead of one.

lattice weight Instead of re-learning all of the weight functions in the first-pass model, we reuse them in the second pass by having

$$w_{\text{lat}}(\langle(\ell_1, s_1, t_1), (\ell_2, s_2, t_2)\rangle) = \alpha \cdot w((\ell_2, s_2, t_2)) \tag{4.21}$$

from the first level of the cascade, where α is a learnable parameter.

bigram LM weight We use the weight

$$w_{\text{LM}}(\langle(\ell_1, s_1, t_1), (\ell_2, s_2, t_2)\rangle) = \beta \cdot \log p(\ell_2|\ell_1) \tag{4.22}$$

to include the log probability of the bigram $\ell_1\ell_2$, where β is a learnable parameter.

second-order boundary weight We define

$$w_{\text{left}-k}(\langle(\ell_1, s_1, t_1), (\ell_2, s_2, t_2)\rangle) = u_{s_2-k,\ell_1,\ell_2} \tag{4.23}$$

$$w_{\text{right}-k}(\langle(\ell_1, s_1, t_1), (\ell_2, s_2, t_2)\rangle) = u'_{t_2+k,\ell_1,\ell_2} \tag{4.24}$$

similarly to the first-order boundary weights, where $u_{t,\ell,\ell'} = \theta_{\ell,\ell'}^{\top} h_t$ and $u'_{t,\ell,\ell'} = {\theta'_{\ell,\ell'}}^{\top} h_t$ for $t = 1, \ldots, \tilde{T}$ and some learnable parameters θ, θ'.

Since the search space is sparse, we can afford to compute segment features based on a neural network for every segment. We choose a similar CNN architecture to the one for frame classification. We pre-train the CNN with a segment classification task. Here the features at the input layer are the log mel filter outputs from a 15-frame window around the segment's central frame. Instead of concatenation with 15-frame MFCCs, we concatenate with a segmental feature vector consisting of the average MFCCs of three sub-segments in the ratio of 3-4-3, plus two four-frame averages at both boundaries and length indicators for length 0 to 20 (similar to the segmental feature vectors of (Halberstadt and Glass, 1998; Clarkson and Moreno, 1999)). This CNN is trained on the ground-truth segments in the training set. Finally, we build an ensemble of such networks with different random seeds and a majority vote. This ensemble classifier has a 15.0% segment classification error on the test set. A comparison of our CNN to other segment classifiers is shown in Table 4.2.

Directly running the CNN segment classifier for every edge in the lattice is, however, still too time-consuming. We instead compress the best-performing (single) CNN into a shallow fully connected network with one hidden layer of 512 ReLUs by training it to predict the log probability outputs of the deep network, as proposed by Ba and Caruana (2014). We then use the log probability outputs of the shallow network. We refer to the result as segment NN weights.

Table 4.2: TIMIT segment classification error rates (%).

	test
Gaussian mixture model (GMM) (Clarkson and Moreno, 1999)	26.3
SVM (Clarkson and Moreno, 1999)	22.4
Hierarchical GMM (Halberstadt, 1998)	21.0
Discriminative hierarchical GMM (Chang and Glass, 2007)	16.8
SVM with deep scattering spectrum (Andén and Mallat, 2014)	15.9
our CNN ensemble (Tang et al., 2015)	15.0

segment NN weight Let

$$w_{\text{seg}}(\langle(\ell_1, s_1, t_1), (\ell_2, s_2, t_2)\rangle) = (u_{s_2,t_2})_{\ell_2} \tag{4.25}$$

where $u_{s,t} = \theta^\top z_{s,t}$, $z_{s,t}$ is a log probability vector of the labels obtained by passing the segment $x_s, \ldots, x_t$ to the CNN segment classifier, and θ is some learnable parameter vector.

To add more context information, we use the same CNN architecture and training setup to learn a bi-phone frame classifier, but with an added 256-unit bottleneck linear layer before the softmax. Each frame is labeled with its segment label and one additional label from a neighboring segment. If the current frame is in the first half of the segment, the additional label is the previous phone; if it is in the second half, then the additional label is the next phone. The learned bottleneck layer outputs are used to define weights (although they do not correspond to log probabilities) with averaging and sampling as for the uni-phone case. We refer to the resulting weights as bi-phone NN bottleneck weights.

We use the sum of the lattice weight, the bigram LM weight, second-order boundary weights, segment NN weights, bi-phone NN bottleneck weights, length indicators, and lexicalized bias as our final weight function for the second level of the cascade. Hinge loss is minimized with AdaGrad for 20 epochs with step size 0.01. Again, no explicit regularizer is used except early stopping based on the PER on the development set. Results with these additional weights are shown in Table 4.3. Adding the second-order weights, bigram LM, and the above NN weights gives a 1.8% absolute improvement over our best first-pass system, demonstrating the value of including such expensive features with wider context.

Table 4.3: Phoneme error rates (%) with discriminative segmental cascades. The search space for the first pass is denoted H_1, and the search space for the second pass $H_1'^{\uparrow}$ is denoted H_2, where H_1 is pruned to get H_1'.

	dev	test
1^{st} pass (H_1)	22.2	21.7
2^{nd} pass (H_2)		
+ bigram LM	19.8	
+ 2nd-order boundary features	19.2	
+ segment NN	18.9	
+ bi-phone NN bottleneck	18.8	19.9

4.4 Improving Decoding and Training Speed

We have shown that segmental cascades can be used to incorporate computationally expensive weight functions while maintaining efficiency by shifting the computationally expensive weights to later passes. In this section, instead of incorporating additional weights, we shift weights to later passes to improve decoding speed. Due to their recent success (Graves et al., 2013), we also explore long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) as encoders instead of CNN encoders.

4.4.1 LSTM Encoders

In this section, we describe long short-term memory (LSTM) networks. We start by defining a single-layer LSTM. Suppose the input vectors are $x_1, \ldots, x_T$. First we apply a linear transformation to the inputs to get

$$x'_t = \begin{pmatrix} W_{xu} \\ W_{xi} \\ W_{xf} \\ W_{xo} \end{pmatrix} x_t, \tag{4.26}$$

which can be done for $t = 1, \ldots, T$ in parallel. Next, assume $h_0 = c_0 = 0$. Suppose we have already computed h_t and c_t, and we want to compute h_{t+1} and c_{t+1}. We apply a linear transformation to get the candidates u', i', f', and o' for the cell update vectors u and the gates i, f and o.

$$\begin{pmatrix} u' \\ i' \\ f' \\ o' \end{pmatrix} = x'_t + \begin{pmatrix} W_{hu} \\ W_{hi} \\ W_{hf} \\ W_{ho} \end{pmatrix} h_t + \begin{pmatrix} 0 \\ W_{ci} \\ W_{cf} \\ 0 \end{pmatrix} c_t. \tag{4.27}$$

The update vectors u, the input gate i, and the forget gate f are then computed from the candidates by applying nonlinear transformations.

$$u = \tanh(u') \tag{4.28}$$
$$i = \text{logistic}(i') \tag{4.29}$$
$$f = \text{logistic}(f') \tag{4.30}$$

The new cell vector c_{t+1} is computed with the update vector gated by the input gate and the previous cell.

$$c_{t+1} = i \odot u + f \odot c_t \tag{4.31}$$

Finally, the output gate o is computed from the updated cell c_{t+1}, and the new hidden vector h_{t+1} is updated with the cell gated by the output gate.

$$o = \text{logistic}(o' + W_{co} c_{t+1}) \tag{4.32}$$
$$h_{t+1} = o \odot \tanh(c_{t+1}) \tag{4.33}$$

The final results after running an LSTM on $x_1, \ldots, x_T$ are $h_1, \ldots, h_T$ and $c_1, \ldots, c_T$. We use $h_{1:T} = \text{LSTM}(x_{1:T})$ to denote the process above.

Since LSTMs have an assigned direction, it is natural to run two LSTMs in opposite directions and combine the hidden vectors. Formally,

$$h^f_{1:T} = \text{LSTM}(x_{1:T}) \tag{4.34}$$
$$h^b_{T:1} = \text{LSTM}(x_{T:1}) \tag{4.35}$$
$$h_t = W_f h^f_t + W_b h^b_t \tag{4.36}$$

We use $h_{1:T} = \text{BiLSTM}(x_{1:T})$ as a shorthand for the bidirectional LSTMs. Note that the parameters of the two LSTMs are not shared. Another way to increase the complexity of the acoustic encoder is to stack them on top of each other. For example, if we stack 3 layers of bidirectional LSTMs, we get

$$h^{(1)}_{1:T} = \text{BLSTM}(x_{1:T}) \tag{4.37}$$
$$h^{(2)}_{1:T} = \text{BLSTM}(h^{(1)}_{1:T}) \tag{4.38}$$
$$h^{(3)}_{1:T} = \text{BLSTM}(h^{(2)}_{1:T}) \tag{4.39}$$

We use $h_{1:T} = \text{DBLSTM}^3(x_{1:T})$ to denote the 3-layer bidirectional LSTMs (where "D" refers to "deep"). Again the parameters of the LSTMs in each layer and each direction are not shared.

4.4.2 Experiments

As in Section 4.3, we first explore LSTMs on a frame classification task, Specifically, the log probability vector of a frame at time t is $z_t = \text{logsoftmax}(W h_t + b)$ where h_t is the hidden

vector at time t produced by an LSTM, and W and b are the parameters. We then explore segmental cascades with weight functions based on log probabilities produced by LSTMs. Experiments are conducted on the TIMIT data set. We compute 41-dimensional log mel filter bank outputs including energy, and concatenate with the delta's and double delta's to form the final 123-dimensional acoustic feature vectors. Instead of using a 192-utterance development set, we follow the Kaldi recipe (Povey et al., 2011) and use 400 utterances from the complete test set (disjoint from the core test set) as the validation set. We report numbers on the core test set, the same test set as in Section 3.5.2.

LSTM frame classification

We build a frame classifier by stacking three layers of bidirectional LSTMs. The cell and hidden vectors have 256 units. We train the frame classifier with frame-wise cross entropy and optimize with AdaGrad (Duchi et al., 2011) with mini-batch size of 1 utterance and step size 0.01 for 30 epochs. We choose the best-performing model on the development set (early stopping).

Following (Vanhoucke et al., 2013; Miao et al., 2015), we consider dropping half of the frames for any given utterance in order to save time on feeding forward. Specifically, we only use $x_2, x_4, \ldots, x_{T-2}, x_T$ to feed forward through the deep LSTM and generate $z_2, z_4, \ldots, z_{T-2}, z_T$ (assuming T is even, without loss of generality). We then copy each even-indexed output to its previous frame, i.e., $z_{i-1} = z_i$ for $i = 2, 4, \ldots, T-2, T$. During training, the cross entropy is calculated over all frames and propagated back. Specifically, let E_i be the cross entropy at frame i, and $E = \sum_{i=1}^{T} E_i$. The gradient of z_i is the sum of the gradients from the current frame and the copied frame. Dropping even-indexed frames is similar to dropping odd-indexed frames, except the outputs are copied from z_i to z_{i+1} for $i = 1, 3, \ldots, T-1$. When training LSTMs with subsampling, we alternate between dropping even- and odd-numbered frames every other epoch. The results are shown in Figure 4.3.

We observe that with subsampling the model converges more slowly than without, in terms of number of epochs. However, by the end of epoch 30, there is almost no loss in frame error rates when we drop half of the frames. Considering the more important measure of training time rather than number of epochs, the LSTMs with frame subsampling converge twice as fast as those without subsampling. For the remaining experiments, we use the log posteriors at each frame of the subsampled LSTM outputs as the inputs to the segmental models.

Segmental cascade experiments

Our baseline system, denoted R, is a first-pass segmental model with the FC weight function (Section 3.3.1). The baseline system is trained by optimizing hinge loss with AdaGrad using mini-batch size 1 utterance, step size 0.1, and early stopping for 50 epochs.

We decide to shift the FC weight function to the second pass, and train a segmental model, denoted A_1, with just the label posterior and a label-independent bias. Specifically,

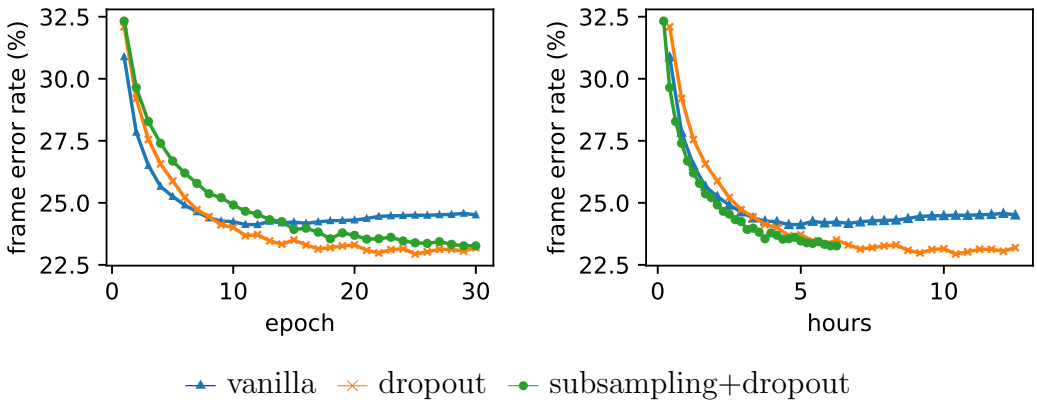


Figure 4.3: Development set frame error rates vs. number of epochs (left) and vs. training hours (right).

the label posterior weight is defined as

$$w((\ell, s, t)) = \sum_{i=s}^{t} (z_i)_\ell, \tag{4.40}$$

where $z_i = \text{logsoftmax}(Wh_i + b)$ is the log probability vector at time i based on the vector h_i produced by the LSTM. This reduces the number of features from 24528 to two. We use hinge loss optimized with AdaGrad with mini-batch size 1 and step size 1. Since we only have two features, learning converges very quickly, in only three epochs. We take the model from the third epoch to produce lattices for subsequent passes in the cascade.

Lattices are generated with max-marginal pruning with $\lambda = 0.85$. The resulting lattices have an average oracle error rates of 1.4% and average density of 213.02. We train a second-pass model, denoted A_2, with the FC weight function except that we add a "lattice score" feature corresponding to the segment score given by the two-feature system. Hinge loss is optimized with AdaGrad with mini-batch size 1, step size 0.1, and early stopping for 20 epochs.

The learning curve comparing R and A_1 followed by A_2 is shown in Figure 4.4. We observe that the learning time per epoch of the two-feature system A_1 is only one-third of the baseline system R. We also observe that training of A_2 converges faster than training R, despite the fact that they use almost identical feature functions. The baseline system achieves the best result at epoch 49. In contrast, the two-pass system is done before the baseline even finishes the third epoch.

Following Section 4.3, given the first-pass baseline, we apply max-marginal pruning to produce lattices for the second-pass baseline with $\lambda = 0.8$. The second-pass baseline features are the lattice score from the first-pass baseline, a bigram language model score, first-order length indicators, and a bias. Hinge loss is optimized with AdaGrad with mini batch size 1, step size 0.01, and early stopping for 20 epochs. For the proposed system, we produce

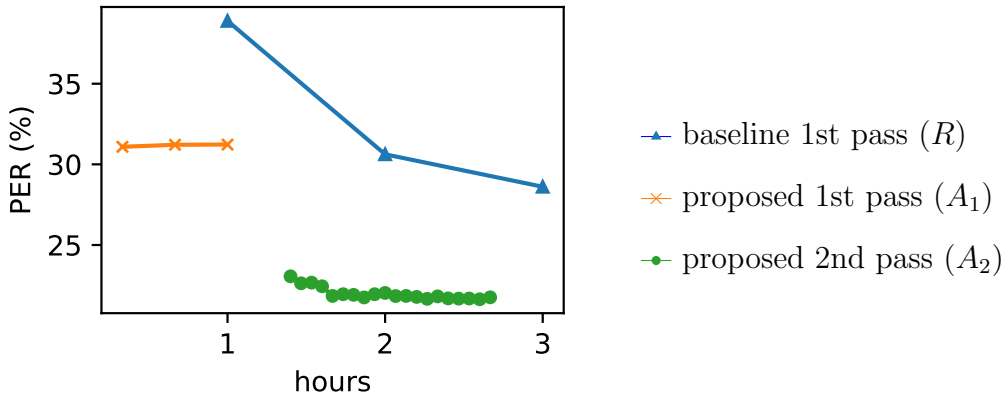


Figure 4.4: Learning curve of the proposed two-pass system compared with the baseline system. The time gap between the first pass and the second pass is the time spent on pruning.

Table 4.4: Phone error rates (%) of proposed and baseline systems.

		1st pass	2nd pass	3rd pass
baseline	dev	21.9	21.0	
	test	24.0	23.0	
proposed	dev	33.6	21.5	21.3
	test		23.7	23.4

lattices with max-marginal pruning and $\lambda = 0.3$ for the third-pass system. We use the same set of features and hyper-parameters as the second-pass baseline for the third pass.

Phone error rates of all passes are shown in Table 4.4. First, if we compare the one-pass baseline with the proposed two-pass system, our system is close to the baseline. Second, we observe a healthy improvement by just adding the bigram language model score to the second-pass baseline. The improvement for our third-pass system is small but brings our final performance to within 0.4% of the baseline second pass.

Next we report on the speedups in training and decoding obtained with our proposed approach. Table 4.5 shows the real-time factors for decoding with the baseline and proposed systems. In terms of decoding time alone, we achieve a factor of 2.4 speedup compared to the baseline. If the time of feeding forward LSTMs is included, then our proposed system is two times faster than the baseline.

Table 4.6 shows the times needed to train a system to get to the performance in Table 4.4. The speedup mostly comes from the fast convergence of the first pass. In terms of training the segmental models alone, we achieve an 18.0-fold speedup. If the time to train the LSTMs is included, then we obtain a 3.4-fold speedup compared to the baseline.

To summarize some of the above results: With a combination of the first-pass two-

Table 4.5: Real-time factors for decoding.

	1st pass	2nd pass	3rd pass	total decoding	feeding forward	total overall
baseline	0.33	0.01		0.34	0.33	0.67
proposed	0.11	0.02	0.01	0.14	0.17	0.31

Table 4.6: Hours for training the system.

	1st pass	2nd pass	3rd pass	total decoding	feeding forward	total overall
baseline	49.5	0.6		50.1	59.4	109.5
proposed	1.0	1.2	0.6	2.8	29.7	32.5

feature system and edge pruning, we prune 95% of the segments in the first-pass hypothesis space, leading to significant speedup in both decoding and training. The feed-forward time for our LSTMs is halved through frame subsampling. In the end, with a single four-core CPU, we achieve 0.31 times real-time decoding including feeding forward, which is 2.2 times faster than the baseline, and 32.5 hours in total to obtain our final model including LSTM training, which is 3.4 times faster than the baseline. Excluding the LSTMs, the segmental model decoding alone is 2.4 times faster than the baseline, and training the segmental models alone is 18 times faster than the baseline.

4.5 Summary

In this chapter, we have described discriminative segmental cascades, a multi-pass system for segmental models. Max-marginal pruning lies at the heart of segmental cascades, producing sparse lattices while having low oracle error rates. After pruning, it becomes feasible to incorporate computationally expensive weight functions, such as higher-order LMs, and weight functions based on neural networks. We demonstrate that incorporating these weights significantly improves the performance of segmental models. We also show that decoding and training speed can be significantly improved with segmental cascades by shifting weights to later passes without sacrificing accuracy.

Though we have shown improvement with segmental cascades, how the density and structure of search spaces affect the training of segmental models is still not fully known. Task-dependent priors, such as phoneme duration, are not fully explored and can potentially be integrated into the pruning procedure. The fact that beam search works well for frame-based models but fails miserably for segmental models is also worth investigating. Converting other more sophisticated search algorithms, such as A* search (Kenny et al., 1993), into pruning algorithms is a possibility that might show us signs as to why beam search fails for segmental models.

Chapter 5

End-to-End Training Approaches

In the previous chapter, we have seen how a sequence recognizer can be trained in multiple stages: a frame classifier is first trained with frame-wise cross entropy, and a segmental model is trained with a sequence loss, such as the ones defined in Section 3.4, based on the classifier's outputs. In other words, the frame classifier is trained independently from the segmental model, and is held fixed when the segmental model is being trained. A natural question to ask is whether we can further improve the sequence loss by updating the frame classifier while holding the segmental model fixed. A more general question would be whether we can improve the sequence loss by updating the frame classifier and the segmental model simultaneously.

Optimizing all of the parameters simultaneously against a single loss function is commonly referred to as end-to-end training. Since the loss function with respect to all the parameters in a neural network is non-convex, end-to-end training might be more difficult than training in multiple stages. On the other hand, since end-to-end training is jointly optimizing all parameters, it might achieve a lower loss, leading to better performance.

In this chapter, we formally define these training settings, including multi-stage training and end-to-end training, Different training approaches and losses have different training requirements. We discuss the pros and cons of these training approaches, and conduct experiments to see empirically how well they compare to each other.

5.1 Training Settings

Recall that the set of parameters Θ consists of Θ_{enc} and Θ_{dec}, where Θ_{enc} is the set of parameters in the encoder and Θ_{dec} are the rest of the parameters in the weight function.

The training approach we have seen in the previous chapter is referred to as **multi-stage training**. Specifically, the encoder is trained in the first stage with an encoder-specific loss

$$\mathcal{L}_{\text{enc}}(\Theta_{\text{enc}}), \tag{5.1}$$

for example, the frame-wise cross entropy. Let $\hat{\Theta}_{\text{enc}} = \text{argmin}_{\Theta_{\text{enc}}} \mathcal{L}_{\text{enc}}(\Theta_{\text{enc}})$. In the second

stage, we solve the optimization problem

$$\min_{\Theta_{\text{dec}}} \mathcal{L}_{\text{seq}}(\hat{\Theta}_{\text{enc}}, \Theta_{\text{dec}}), \tag{5.2}$$

where $\mathcal{L}$ is a sequence loss, such as hinge loss, log loss, or marginal log loss, defined in Section 3.4. Note that in the second stage the parameters of the encoder are held fixed. Typically, after the first stage, we can feed the inputs in the entire data set forward, and reuse the vectors while optimizing the sequence loss to save computation. Another benefit of optimizing (5.2) with $\hat{\Theta}_{\text{enc}}$ fixed is that the loss function $\mathcal{L}$ might be convex in Θ_{dec}, for example, when hinge loss or log loss is used and the weight function is linear in Θ_{dec}. The characteristics of convex functions are well-understood (Boyd and Vandenberghe, 2004), and many algorithms, other than first-order methods, have been developed for optimizing convex functions (Bertsekas et al., 2003).

In contrast to multi-stage training, we refer to solving

$$\min_{\Theta_{\text{enc}}, \Theta_{\text{dec}}} \mathcal{L}_{\text{seq}}(\Theta_{\text{enc}}, \Theta_{\text{dec}}) \tag{5.3}$$

as **end-to-end training**, where all parameters Θ_{enc} and Θ_{dec} are optimized jointly. Since all parameters are optimized jointly, the loss function is in general non-convex. Though end-to-end training might be harder to solve, it might lead to a lower loss value. Typically, the solution found by multi-stage training is a reasonably good solution for the sequence loss. As a result, a safe strategy, which we refer to as **end-to-end fine-tuning**, may be to use the solution from multi-stage training as an initialization for solving (5.3). One major difference between multi-stage training and end-to-end training is their resource requirements. Multi-stage training requires extra labels for optimizing the encoder loss $\mathcal{L}_{\text{enc}}$, while end-to-end training requires only whatever the sequence loss $\mathcal{L}_{\text{seq}}$ requires.

5.2 Experiments

For experiments, we use the same phoneme recognition setting on the TIMIT data set as in Section 4.4. In the sections below, we first compare the performance of CNNs and LSTMs, and then compare two training settings for training segmental models, multi-stage training followed by end-to-end fine-tuning and end-to-end training from random initialization. We also explore various weight functions paired with various loss functions in the context of end-to-end training from random initialization.

5.2.1 Multi-stage training

Recall that the multi-stage training approach demonstrated in Section 4.3 consists of two stages: first training a frame classifier, and second training a segmental model based on the output probabilities of the frame classifier. Instead of a CNN encoder, a 3-layer bidirectional LSTM with hidden vector size of 250 is trained for frame classification. Parameters are

Table 5.1: Frame error rates (FER) and phoneme error rates (PER) on the development set comparing the CNN and the LSTM encoder.

	FER (%)	PER (%)
CNN	22.3	22.2
LSTM	17.8	19.7

initialized according to (Glorot and Bengio, 2010). Vanilla SGD is used to minimize frame-wise cross entropy for 20 epochs with step size 0.1 and a mini-batch size of 1 utterance. Gradients are clipped to norm 5 if the norm is above 5. The best model (measured by frame error rates on the development set) is chosen among the 20 epochs, and trained for another 20 epochs with step size 0.75 decayed by 0.75 after every epoch. For phoneme recognition, we use segmental models with the FC weight function, the same setting as in Section 4.3. Hinge loss with the overlap cost are minimized with RMSProp (Mukkamala and Hein, 2017) for 20 epochs with step size 10^{-4} and decay 0.9,

The frame error rates (FER) and phoneme error rates (PER) compared to the CNN in Section 4.3 are shown in Table 5.1. The LSTM performs better than the CNN for both tasks. We suspect that the LSTM's superior performance is due to the training approaches: the LSTM is trained on entire utterances, while the CNN is trained on batches of 15-frame windows. This effect was also observed in (Jaitly et al., 2014). For the rest of the experiments, we use 3-layer LSTMs as our encoders.

Next, we compare segmental models trained with different losses, including hinge loss, log loss, and marginal log loss. We use the FC weight function and a maximum segment duration of 30 frames. All losses are minimized with RMSProp for 20 epochs with step size 10^{-4} and decay 0.9. Results are shown in Table 5.2. No explicit regularizers are used except early stopping on the development set. All losses achieve similar results with log loss slightly behind. Note that although marginal log loss does not require manual alignments, the frame-wise cross entropy used to train the frame classifier still does.

After two-stage training, we use the trained segmental models and frame classifiers as an initialization for end-to-end training. Each loss is optimized with vanilla SGD for 20 epochs with step size 0.1 decayed by 0.75 after each epoch. Gradients are clipped with norm 5. Dropout is added to the LSTMs with rate 0.2, and no other regularizers are used except early stopping on the development set. Results are shown in Table 5.2. End-to-end fine-tuning is able to improve the error rates for all losses with marginal log loss being the best performer.

After end-to-end fine-tuning, the encoders are not constrained to perform well on frame-wise cross entropy. We evaluate the encoders on the frame classification task to see how much the encoders deviate from the initialization, and whether the intermediate representations trained for frame classification are still preserved. Results are shown in Table 5.3. The encoders fine-tuned with hinge loss and log loss deviate less compared to the one fine-tuned with marginal log loss. We suspect this is because hinge loss and log loss use the ground-truth

Table 5.2: Phoneme error rates (%) comparing different losses with two-stage training (2s) followed by end-to-end fine-tuning (ft).

weight function	loss	2s dev	2s test	+fine-tuning dev	+fine-tuning test
FC	hinge	19.7		18.2	
	log	21.0		17.3	
	MLL	21.8		16.9	19.5

Table 5.3: Frame error rates (%) of LSTM encoders after end-to-end fine-tuning with different losses .

		FER
frame-wise cross entropy		17.8
FC	hinge	18.4
	log	18.7
	MLL	26.8

alignments in training while marginal log loss does not.

5.2.2 End-to-end training

After observing the success of end-to-end fine-tuning, we conduct experiments to see whether it is possible to train segmental models end to end from random initialization. We use the FC weight function and a maximum duration of 30 frames, the same setting as in the multi-stage experiments. We initialize the parameters according to (Glorot and Bengio, 2010). Each loss is optimized with vanilla SGD for 20 epochs with step size 0.1 and gradient clipping with norm 5. The best performing model, chosen in the first 20 epochs, is trained for another 20 epochs with vanilla SGD step size 0.75 decayed by 0.75 after each epoch. Dropout of rate 0.2 is used throughout the training process. No other regularizers are used except early stopping on the development set. Results are shown in Table 5.4.

There is no significant difference in terms of performance between models trained end to end from random initialization and ones trained in multiple stages followed by fine-tuning. This shows that it is possible to train segmental models end to end from random initialization, and that models trained in multiple stages can serve as a good initialization for end-to-end training.

Table 5.4: Phoneme error rates (%) on the development set comparing multi-stage training followed by end-to-end fine-tuning (2s+ft) and end-to-end training from random initialization (e2e).

weight function	loss	2s+ft	e2e
FC	hinge	18.2	18.4
	log	17.3	17.4
	MLL	16.9	16.7

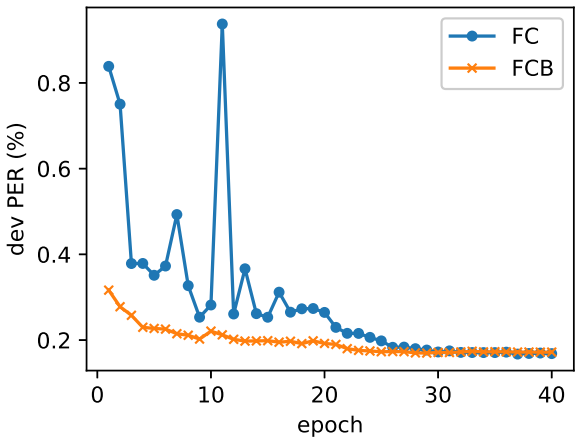


Figure 5.1: Learning curves of segmental models trained end to end with the FC weight function and FCB weight function with marginal log loss.

5.2.3 Weight function comparison

Next, we consider various weight functions for end-to-end training from random initialization. Note that the initialization scheme in (Glorot and Bengio, 2010) is not suitable the for log-soft-max operation used after the LSTMs for producing log probabilities, because the variances before and after the log-soft-max operation are very different. Based on this intuition, we define a new weight function, called **FC bottleneck (FCB)** weight function, removing the log-soft-max layer when using the FC weight function. More precisely, suppose $h_1, \ldots, h_T$ is the sequence of vectors produced by an LSTM. Whereas the FC weight function uses the average, samples, and boundaries of $z_i = \text{logsoftmax}(Wh_i + b)$ for $i = 1, \ldots, T$, the FCB weight function uses the average, samples, and boundaries of h_i's directly. The learning curves of segmental models trained end to end with FC weight function and the FCB weight function are shown in Figure 5.1. We observe significantly faster convergence with the FCB weight function than with the FC weight function.

We do not compare the MLP weight function to the FC and FCB weight functions in the current setting, because it is too time-consuming to train segmental models with the MLP weight function on the entire search space. To train segmental models with the MLP weight function, we follow (Lu et al., 2016) and reduce the time resolution by a factor of four using pyramid LSTMs. Specifically, for an input sequence $x_{1:T}$,

$$h^{(1)}_{1:T} = \text{BLSTM}(x_{1:T}) \tag{5.4}$$

$$z^{(1)}_{1:\lfloor T/2 \rfloor} = \text{subsample}(h^{(1)}_{1:T}) \tag{5.5}$$

$$h^{(2)}_{1:\lfloor T/2 \rfloor} = \text{BLSTM}(z^{(1)}_{1:\lfloor T/2 \rfloor}) \tag{5.6}$$

$$z^{(2)}_{1:\lfloor T/4 \rfloor} = \text{subsample}(h^{(2)}_{1:\lfloor T/2 \rfloor}) \tag{5.7}$$

$$h^{(3)}_{1:\lfloor T/4 \rfloor} = \text{BLSTM}(z^{(2)}_{1:\lfloor T/4 \rfloor}) \tag{5.8}$$

where

$$h_2, h_4, \ldots, h_{2(k-1)}, h_{2k} = \text{subsample}(h_1, h_2, \ldots, h_{2k}) \tag{5.9}$$

if we drop a frame for every two frames. The final output sequence $h^{(3)}$ is then used to compute edge weights. An encoder with several layers of LSTMs interleaved with subsampling is also known as a **pyramid LSTM** and has been used in other prior work, e.g., (Chan et al., 2016). In addition, we set the maximum duration to 8 frames, effectively $4 \times 8 = 32$ frames on the original time scale. The reduced time resolution and maximum duration result in search spaces 16 times smaller than the original ones, making it feasible to compute the MLP weight function on all edges.

With 3-layer pyramid LSTMs, we compare segmental models trained with the three weight functions and three losses. All models are trained with the same step size scheduling as in the previous experiments. Results with and without pyramid LSTMs are both shown in Table 5.5. First, we observe that using pyramid LSTMs leads to a degradation in performance for segmental models with the FC and FCB weight functions. However, using pyramid LSTMs has less impact on segmental models trained with marginal log loss compared to the ones trained with hinge loss and log loss. We suspect this is because hinge loss and log loss rely on the ground-truth alignments for training while marginal log loss does not. Second, we observe that while segmental models with the FC and FCB weight functions are able to achieve low hinge loss values, the ones with the MLP weight function completely fail to do so. Finally, the best result on the development set obtained by using the FC weight function is slightly worse than using the MLP weight function.

Segmental models trained with marginal log loss do not require ground-truth alignments during training, making annotating data sets easier, cheaper, and less time-consuming. However, it is harder to diagnose errors made by models trained end to end because the intermediate representations are not interpretable. Models trained in multiple stages are easier to diagnose by looking at the performance of proxy tasks, such as frame classification in our case.

Table 5.5: Phoneme error rates (%) comparing different losses for end-to-end training from random initialization.

weight function	loss	regular LSTM		pyramid LSTM	
		dev	test	dev	test
FC	hinge	18.4		23.4	
	log	17.4		22.4	
	MLL	16.7	19.6	17.9	
FCB	hinge	18.6		24.2	
	log	17.8		23.0	
	MLL	17.0		17.7	
MLP	hinge			67.7	
	log			22.4	
	MLL			17.1	19.2

Table 5.6: Average number of minutes for one epoch of end-to-end training on TIMIT.

weight function	loss	regular LSTM	pyramid LSTM
FC	hinge	119	38
	log	177	44
	MLL	260	47
MLP	hinge		83
	log		110
	MLL		116

5.2.4 Training time comparison

We compare the time to train segmental models end to end. Times are measured on a 3GHz 4-core CPU. Multithreading is only used for matrix operations; all FST algorithms are implemented single-threaded. Results are shown in Table 5.6. Reducing the time resolution with pyramid LSTMs significantly improves the runtime for all losses. The MLP weight function is about 2.5 times slower than the FC weight function.

5.2.5 Segmentation Analysis

We are interested in how well segmental models recover the phone boundaries in the end-to-end setting when manual alignments are not used during training. The task, commonly known as phonetic segmentation, is to align the phonetic transcription to the acoustic frames.

Table 5.7: Boundary error rates (%) of phonetic segmentation at different tolerance levels on the TIMIT core test set (except the last row) for segmental models trained end to end with marginal log loss compared to past results.

	$t \leq 0$ms	$t \leq 10$ms	$t \leq 20$ms	$t \leq 30$ms	$t \leq 40$ms
(Keshet et al., 2007)		20.3%	7.9%	3.8%	1.9%
(Rendel et al., 2012)		19.5%	6.3%	2.4%	1.0%
(Yuan et al., 2013)	22.6%	6.1%	2.6%	1.2%	0.6%
seg FC	25.0%	10.0%	5.1%	3.1%	2.1%
seg FC on train	24.2%	9.6%	5.0%	3.1%	2.1%

Unlike sequence prediction, the phonetic transcription is known. We measure the error rates of the boundaries under some tolerance level. Specifically, suppose the predicted boundary is at time $\hat{b}$ and the ground truth is at time b. The predicted boundary $\hat{b}$ is considered correct if the absolute distance $t = |\hat{b} - b| \leq \tau$ for some τ. The error rates under various thresholds for the segmental model with the FC weight function trained with marginal log loss are shown in Table 5.7. Models proposed in (Keshet et al., 2007; Rendel et al., 2012; Yuan et al., 2013), shown in row 1–3 in Table 5.7, are specifically trained for phonetic segmentation. Though the alignment results are behind models trained specifically to align, the segmental model trained with marginal log loss is not supervised with any ground-truth alignments, and its performance is sufficient for many tasks.

We analyze the errors made by the segmental model. The top 30 most errorful boundary types sorted by error rates are shown in Table 5.8. We observe that most of the boundaries in Table 5.8 involve silences, vowels, and semi-vowels. The ones involving silences often appears at the start or the end of utterances. Boundaries between two vowels and between a semi-vowel and a vowel are known to be ambiguous (Schwartz and Makhoul, 1975; Umeda, 1975). We see few alignment errors between consonants and vowels.

The top 30 most errorful boundary types sorted by error counts are shown in Table 5.9. We are also interested in boundary types that have hight error counts but not necessary have high error rates. Compared to Table 5.8, we see a very different pattern. Most of the errors in Table 5.9 involve silences (including voiced and unvoiced closures). For 0ms and 10ms, the segmental model errs at the boundaries between closures and the bursts of stop consonants, such as /b/, /d/, /g/, /p/, /t/, and /g/. It also errs at boundaries involving liquids, such as /r/ and /l/, across all tolerance levels. Similarly to semi-vowels, the boundaries of liquids are also known to be ambiguous (Schwartz and Makhoul, 1975; Umeda, 1975). The errors for higher tolerance levels, such as 30ms and 40ms, mostly involve silences that appear at the start or end of utterances. We notice that 10.5% of the silences in the training set are longer than 30 frames. We suspect the 30-frame maximum duration constraint is too restrictive for silences.

Some phoneme boundaries are inherently ambiguous. In other words, there are cases

where segments do not have clear start times and end times. We argue that segmental models are still well-motivated and suitable for these tasks, such as phonetic recognition. First, the search space are stochastic when the path weights are interpreted as probabilities. In fact, lattices, when first proposed by Schwartz and Makhoul (1975), were used to account for phoneme boundary ambiguity. Training segmental models with marginal log loss also takes the ambiguity into account by marginalizing over all segmentations. For decoding, ideally we want to find the label sequence that has the maximum marginal weight

$$\operatorname*{argmax}_{y} \sum_{p \in \Gamma(y)} w(p), \tag{5.10}$$

where $\Gamma(y)$ is the set of paths with the label sequence y. However, the above is not tractable and we typically approximate it by finding the most probable path

$$\operatorname*{argmax}_{p \in \mathcal{P}} w(p). \tag{5.11}$$

Another approach to approximate (5.10) is to use beam search. In summary, segmental models are capable of handling ambiguous segment boundaries if the training and decoding take the ambiguity into account.

5.3 Summary

We have compared different training approaches for segmental models, including multi-stage training and end-to-end training. End-to-end training improves over multi-stage training, and end-to-end training from random initialization is on par with end-to-end fine-tuning. Multi-stage training is useful for diagnosing end-to-end training, because every model found by multi-stage training is a valid model for end-to-end training objectives. We have shown that segmental models can be trained with marginal log loss from random initialization, without requiring manual alignments for training. As a byproduct, segmental models trained with marginal log loss perform reasonably well on phonetic segmentation. Segmentation errors made by end-to-end segmental models are aligned with the studies in acoustic phonetics.

Table 5.8: Top 30 boundaries and its preceding and following phonemes (with at least five occurrences) in the training set sorted according to alignment error rates of the segmental model trained with marginal log loss.

$t \leq 0$ms		$t \leq 10$ms		$t \leq 20$ms		$t \leq 30$ms		$t \leq 40$ms	
en	w	ix	ae	ix	ax	uw	uw	ix	ae
sil	ae	ix	eh	ih	eh	ax	eh	ng	ey
dx	aw	ih	eh	ao	aw	ix	ae	uw	uw
sil	y	sil	aw	ax	eh	ax	aa	ao	aw
ao	w	ao	aw	ax	aw	eh	y	eh	y
ix	ae	y	iy	eh	y	ih	eh	ih	iy
oy	ae	ah	aa	uw	uw	ix	eh	hh	sil
s	y	ax	eh	ix	eh	hh	sil	ix	eh
oy	ao	ao	ey	ax	aa	ng	ey	ih	eh
jh	aa	ax	aw	ao	eh	sil	aa	ih	sil
sh	ay	ng	ey	sil	ao	ao	eh	w	oy
jh	uh	ax	ay	cl	ow	ow	ow	ax	ay
f	w	ao	ow	ih	iy	cl	ao	vcl	ah
ix	eh	eh	y	sil	ay	ao	aw	ow	ow
z	ay	cl	ah	ix	ih	ah	aa	cl	ao
el	ay	uw	uw	y	iy	ih	iy	ao	eh
ih	eh	ey	ey	aw	ay	ax	ow	ix	ih
z	y	aa	ax	ax	ay	ih	sil	sil	aa
sil	aw	aw	oy	ao	ow	ix	ih	sil	ao
ax	ow	w	en	sil	aa	w	oy	er	aw
jh	ao	sil	y	sil	w	ax	aw	sil	ay
n	oy	sil	w	ow	ow	aw	ay	oy	ao
ng	eh	sil	ow	cl	ao	ax	ay	ow	ey
l	en	sil	ay	ix	ow	vcl	ah	ax	ow
epi	dh	sil	ah	er	sil	ey	ey	ay	ey
aw	hh	sil	aa	sil	ah	sil	ao	ax	aa
s	r	sil	r	hh	sil	sil	ow	y	iy
m	aw	sil	ao	ng	ey	ix	iy	ax	eh
sh	ah	sil	dh	ao	ih	sil	ay	ao	ey
aa	z	ix	ih	iy	iy	oy	ae	ax	aw

Table 5.9: Top 30 boundaries and its preceeding and following phonemes (with at least five occurrences) in the training set sorted according to alignment error counts of the segmental model trained with marginal log loss.

$t \leq 0$ms		$t \leq 10$ms		$t \leq 20$ms		$t \leq 30$ms		$t \leq 40$ms	
vcl	b	vcl	b	sil	dh	sil	dh	iy	sil
vcl	d	sil	dh	sil	w	sil	w	er	sil
cl	k	vcl	d	er	sil	er	sil	sil	dh
cl	t	vcl	g	iy	sil	iy	sil	n	sil
cl	p	sil	w	n	sil	n	sil	sil	w
s	cl	t	r	t	r	s	sil	s	sil
ix	n	er	sil	k	l	cl	sil	cl	sil
vcl	g	iy	sil	sil	hh	m	sil	ng	sil
sil	dh	ao	r	ao	r	ng	sil	m	sil
ix	cl	p	r	s	sil	sil	hh	z	sil
vcl	jh	sil	hh	p	r	sil	r	m	sil
n	cl	l	iy	m	sil	ao	r	z	sil
ao	r	aa	r	p	l	z	sil	sil	aa
n	vcl	n	sil	vcl	b	ao	l	sil	ay
ix	z	y	uw	ng	sil	sil	aa	sil	hh
ix	vcl	p	l	cl	sil	sil	ae	d	sil
s	sil	k	l	aa	r	sil	y	vcl	sil
aa	r	r	iy	sil	m	vcl	sil	l	sil
l	iy	s	sil	y	uw	el	sil	ao	l
r	iy	n	vcl	k	r	sil	m	en	sil
iy	cl	sil	b	ao	l	sil	ay	sil	ao
y	uw	ih	z	l	iy	d	sil	sil	ae
ax	cl	k	r	sil	r	k	l	t	sil
dx	ix	vcl	jh	r	ay	y	uw	sil	ah
t	r	ix	z	sil	b	l	sil	ao	r
er	cl	n	cl	sil	n	aa	r	aa	r
ix	s	m	sil	r	aa	t	sil	ix	sil
p	r	z	sil	sil	ae	en	sil	sil	y
cl	ch	sil	m	z	sil	sil	ah	el	sil
dh	ax	k	w	k	w	sil	ax	sil	r

Chapter 6

Historical Overview of Segmental Models

Automatic speech recognition (ASR) has been posed as a graph search problem since the 1970s (Jelinek, 1976). A search space, represented as a graph, is first constructed; edges in the graph are assigned weights based on the input and the edges; to predict, we simply run the shortest-path algorithm on the graph. The graph search paradigm has been popularized by the use of hidden Markov models (HMM) (Rabiner, 1989; Jelinek, 1998), where the shortest-path algorithm is known as the Viterbi algorithm. In the probabilistic setting, finding the shortest path can be seen as finding the maximum a posteriori solution, justifying the graph search paradigm. The probabilistic view has spawned many algorithms, such as segmental k-means (Juang and Rabiner, 1990) and expectation maximization (Russell and Moore, 1985), for estimating parameters in the weight function.

Segmental models follow the same graph search paradigm while having a different type of search spaces and different weight functions. Many variants of segmental models were proposed in the past, such as stochastic segment models (Ostendorf and Roukos, 1989; Ostendorf et al., 1996), semi-Markov HMMs (Russell and Cook, 1987), segmental HMMs (Russell, 1993; Gales and Young, 1993a), semi-Markov conditional random fields (CRF) (Sarawagi and Cohen, 2005), and segmental CRFs (Zweig and Nguyen, 2009). However, the type of search spaces and the shortest-path algorithm stay the same. In this chapter, we review these variants from the graph search point of view.

The idea of using segmental features for speech recognition can be traced back to the 1970s (Weinstein et al., 1975). The definition of segmental models was not explicit. Most of the studies still followed the graph search paradigm, though the graph structures were typically constructed from a lexicon with a small vocabulary, and the weights on the edges were typically estimated with heuristics or a small amount of data (Kopec and Bush, 1985; Cole et al., 1983).

In the following sections, we categorize variants of segmental models as either generative or discriminative. This categorization also aligns well with their rough chronological order.

6.1 Generative Segmental Models

Hidden semi-Markov Models are arguably the first segmental models applied to speech recognition (Levinson, 1986; Russell and Cook, 1987). Given a sequence of observations $x = (x_1, \ldots, x_T)$ of length T, let $y = (\ell_1, \ldots, \ell_K)$ be the label sequence and $z = ((s_1, t_1), \ldots, (s_K, t_K))$ be the segmentation, where $\ell_1, \ldots, \ell_K \in L$ for some discrete label set L, and $s_1 = 1$, $t_k = T$, $s_k \leq t_k$, $t_{k-1} + 1 = s_k$, for $k = 2, \ldots, n$. Let $e_k = (\ell_k, s_k, t_k)$ for $k = 1, \ldots, K$. The probability of $x_{1:T}$ defined by hidden semi-Markov models is

$$p(x, y, z) = p(x_{1:T}, e_{1:K}) = p(e_1) \prod_{k=2}^{K} p(e_k|e_{k-1}) \prod_{k=1}^{K} p(x_{s_k:t_k}|e_k). \tag{6.1}$$

The generative story is straightforward: the segments are generated one by one, following a Markov chain, and each segment $e = (\ell, s, t)$ generates $t - s + 1$ observations. Hidden Markov models are special cases of hidden semi-Markov models with the constraints $K = T$ and $s_k = t_k$ for $k = 1, \ldots, K$.

There are many ways to define and parameterize $p(e_k|e_{k-1})$ and $p(x_{s_k:t_k}|e_k)$. The most common assumptions are

$$p(e_k|e_{k-1}) = p(\ell_k|\ell_{k-1}) \tag{6.2}$$

$$p(x_{s_k:t_k}|e_k) = p(x_{s_k:t_k}|t - s + 1, \ell_k)p(t - s + 1|\ell_k) \tag{6.3}$$

where $p(\ell_k|\ell_{k-1})$ is commonly known as the transition probability, $p(x_{s_k:t_k}|t - s + 1, \ell_k)$ the emission probability, and $p(t - s + 1|\ell_k)$ the duration probability. The observations within a segment are typically assumed to be independent, i.e.,

$$p(x_{s_k:t_k}|t - s + 1, \ell_k) = \prod_{j=s_k}^{t_k} \mathcal{N}(x_j; \mu_{\ell_k}, \sigma^2_{\ell_k}), \tag{6.4}$$

where μ_ℓ and σ^2_ℓ are the mean and variance of the Gaussian distribution for the label ℓ. The single Gaussian case can be easily extended to a mixture of Gaussians. Continuously variable duration HMMs (Levinson, 1986), stochastic segment models (Ostendorf and Roukos, 1989) and segmental HMMs (Russell, 1993; Gales and Young, 1993a) are all hidden semi-Markov models with the above assumptions. The difference among them lies in how the emission probability $p(x_{s_k:t_k}|t - s + 1, \ell_k)$ is defined. The subtle differences are summarized in (Gales and Young, 1993b; Ostendorf et al., 1996). The duration probability is typically defined by a Poisson distribution (Russell and Moore, 1985) or Gamma distribution (Levinson, 1986). See (Ostendorf et al., 1996) and the citations therein for other options.

Decoding with hidden semi-Markov models is done by solving

$$\operatorname*{argmax}_{y,z} p(y, z|x) = \operatorname*{argmax}_{y,z} \log p(y, z|x) = \operatorname*{argmax}_{y,z} \log \frac{p(x, y, z)}{p(x)} \tag{6.5}$$

$$= \operatorname*{argmax}_{y,z} \log p(x, y, z), \tag{6.6}$$

where y is the label sequence and z is the segmentation, and is equivalent to finding the maximum-weight path with

$$w((\ell, s, t)) = \log p(t - s + 1|\ell) + \sum_{j=s}^{t} \log \mathcal{N}(x_j; \mu_\ell, \sigma_\ell^2). \tag{6.7}$$

The term $p(\ell'|\ell)$ is ignored here for simplicity, but can be included once the search space is self-expanded (Section 4.2.1). Training hidden semi-Markov models can be done by maximizing the likelihood of the training set. The likelihood can be maximized using gradient-based methods or expectation maximization (EM) (Russell and Moore, 1985). See (Gales and Young, 1993b) for a detailed explanation of the EM algorithm for estimating the parameters of hidden semi-Markov models.

Originally motivated by using rich segmental features (Zue et al., 1989), the SUMMIT system was defined as a generative model (Glass et al., 1996). However, Glass et al. (1996) introduced the notion of anti-phones and later Chang and Glass (1997) introduced near-misses, training the system with a discriminative touch. It held the state-of-the-art result for speaker-independent phoneme recognition on TIMIT (Halberstadt, 1998) until the rise of deep neural networks (Mohamed et al., 2009).

6.2 Discriminative Segmental Models

Since maximum mutual information was introduced as a training criterion for HMMs (Bahl et al., 1986), ASR studies have gradually shifted from generative to discriminative modeling.

Parallel to the development of segmental models in the ASR community, Sarawagi and Cohen (2005) proposed **semi-Markov CRFs** for named entity recognition. Using the same notation as in the previous section, the probability of a label sequence $y = (\ell_1, \ldots, \ell_n)$ and a segmentation $z = ((s_1, t_1), \ldots, (s_K, t_K))$ given an observation sequence $x = (x_1, \ldots, x_T)$ is defined as

$$p(y, z|x) = \frac{1}{Z(x)} \exp\left(\sum_{k=1}^{K} \theta^\top \phi(x, e_k)\right) \tag{6.8}$$

where $Z(x) = \sum_{y',z'} \exp(\sum_{e \in (y',z')} \theta^\top \phi(x, e))$ and $e \in (y', z')$ is a shorthand for enumerating $(\ell_1, s_1, t_1), \ldots, (\ell_{K'}, s_{K'}, t_{K'})$ for $y' = (\ell_1, \ldots, \ell_{K'})$ and $z' = ((s_1, t_1), \ldots, (s_{K'}, y_{K'}))$ of length K'. The function ϕ, called the feature function, extracts feature vectors that are intended to correlate well with y and z. The parameter vector θ can be learned by maximizing the likelihood of the training set.

Decoding with semi-Markov CRFs is done by solving

$$\operatorname*{argmax}_{y,z} \log p(y, z|x) = \operatorname*{argmax}_{y,z} \sum_{e \in (y,z)} \theta^\top \phi(x, e), \tag{6.9}$$

and is equivalent to finding the maximum-weight path if we let $w(x, e) = \theta^\top \phi(x, e)$. Similarly, training is done by maximizing the conditional likelihood of the data, and is equivalent to minimizing the negative log likelihood, or log loss.

Heavily influenced by (Sarawagi and Cohen, 2005), Zweig and Nguyen (2009) proposed **segmental CRFs**. The difference between segmental CRFs and semi-Markov CRFs lies in the training loss. Instead of optimizing the conditional likelihood $p(y, z|x)$, segmental CRFs optimize the marginal likelihood

$$p(y|x) = \sum_z p(y, z|x). \tag{6.10}$$

The connection between the marginal likelihood and marginal log loss is clear once we take the log of the probability distribution.

As we defined in Chapter 3, we distinguish between segmental models that consider the entire search space and ones that do not. The former are called first-pass segmental models. Whether a segmental model is first-pass or not is independent of its definition. For example, semi-Markov CRFs were used as first-pass segmental models in (Sarawagi and Cohen, 2005); segmental CRFs were first used as a second-pass model in (Zweig and Nguyen, 2009), and were later used as a first-pass model in (Zweig, 2012).

In Table 6.1, we provide a set of highlights of results in the development of segmental models on the TIMIT data set. Zweig (2012) was the first to explore discriminative segmental models that search over sequences and segmentations exhaustively, and did not use neural networks. He and Fosler-Lussier (2012) first used (shallow) neural network-based frame classifiers to define weight functions, and later extended the idea to deep neural networks in (He, 2015). Abdel-Hamid et al. (2013) were the first to use deep convolutional neural networks for the weight functions, and were the first to train segmental models end to end. We have compared different losses and training strategies for segmental models, first in a rescoring framework (Tang et al., 2014) and then in first-pass segmental models (Tang et al., 2016). We also introduced segment-level classifiers and segmental cascades for incorporating them (and other expensive features) into segmental weight functions (Tang et al., 2015). Lu et al. (2016) introduced an LSTM-based weight function for every segment, and were also the first to use pyramid LSTMs to speed up inference for segmental models.

6.3 Summary

In this chapter, we have reviewed variants of segmental models categorized as either generative or discriminative in rough chronological order. We review hidden semi-Markov models, a broad class of generative segmental models subsuming stochastic segment models and segmental HMMs. We then review semi-Markov CRFs, the discriminative counterpart of hidden semi-Markov models. We have established connections between these special cases and the general segmental models defined in Chapter 3.

Table 6.1: TIMIT PERs (%) for various segmental models compared with HMMs and the current state of the art. The acoustic features are speaker-independent (spk indep) or speaker-adapted with mean and variance normalization (mvn) or maximum likelihood linear regression (fMLLR) (Povey et al., 2011). Some results were obtained with MFCCs and some with log filter bank features.

	spk indep	+mvn	+fMLLR
HMM-DNN (Povey et al., 2011)	21.4		18.3
HMM-CNN (Tóth, 2015)	16.5		
SUMMIT (Halberstadt and Glass, 1998; Glass, 2003)	24.4		
segmental CRF (SCRF) (Zweig, 2012)	33.1		
SCRF + shallow NN (He and Fosler-Lussier, 2012)	26.5		
SCRF + DNN (He, 2015)			19.1
deep segmental NN (Abdel-Hamid et al., 2013)	21.9		
segmental cascades (Tang et al., 2015)	19.9		
segmental RNN (SRNN) (Lu et al., 2016)		18.9	17.3
two-stage + end-to-end training (Tang et al., 2016)	19.7		
SRNN + multitask (Lu et al., 2017; Tang et al., 2017)		18.5	17.5
Additional results in this work			
two-stage + end-to-end seg FC	19.6		
end-to-end seg FC	19.6		
end-to-end seg MLP	19.2		

Chapter 7

A Unified Framework for Graph Search-Based Models

Beyond segmental models, this chapter reviews other modern models trained end to end within the graph search paradigm. We review connectionist temporal classification (CTC) (Graves et al., 2006), a popular approach for training frame-based LSTMs end to end. Drawing on the connection between CTC and marginal log loss, we propose a framework, consisting of search spaces (represented as FSTs), weight functions, and training losses, that can encompass many end-to-end models, such as hidden Markov models trained with lattice-free maximum mutual information (Povey et al., 2016), and LSTMs trained with CTC, as special cases.

We compare end-to-end segmental models and end-to-end frame-based models, including one-state HMMs and two-state HMMs with LSTM encoders, and LSTMs trained with CTC. Having these end-to-end models within the unified framework allows us to see the effect of each component while holding the other components fixed.

7.1 Connectionist Temporal Classification

In this section, we review connectionist temporal classification (CTC) (Graves et al., 2006). While being conceptually simple, LSTMs trained with CTC were the state of art for phonetic recognition in 2013 (Graves et al., 2013), and have achieved competitive results on large-vocabular speech recognition (Miao et al., 2015; Maas et al., 2015; Zweig et al., 2017).

Consider a sequence of acoustic vectors $x_1, \ldots, x_T$ and its corresponding labels $y_1, \ldots, y_n$, where $y_i \in L$ for $i = 1, \ldots, n$ and some label set L. We assume $n < T$ because a phoneme is typically more than a frame long. Suppose there exists a function $\mathcal{P}$ that maps $y_1, \ldots, y_n$ to a path $a_1, \ldots, a_T$, where $a_t \in L'$ for some other label set L'. Minimizing $p(a_{1:T}|x_{1:T})$ can be done by simply minimizing the frame-wise cross entropy

$$p(a_{1:T}|x_{1:T}) = \prod_{t=1}^{T} p(a_t|x_t). \tag{7.1}$$

Graves et al. (2006) proposed a mapping $\mathcal{P}$ as follows. Given a sequence $a_1, \ldots, a_T$ where $a_t \in L \cup \{\varnothing\}$ for $t = 1, \ldots, T$ and $\varnothing$ is the blank symbol. Let $\mathcal{B}$ be a function that first replaces duplicate contiguous labels into a single label, and second removes all the blank symbols. For example,

$$\texttt{k ae t} = \mathcal{B}(\varnothing\ \texttt{k k}\ \varnothing\ \texttt{ae ae t t}\ \varnothing). \tag{7.2}$$

Graves et al. (2006) used

$$\mathcal{P}(y) = \mathcal{B}^{-1}(y) = \{a = (a_1, \ldots, a_T) : a_t \in L \cup \{\varnothing\}, \mathcal{B}(a) = y\} \tag{7.3}$$

to map a label sequence to a sequence of the same length as the input sequence. However, this function $\mathcal{P}$ returns a set of sequences, so the objective is modified to

$$p(y_{1:n}|x_{1:T}) = \sum_{a_{1:T} \in \mathcal{B}^{-1}(y_{1:n})} p(a_{1:T}|x_{1:T}) = \sum_{a_{1:T} \in \mathcal{B}^{-1}(y_{1:n})} \prod_{t=1}^{T} p(a_t|x_t), \tag{7.4}$$

marginalizing over all the possible paths.

Given an input sequence x, predicting a label sequence is done by finding

$$\operatorname*{argmax}_y p(y|x) = \operatorname*{argmax}_y \sum_{a \in \mathcal{B}^{-1}(y)} p(a|x). \tag{7.5}$$

However, there are currently no algorithms that can solve the above efficiently, so it is typically approximated by finding the greedy best path

$$\operatorname*{argmax}_y p(y|x) = \operatorname*{argmax}_y \max_{a \in \mathcal{B}^{-1}(y)} p(a|x) = \mathcal{B}(\operatorname*{argmax}_a p(a|x)). \tag{7.6}$$

In other words, we simply find the label that achieves the highest probability at every time point and remove the duplicates and blank symbols to produce the final decoded sequence.

7.1.1 Connection to the marginal log loss

As a result of (7.6), the search space of CTC has an edge for every label in the label set (including the blank label) at every time step. Specifically, the search space G includes the edges $\{e_{\ell,t} : \ell \in L, t \in \{1, \ldots, T\}\}$ with $v_{t-1} = \text{tail}(e_{\ell,t})$ and $v_t = \text{head}(e_{\ell,t})$. An example is shown in Figure 7.1. The weight of an edge $e_{\ell,t}$ is the log probability of label ℓ at time t. By construction, the decision made at every time point is independent of the decision at other time points conditioned on $x_{1:T}$. In addition, since the probabilities at every time point sum to one, the partition function $Z(x)$ of the search space, i.e., the sum of all the path probabilities, is always 1.

Recall that the marginal log loss is defined as

$$\mathcal{L}_{\text{mll}} = -\log \sum_{p \in \mathcal{P}_{G \circ_\sigma F_y}} \exp(w(p)) + \log \sum_{p \in \mathcal{P}_G} \exp(w(p)) \tag{7.7}$$

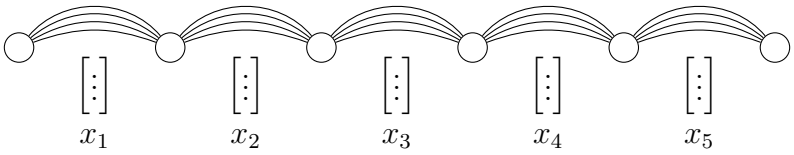


Figure 7.1: An example of the CTC search space for a five-frame utterance with a label set of size three (plus one blank).

where $\mathcal{P}_K$ is the set of paths in the FST K and F_y is the constraint FST constructed from the ground truth label sequence y. Since $Z(x) = 1$ for CTC, the second term is zero. By the definition of $\mathcal{B}$, we construct the constraint FST F_y such that it consists of the sequences of one or more labels with zero or more blanks in between labels. For example, for the label sequence "`k ae t`," the constraint FST is the regular expression $\varnothing^*\texttt{k}^+\varnothing^*\texttt{ae}^+\varnothing^*\texttt{t}^+\varnothing^*$. We have $\mathcal{B}^{-1}(y) = G \circ_\sigma F_y$. In other words, the sum of path probabilities in $G \circ_\sigma F_y$ exactly matches the CTC objective. Because the first term matches the CTC objective and the second term is zero, marginal log loss becomes the CTC objective for this type of search spaces and the log probability weight function.

7.2 Other Recent End-to-End Models

Most mainstream end-to-end speech recognition models can be broadly categorized as either frame-based models or encoder-decoder models. Frame-based LSTMs trained with CTC, HMMs, and some newer approaches like the auto-segmentation criterion (ASG) (Collobert et al., 2016) fall under the first category, because these models emit one symbol for every frame. Falling under the second category, encoder-decoder models proposed by (Chorowski et al., 2015; Bahdanau et al., 2016; Chan et al., 2016) generate labels one at a time while conditioning on the input and the labels generated in the past, without an explicit alignment between labels and frames. Since frame-based models follow the same graph search framework as segmental models, we will focus on discussing the connection between these and segmental models.

Recall that marginal log loss requires a search space G and a constraint FST F to limit the search space to ground-truth labels. To compute marginal log loss, we first compute the marginals on G for computing the partition function $Z(x)$, and then compute the marginals on the σ-composed FST $G \circ_\sigma F$ for computing $Z(x, y)$. CTC, HMMs whtn trained with lattice-free MMI (Povey et al., 2016), and ASG can all be seen as special cases of this framework.

Comparing CTC to HMMs, the search space is different depending on the HMM topology. For example, two-state HMMs are used in (Povey et al., 2016). Since the transition probabilities and emission probabilities are all locally normalized, the partition function $Z(x)$ is always 1. The constraint FST representing the ground-truth labels consists simply of sequences of repeating labels. For example, for the label sequence "`k ae t`," the constraint FST

for one-state HMMs is the regular expression $\texttt{k}^{+}\texttt{ae}^{+}\texttt{t}^{+}$. For two-state HMMs, the constraint FST is the regular expression $\texttt{k}_1\texttt{k}_2^{*}\texttt{ae}_1\texttt{ae}_2^{*}\texttt{t}_1\texttt{t}_2^{*}$. With the above construction, marginal log loss applied to HMMs is equivalent to lattice-free MMI (Povey et al., 2016).

For ASG, the search space is equivalent to that of one-state HMMs. Instead of assuming conditional independence as in CTC, ASG includes transition probabilities between states. The constraint FST is identical to that of HMMs, with repeated ground-truth labels. However, in ASG the weights on the edges are not locally normalized, so the partition function $Z(x)$ is not always 1 and has to be computed. With the above search space construction, marginal log loss becomes ASG.

Another approach similar to CTC proposed in (Graves, 2012) is called RNN transducers. The search space of an RNN transducer is the set of alignments from the speech signal to all possible label sequences, so the search space grows exponentially in the number of labels. The weight function of a path in this approach relies on an RNN, and is not decomposable as a sum of weights of the edges. RNN transducers are trained with marginal log loss. By the independence assumption imposed in (Graves, 2012), the partition function $Z(x)$ is still 1, so we do not need to marginalize over the exponentially large space. During decoding, however, we still have to search over the exponentially large space with, for example, beam search.

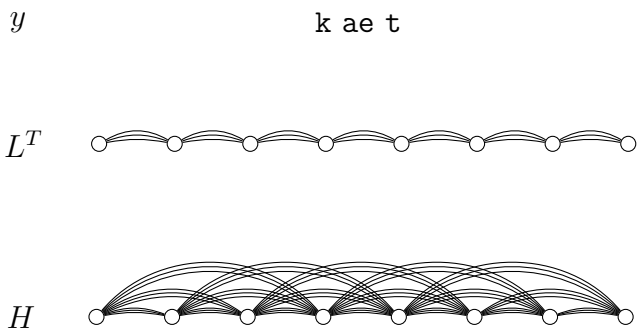
In view of this framework, even when using the same loss function, i.e., marginal log loss, segmental models and frame-based models differ in their search space, weight functions, and how the search space is constrained by the ground-truth labels during training. A summary of special cases is shown in Table 7.1.

7.3 TIMIT Experiments

We compare various frame-based models trained end to end on the same phonetic recognition task in the same setting as in Section 5.2. We use 3-layer 256-unit biderectional LSTMs paired with CTC, one-state HMM, and two-state HMMs. We do not use transition probabilities for two-state HMMs. We optimize marginal log loss with the corresponding weight functions and the corresponding constraint FSTs for CTC, one-state HMM, and two-state HMMs. We run vanilla SGD with step size 0.1 and gradient clipping of norm 5 for 20 epochs. Starting from the best performing model of the first 20 epochs, we run vanilla SGD for another 20 epochs with step size 0.75 decayed by 0.75 after each epoch. The dropout rate is 0.2, and no other explicit regularizer is used except early stopping on the development set. The same experiments are repeated with pyramid LSTMs. Results comparing with segmental models are shown in Table 7.2.

When standard LSTMs are used, we fail to minimize the training loss for one-state HMMs and two-state HMMs The only difference between CTC and one-state HMMs is the use of blank symbols, suggesting that blank symbols play an important role in end-to-end frame-based models. In particular, one-state HMMs and two-state HMMs explicitly marginalize over segmentations in training and are thus sensitive to time, while LSTMs trained with CTC are not. Besides, the tail states in L_2 can be seen as label-dependent blank symbols.

Table 7.1: An example instantiation of the components used in marginal log loss with the ground-truth sequence "k ae t" and T input frames for segmental models and various end-to-end models. The search space L^T consists of sequences of T labels. The label sets L_1 and L_2 contain labels in L with subscript 1 and 2 respectively. The search space is denoted G and the constraint FST is denoted F_y in the table.



	G	F_y	weight
segmental models	H	k ae t	FC, FCB, MLP
1-state HMMs	L^T	$\mathtt{k}^+\mathtt{ae}^+\mathtt{t}^+$	posteriors
2-state HMMs (Povey et al., 2016)	$(L_1 \cup L_2)^T$	$\mathtt{k}_1\mathtt{k}_2^*\mathtt{ae}_1\mathtt{ae}_2^*\mathtt{t}_1\mathtt{t}_2^*$	posteriors +transition
CTC (Graves et al., 2006)	$(L \cup \{\varnothing\})^T$	$\varnothing^*\mathtt{k}^+\varnothing^*\mathtt{ae}^+\varnothing^*\mathtt{t}^+\varnothing^*$	posteriors
ASG (Collobert et al., 2016)	$(L \times L)^T$	$\mathtt{k}^+\mathtt{ae}^+\mathtt{t}^+$	posteriors +transition
Gram-CTC (Liu et al., 2017)	$(L^n \cup \{\varnothing\})^{T-n+1}$	n-grams of k ae t	posteriors

Table 7.2: Phoneme error rates (%) for end-to-end training from random initialization comparing CTC and segmental models trained with marginal log loss.

	regular LSMT		pyramid LSTM	
	dev	test	dev	test
CTC	17.4		16.6	18.7
1-state HMM	82.2		17.5	
2-state HMM	78.1		20.6	
seg FC	16.7	19.6	17.9	
seg FCB	17.0		17.7	
seg MLP			17.1	19.2

Since two-state HMMs fail to achieve a low training loss, having label-dependent blanks is not helpful in this case.

When pyramid LSTMs are used, all variants of frame-based models achieve low PERs on the development set and improve over ones with standard LSTMs. Since the time resolution is reduced by four with pyramid LSTMs, we hypothesize that using the pyramid LSTMs introduces a bias similar to a minimum duration constraint, which helps end-to-end training for frame-based models.

The number of minutes per epoch for training segmental models and LSTMs trained with CTC is shown in Table 7.3, and a scatter plot showing number of minutes per epoch vs. PER is shown in Figure 7.2. Pyramid LSTMs trained with CTC are the best performer while also being the most efficient to train. The real-time factors for decoding are shown in Table 7.4. Due to smaller and simpler search spaces, frame-based LSTMs trained with CTC decode faster than segmental models. While frame-based LSTMs train and decode faster, segmental models are more flexible, so they might be improved with additional feature functions. Explicitly hypothesizing segments can serve as a prior, so segmental models might perform better in low-resource settings.

7.4 ASL Experiments

For American Sign Language (ASL) fingerspelling experiments, we follow (Kim et al., 2017) and compare end-to-end models in both signer-dependent setting and signer-independent setting. The input is a sequence of 128-dimensional histograms of oriented gradients (HoG) feature vectors computed over 128×128 hand shape images. The output is a sequence of English letters, and the label set consists of the 26 English characters plus two labels for initial and final non-signing regions. The evaluation metric is the edit distance between the predicted letter sequence and the ground-truth letter sequence.

For the signer-dependent setting, the data for each signer is split into ten folds, in which

Table 7.3: Average number of minutes for one epoch of end-to-end training on TIMIT.

		regular LSTM	pyramid LSTM
CTC		104	44
seg FC	hinge	119	38
	log	177	44
	MLL	260	47
seg MLP	hinge		83
	log		110
	MLL		116

Table 7.4: Real-time factors for decoding comparing CTC and segmental models.

	regular LSTM	pyramid LSTM
CTC	0.12	0.07
seg FC	0.38	0.12
seg MLP		0.59

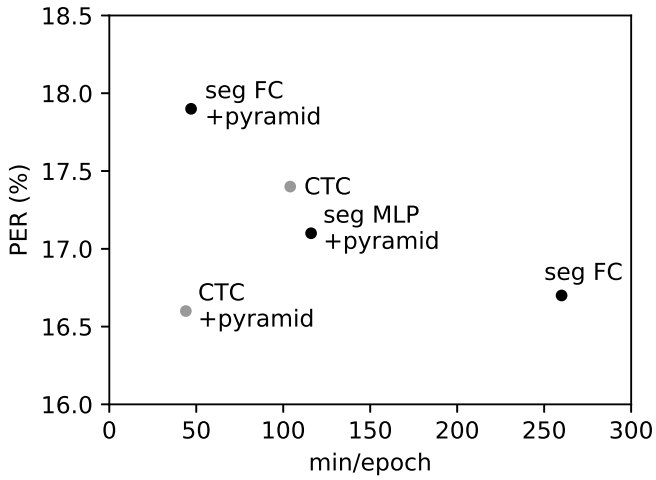


Figure 7.2: Training minutes per epoch vs. PER (%) for segmental models and LSTMs trained with CTC.

Table 7.5: Letter error rates (%) for signer-dependent models averaged over ten folds.

	Andy		Drucie		Rita		Robin		Avg	
	dev	test	dev	test	dev	test	dev	test	dev	test
(Kim et al., 2017)										
Tandem HMM		13.8		7.1		26.1		11.5		14.6
two-stage seg FC		8.1		7.7		9.3		10.1		8.8
DSC		7.2		6.5		8.1		8.6		7.6
CTC	6.1	7.8	7.2	8.4	11.0	15.0	10.8	12.6	8.8	11.0
seg FCB	5.1	7.8	5.6	8.2	8.3	11.8	7.5	9.1	6.6	9.2

eight are for training, one is for development, and one is for testing. We report the letter error rate averaged over the 10 experiments. We train LSTMs with CTC and segmental models with the FCB weight function. Both models are trained with marginal log loss from random initialization. The encoder is a one-layer 128-unit bidirectional LSTM. For CTC, we run vanilla SGD with step sizes in $\{0.1, 0.05, 0.025\}$ and gradient clipping with norm 5 for 200 epochs. Dropout is used with a rate of 0.2, and no other explicit regularizer is used except early stopping. Step sizes and early stopping are tuned on the development set for each individual experiment. For segmental models, we run vanilla SGD with step size 0.1 for 75 epochs. The maximum segment length is 75 frames. Other hyperparameters, such as dropout and gradient clipping, are the same as for CTC. Results are shown in Table 7.5. The frame-based LSTMs trained with CTC are better than Tandem HMMs, but are behind the segmental models. The segmental models trained end to end are on par with the two-stage system in (Kim et al., 2017), but are behind the discriminative segmental cascades (DSC).

For the signer-independent setting, we train on three signers and use the last signer for development and testing. Specifically, the data of the last signer is split into ten folds, and we use two folds for development and the rest of the eight folds for testing. Instead of dividing the accumulated error counts by the accumulated label sequence lengths, letter error rates are averaged over the folds. We compare LSTMs trained with CTC and segmental models with FCB weight function. The loss function and training procedure stay the same except that we only run 40 epochs of vanilla SGD for segmental models. Results are shown in Table 7.6. In this setting, CTC and segmental models trained end to end perform significantly better than Tandem HMMs and segmental models trained with the FC weight function in two stages.

To compare signer-independent and signer-dependent settings, we average the letter error rates over the same eight folds. Results are shown in Table 7.7. Segmental models perform significantly better than CTC in both settings.

Table 7.6: Letter error rates (%) for signer-independent models averaged over eight folds.

	Andy		Drucie		Rita		Robin		Avg	
	dev	test	dev	test	dev	test	dev	test	dev	test
(Kim et al., 2017)										
Tandem HMM		54.1		54.7		62.6		57.5		57.2
two-stage seg FC		55.3		53.3		72.5		61.4		60.6
CTC	47.2	50.2	55.2	54.2	50.1	49.3	55.7	54.4	52.1	52.0
seg FCB	44.5	43.2	41.8	43.5	40.6	44.7	51.0	48.7	44.5	45.0

Table 7.7: Letter error rates (%) for signer-dependent and signer-independent models averaged over the same eight folds.

	Andy	Drucie	Rita	Robin	Avg
CTC	8.1	8.2	14.5	12.1	10.7
seg FCB	7.9	8.3	12.0	8.2	9.1
CTC	50.2	54.2	49.3	54.4	52.0
seg FCB	43.2	43.5	44.7	48.7	45.0

7.5 Summary

We have discussed how other end-to-end frame-based models, such as CTC, HMMs trained with lattice-free MMI, ASG, and RNN transducers are all trained with marginal log loss. The differences among them lie in the search spaces and the weight functions. Drawing these connections allows us to generalize search spaces, loss functions, and weight functions to a broad class of models.

From the results of comparing CTC to one-state HMMs and two-state HMMs, we have found that the blank symbol seems to play an important role in training LSTMs end to end. Using pyramid LSTMs improves both the performance and the runtime of decoding and training for CTC, one-state HMMs, and two-state HMMs, but not for segmental models. We have also shown that segmental models with regular LSTMs are better than regular LSTMs trained with CTC on both phonetic recognition and ASL fingerspelling recognition.

Chapter 8

Conclusion and Future Work

In this thesis, we have made the following contributions in advancing the study of discriminative segmental models.

- We have proposed discriminative segmental cascades for incorporating rich computationally expensive features while maintaining efficiency. We use max-marginal pruning to reduce the size of search spaces, generating sparse lattices while having low oracle error rates. We obtain improved performance over most earlier work while greatly improving efficiency.
- We have explored the space of losses, multi-stage training, and end-to-end training for segmental models with various losses and weight functions. We have shown that segmental models trained with multi-stage training can serve as a good initialization for end-to-end training.
- We have presented a unified framework including many end-to-end models, such as hidden Markov models, connectionist temporal classification, and segmental models, as special cases. Drawing this connection allows us to design general search spaces, loss functions, and weight functions applicable to models in a broad class.
- We achieve competitive results on phonetic recognition and ASL fingerspelling recognition with a segmental model trained end to end. Earlier work (not reported in this thesis) by (Kim et al., 2017) has obtained the best reported results to date using our discriminative segmental cascades on signer-dependent fingerspelling recognition.

In this chapter, we discuss some potential future work extending segmental models to large-vocabulary tasks and to unsupervised settings. Another potential direction for future work is even richer feature functions.

8.1 Word Recognition

Since first-pass segmental models are computationally demanding, it is very slow to train and decode segmental models with label sets of size in the order of 10,000. This poses a

challenge for tasks with large label sets, such as word recognition. Zweig et al. (2010); Maas et al. (2015) bypassed this difficulty by using a baseline HMM recognizer to generate word lattices, and used segmental models to rescore these lattices, exploring various word-level features. However, the performance is constrained by the quality of the baseline recognizer and the quality of the lattices.

First-pass segmental models have previously been successfully applied to word recognition (Glass, 2003; He and Fosler-Lussier, 2015). This previous work treats first-pass segmental models as a drop-in replacement for HMM phoneme recognizers, because both models serve as functions that map acoustic features to phoneme strings. The phoneme recognizers are then composed with a lexicon and a language model to form a word recognizer, as described in Section 2.2. This is also an option for extending our work to word recognition.

Recent work has explored models that directly predict characters, avoiding the need for a lexicon (Graves and Jaitly, 2014; Miao et al., 2015) but still allowing for improved performance when constraining the search space with a lexicon (through FST composition) (Miao et al., 2015). Segmental models can also be used to predict characters by changing the label set, but might be hampered by the poor alignment of characters to acoustics. Syllables might be a better option than characters.

Instead of using intermediate discrete representations, such as phonemes or characters, recent advances in computing power have made it feasible to directly predict words (Maas et al., 2012; Bengio and Heigold, 2014; Soltau et al., 2016; Audhkhasi et al., 2017). In this case, rather than using a pronunciation dictionary, only a list of words is needed for decoding. Segmental models can also be used to directly predict words by using the list of words as the label set. This approach is worth exploring further, although efficiency issues make it nontrivial to train such models (Audhkhasi et al., 2017).

8.2 Unsupervised Sequence Prediction

We have presented segmental models for supervised sequence prediction in this thesis. Our framework can be extended to unsupervised sequence prediction. The goal in this setting is typically grouping segments of varying length into clusters. We define unsupervised sequence prediction as finding a function that maps an input sequence $x_1, \dots, x_T$ into a sequence of segments $(\ell_1, s_1, t_1), \dots, (\ell_n, s_n, t_n)$ where $\ell_i \in L$ for $i = 1, \dots, n$ and some label set L, and $s_1 = 1$, $t_n = T + 1$, $s_i < t_i$, $t_{i-1} = s_i$ for $i = 1, \dots, n$, given a data set S of sequences without labels. Since we do not have labels for the data samples, the goal in general is to design a loss function $\mathcal{L}(\Theta; x)$ that only depends on the input sequence x, or even more generally a loss function $\mathcal{L}(\Theta; S)$ that depends on the data set S. The label set L is typically predefined to be just a set of identifiers, and the correspondence between the input sequence and the identifier sequence is learned from a data set. As a result, a label in L does not have a predefined meaning unless the user assigns a post hoc meaning or the loss function enforces one.

Generative segmental models, such as hidden semi-Markov models, can be naturally extended to the unsupervised setting by maximizing the marginal likelihood of the data.

For example, Bayesian segmental models have been applied to small-vocabulary (Kamper et al., 2015) and larger-vocabulary (Kamper et al., 2017a) word discovery. Viterbi-style training for such models has also been explored (Kamper et al., 2017b). Recently, Tran et al. (2016) proposed unsupervised neural HMMs, which can be easily extended to hidden semi-Markov models. In fact, Dai et al. (2017) proposed a neural version of hidden semi-Markov models similar to Tran et al. (2016). All the above approaches aim to maximize the marginal likelihood of the data.

Besides generative models, approaches for training log-linear models in an unsupervised fashion, notably contrastive estimation (Smith and Eisner, 2005; Poon et al., 2009), noise-contrastive estimation (Gutmann and Hyvärinen, 2010), and auto-encoders (Ammar et al., 2014), have also been extensively studied. These approaches are designed for discriminative models, and can be readily applied to our segmental models.

A segmental model trained in an unsupervised fashion can be used for segment clustering. One major application of segment clustering is for lexical unit discovery. There is a long history of lexical unit discovery from acoustic signals (Brent, 1999), sometimes called spoken term discovery (Park and Glass, 2008; Jansen et al., 2010). Much of the recent work relies on dynamic time warping (DTW) (Jansen and Van Durme, 2011; Carlin et al., 2011) for spoken term discovery, with Lee et al. (2015) being out of the few exceptions who took a nonparametric Bayesian approach. DTW has also been found to help lexical discovery in an HMM system (Walter et al., 2013). Other related tasks that we do not cover here include unsupervised word segmentation (Goldwater et al., 2006), unsupervised part-of-speech tagging (Merialdo, 1994), and unsupervised dependency parsing (Klein and Manning, 2004). Though segmental models were not used in the above studies, many ideas can be borrowed and help advance the study of segmental models in unsupervised settings.

The assumption that input sequences can be decomposed into a sequence of segments serves as a strong inductive bias, and it is particularly useful in unsupervised settings when little is assumed about the data. Segmental models are more flexible than frame-based models when additional assumptions, such as the form of the feature functions, are needed. Therefore, we believe segmental models have the potential to perform well in these unsupervised settings,

Bibliography

Ossama Abdel-Hamid, Li Deng, Dong Yu, and Hui Jiang. Deep segmental neural networks for speech recognition. In *Annual Conference of International Speech Communication Association*, 2013.

Cyril Allauzen and Mehryar Mohri. N-way composition of weighted finite-state transducers. *International Journal of Foundations of Computer Science*, 20, 2009.

Cyril Allauzen, Mehryar Mohri, and Brian Roark. Generalized algorithms for constructing statistical language models. In *Annual Meeting on Association for Computational Linguistics*, 2003.

Waleed Ammar, Chris Dyer, and Noah A Smith. Conditional random field autoencoders for unsupervised structured prediction. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Anastasios Anastasakos, Richard Schwartz, and Han Shu. Duration modeling in large vocabulary speech recognition. 1995.

Joakim Andén and Stéphane Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62, 2014.

Kartik Audhkhasi, Bhuvana Ramabhadran, George Saon, Michael Picheny, and David Nahamoo. Direct acoustics-to-word models for english conversational speech recognition. *CoRR*, abs/1703.07754, 2017.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

Lalit R. Bahl, Peter F. Brown, Peter V. de Souza, and Robert L. Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1986.

Samy Bengio and Georg Heigold. Word embeddings for speech recognition. In *INTERSPEECH*, 2014.

Dimitri P Bertsekas, Angelia Nedi, and Asuman E Ozdaglar. Convex analysis and optimization. 2003.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Michael R Brent. An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning*, 34, 1999.

Michael A Carlin, Samuel Thomas, Aren Jansen, and Hynek Hermansky. Rapid evaluation of speech representations for spoken term discovery. In *Annual Conference of the International Speech Communication Association*, 2011.

William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

Hung-An Chang and James Glass. Hierarchical large-margin Gaussian mixture models for phonetic classification. In *IEEE Workshop on Automatic Speech Recognition & Understanding*, 2007.

Jane W Chang and James R Glass. Segmentation and modeling in segment-based recognition. In *European Conference on Speech Communication and Technology*, 1997.

David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Annual Meeting on Association for Computational Linguistics*, 2005.

David Chiang. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13, 2012.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

Philip Clarkson and Pedro Moreno. On the use of support vector machines for phonetic classification. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1999.

Ronald A. Cole, Richard M. Stern, Michael S. Phillips, Scott M. Brill, Andrew P. Pilant, and Philippe Specker. Feature-based speaker-independent recognition of isolated english letters. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1983.

Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Trading convexity for scalability. In *International Conference on Machine learning*, 2006.

Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. Wav2Letter: an end-to-end ConvNet-based speech recognition system. *CoRR*, abs/1609.03193, 2016.

Hanjun Dai, Bo Dai, Yan-Ming Zhang, Shuang Li, and Le Song. Recurrent hidden semi-Markov model. In *International Conference on Representation Learning (ICLR)*, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2011.

John D. Ferguson. Variable duration models for speech recognition. 1980.

Mark Gales and Steve Young. Segmental HMMs for speech recognition. In *Eurospeech*, 1993a.

Mark Gales and Steve Young. The theory of segmental hidden markov models. 1993b.

John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, David S Pallett, Nancy L Dahlgren, and Victor Zue. TIMIT acoustic-phonetic continuous speech corpus. *Linguistic data consortium*, 10(5):0, 1993.

Kevin Gimpel and Noah A Smith. Softmax-margin CRFs: Training log-linear models with cost functions. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

Kevin Gimpel and Noah A Smith. Structured ramp loss minimization for machine translation. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2012.

James Glass, Jane Chang, and Michael McCandless. A probabilistic framework for feature-based speech recognition. In *International Conference on Spoken Language Processing (ICSLP)*, 1996.

James R. Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech & Language*, 17, 2003.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTAT)*, 2010.

Sharon Goldwater, Thomas L Griffiths, and Mark Johnson. Contextual dependencies in unsupervised word segmentation. In *Annual Meeting of the Association for Computational Linguistics*, 2006.

Alex Graves. Sequence transduction with recurrent neural networks. *CoRR*, abs/1211.3711, 2012.

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

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18, 2005.

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

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, speech and signal processing (ICASSP)*, 2013.

Asela Gunawardana, Milind Mahajan, Alex Acero, and John C Platt. Hidden conditional random fields for phone classification. In *Interspeech*, 2005.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics (AISTAT)*, 2010.

Andrew Halberstadt. *Heterogeneous Acoustic Measurements and Multiple Classifiers for Speech Recognition*. PhD thesis, Massachusetts Institute of Technology, 1998.

Andrew Halberstadt and James Glass. Heterogeneous measurements and multiple classifiers for speech recognition. In *International Conference on Spoken Language Processing*, 1998.

Mark Hasegawa-Johnson, James Baker, Steven Greenberg, Katrin Kirchhoff, Jennifer Muller, Kemal Soömez, Sarah Borys, Ken Chen, Amit Juneja, Karen Livescu, Srividya Mohan, Emily Coogan, and Tianyu Wang. Landmark-based speech recogntion: Report of the 2004 Johns Hopkins Summer Workshop. 2005.

Yanzhang He. *Segmental Models with an exploration of acoustic and lexical grouping in automatic speech recognition*. PhD thesis, The Ohio State University, 2015.

Yanzhang He and Eric Fosler-Lussier. Efficient segmental conditional random fields for phone recognition. In *Annual Conference of the International Speech Communication Association*, 2012.

Yanzhang He and Eric Fosler-Lussier. Segmental conditional random fields with deep neural networks as acoustic models for first-pass word recognition. In *INTERSPEECH*, 2015.

Georg Heigold, Wolfgang Macherey, Ralf Schluter, and Hermann Ney. Minimum exact word error training. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2005.

Georg Heigold, Thomas Deselaers, Ralf Schlüter, and Hermann Ney. Modified MMI/MPE: A direct evaluation of the margin in speech recognition. In *International Conference on Machine learning*, 2008.

James Hillenbrand, Laura A. Getty, Michael J. Clark, and Kimberlee Wheeler. Acoustic characteristics of American English vowels. *The Journal of the Acoustical Society of America*, 1995.

Hans-Günter Hirsch and David Pearce. The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *ISCA Workshop ASR2000*, 2000.

Minh Hoai, Zhen-Zhong Lan, and Fernando De la Torre. Joint segmentation and classification of human actions in video. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9, 1997.

John N. Holmes, Wendy J. Holmes, and Philip N. Garner. Using formant frequencies in speech recognition. In *Eurospeech*, 1997.

Navdeep Jaitly, Vincent Vanhoucke, and Geoffrey Hinton. Autoregressive product of multi-frame predictions can improve the accuracy of hybrid models. In *Annual Conference of the International Speech Communication Association*, 2014.

Aren Jansen and Benjamin Van Durme. Efficient spoken term discovery using randomized algorithms. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.

Aren Jansen, Kenneth Church, and Hynek Hermansky. Towards spoken term discovery at scale with zero resources. In *Annual Conference of the International Speech Communication Association*, 2010.

Frederick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

Frederick Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, 1998.

Bing-Hwang Juang and Lawrence R Rabiner. The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38, 1990.

Herman Kamper, Aren Jansen, and Sharon Goldwater. Fully unsupervised small-vocabulary speech recognition using a segmental Bayesian model. In *INTERSPEECH*, 2015.

Herman Kamper, Aren Jansen, and Sharon Goldwater. A segmental framework for fully-unsupervised large-vocabulary speech recognition. 46, 2017a.

Herman Kamper, Karen Livescu, and Sharon Goldwater. An embedded segmental k-means model for unsupervised segmentation and clustering of speech. abs/1703.08135, 2017b.

Danny Karmon and Joseph Keshet. Risk minimization in structured prediction using orbit loss. *CoRR*, abs/1512.02033, 2015.

Patrick Kenny, Rene Hollan, Vishwa N Gupta, Matthew Lennig, Paul Mermelstein, and Douglas O'Shaughnessy. A*-admissible heuristics for rapid lexical access. *IEEE Transactions on Speech and Audio Processing*, 1993.

Joseph Keshet and David McAllester. Generalization bounds and consistency for latent structural probit and ramp loss. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, and Dan Chazan. A large margin algorithm for speech-to-phoneme and music-to-score alignment. *IEEE Transactions on Audio, Speech, and Language Processing*, 15, 2007.

Taehwan Kim, Greg Shakhnarovich, and Karen Livescu. Fingerspelling recognition with semi-markov conditional random fields. In *International Conference on Computer Vision*, 2013.

Taehwan Kim, Jonathan Keane, Weiran Wang, Hao Tang, Jason Riggle, Gregory Shakhnarovich, Diane Brentari, and Karen Livescu. Lexicon-free fingerspelling recognition from video: Data, models, and signer adaptation. *Computer Speech & Language*, 2017.

Dan Klein and Christopher D Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Annual Meeting on Association for Computational Linguistics*, 2004.

Lingpeng Kong, Chris Dyer, and Noah A Smith. Segmental recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

Gary E. Kopec and Marcia A. Bush. Network-based isolated digit recognition using vector quantization. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33, 1985.

Peter Ladefoged. *A Course in Phonetics*. Cengage Learning, 2005.

Yann LeCun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In *AISTATS*, 2005.

Chia-ying Lee, Timothy J O'Donnell, and James Glass. Unsupervised lexicon discovery from acoustic input. *Transactions of the Association for Computational Linguistics*, 3:389–403, 2015.

Kai-Fu Lee and Hsiao-Wuen Hon. Speaker-independent phone recognition using hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37, 1989.

Christopher J Leggetter and Philip C Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech & Language*, 9, 1995.

Stephen E. Levinson. Continuously variable duration hidden markov models for automatic speech recognition. *Computer Speech & Language*, 1, 1986.

Hairong Liu, Zhenyao Zhu, Xiangang Li, and Sanjeev Satheesh. Gram-CTC: Automatic unit selection and target decomposition for sequence labelling. *CoRR*, abs/1703.00096, 2017.

Karen Livescu. *Feature-based pronunciation modeling for automatic speech recognition*. PhD thesis, Massachusetts Institute of Technology, 2005.

Bruce Lowerre. *The Harpy Speech Recognition System*. PhD thesis, Carnegie Mellon University, 1976.

Liang Lu, Lingpeng Kong, Chris Dyer, Noah A. Smith, and Steve Renals. Segmental recurrent neural networks for end-to-end speech recognition. In *Annual Conference of the International Speech Communication Association*, 2016.

Liang Lu, Lingpeng Kong, Chris Dyer, and Noah A Smith. Multi-task learning with CTC and segmental CRF for speech recognition. *CoRR*, abs/1702.06378, 2017.

Andrew L Maas, Stephen D Miller, Tyler M O'neil, Andrew Y Ng, and Patrick Nguyen. Word-level acoustic modeling with convolutional vector regression. In *ICML Workshop on Representation Learning*, 2012.

Andrew L Maas, Ziang Xie, Dan Jurafsky, and Andrew Y Ng. Lexicon-free conversational speech recognition with neural networks. In *Human Language Technologies: The Annual Conference of the North American Chapter of the ACL*, 2015.

Erik McDermott and Atsushi Nakamura. Flexible discriminative training based on equal error group scores obtained from an error-indexed forward-backward algorithm. In *Annual Conference of the International Speech Communication Association*, 2008.

Erik McDermott, Shinji Watanabe, and Atsushi Nakamura. Margin-space integration of MPE loss via differencing of MMI functionals for generalized error-weighted discriminative training. In *Annual Conference of the International Speech Communication Association*, 2009.

Bernard Merialdo. Tagging english text with a probabilistic model. *Computational Linguistics*, 20, 1994.

Yajie Miao, Mohammad Gowayyed, and Florian Metze. EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2015.

George A Miller and Patricia E Nicely. An analysis of perceptual confusions among some English consonants. *The Journal of the Acoustical Society of America*, 27, 1955.

Abdel-rahman Mohamed, George Dahl, and Geoffrey Hinton. Deep belief networks for phone recognition. In *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7, 2002.

Mehryar Mohri. Weighted automata algorithms. *Handbook of weighted automata*, 2009.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. In *European Conference on Artifitial Intelligence*, 1996.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16, 2002.

Jeremy Morris and Eric Fosler-Lussier. Crandem: Conditional random fields for word recognition. In *Annual Conference of the International Speech Communication Association*, 2009.

Mahesh Chandra Mukkamala and Matthias Hein. Variants of RMSProp and adagrad with logarithmic regret bounds. In *International Conference on Machine Learning (ICML)*, 2017.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

Daisuke Okanohara, Yusuke Miyao, Yoshimasa Tsuruoka, and Jun'ichi Tsujii. Improving the scalability of semi-Markov conditional random fields for named entity recognition. In *Annual meeting of the Association for Computational Linguistics (ACL)*, 2006.

Mari Ostendorf and Salim Roukos. A stochastic segment model for phoneme-based continuous speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37, 1989.

Mari Ostendorf, Vassilios V. Digalakis, and Owen A. Kimball. From HMM's to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on Speech and Audio processing*, 4, 1996.

Alex S Park and James R Glass. Unsupervised pattern discovery in speech. *IEEE Transactions on Audio, Speech, and Language Processing*, 16, 2008.

Gordon E. Peterson and Ilse Lehiste. Duration of syllable nuclei in English. *The Journal of The Acoustical Society of America*, 32, 1960.

Hoifung Poon, Colin Cherry, and Kristina Toutanova. Unsupervised morphological segmentation with log-linear models. In *Human Language Technologies: Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

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

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

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The Kaldi speech recognition toolkit. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.

Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahrmani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *INTERSPEECH*, 2016.

Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77, 1989.

Asaf Rendel, Alexander Sorin, Ron Hoory, and Andrew Breen. Towards automatic phonetic segmentation for TTS. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE International Conference on Neural Networks*, 1993.

Martin Russell. A segmental HMM for speech pattern modelling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1993.

Martin Russell and Roger Moore. Explicit modelling of state occupancy in hidden Markov models for automatic speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1985.

Martin J. Russell and Anneliese E. Cook. Experimental evaluation of duration modelling techniques for automatic speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1987.

Sunita Sarawagi and William W. Cohen. Semi-Markov conditional random fields for information extraction. In *Advances in neural information processing systems*, 2005.

Rich Schwartz, Y Chow, S Roucos, M Krasner, and J Makhoul. Improved hidden Markov modeling of phonemes for continuous speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 9, 1984.

Richard Schwartz and John Makhoul. Where the phonemes are: dealing with ambiguity in acoustic-phonetic recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23, 1975.

Matt Shannon. Optimizing expected word error rate via sampling for speech recognition. In *Interspeech*, 2017.

Tomas Simon, Minh Hoai Nguyen, Fernando De la Torre, and Jeffrey F. Cohn. Action unit detection with segment-based svms. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Achim Sixtus and Stefan Ortmanns. High quality word graphs using forward-backward pruning. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1999.

David A Smith and Jason Eisner. Minimum risk annealing for training log-linear models. In *Annual Meeting on Association for Computational Linguistics*, 2006.

Noah A Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Annual Meeting on Association for Computational Linguistics*, 2005.

Hagen Soltau, Hank Liao, and Hasim Sak. Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition. *CoRR*, abs/1610.09975, 2016.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 2014.

Hao Tang, Kevin Gimpel, and Karen Livescu. A comparison of training approaches for discriminative segmental models. In *INTERSPEECH*, 2014.

Hao Tang, Weiran Wang, Kevin Gimpel, and Karen Livescu. Discriminative segmental cascades for feature-rich phone recognition. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2015.

Hao Tang, Weiran Wang, Kevin Gimpel, and Karen Livescu. End-to-end training approaches for discriminative segmental models. In *Spoken Language Technology (SLT)*, 2016.

Hao Tang, Liang Lu, Lingpeng Kong, Kevin Gimpel, Karen Livescu, Chris Dyer, Noah A. Smith, and Steve Renals. *IEEE Journal of Selected Topics in Signal Processing*, to appear, 2017.

László Tóth. Phone recognition with hierarchical convolutional deep maxout networks. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015(1):25, 2015.

Ke Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. Unsupervised neural hidden Markov models. *CoRR*, abs/1609.09007, 2016.

Noriko Umeda. Vowel duration in American English. *The Journal of the Acoustical Society of America*, 58, 1975.

Noriko Umeda. Consonant duration in American English. *The Journal of the Acoustical Society of America*, 61(3):846–858, 1977.

Rogier C van Dalen and Mark J Gales. Annotating large lattices with the exact word error. In *Interspeech*, 2015.

Vincent Vanhoucke, Matthieu Devin, and Georg Heigold. Multiframe deep neural networks for acoustic modeling. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

La The Vinh, Sungyoung Lee, and Young-Koo Lee. A fast implementation of semi-Markov conditional random fields. In *Signal Processing, Image Processing and Pattern Recognition*. 2011.

Oliver Walter, Timo Korthals, Reinhold Haeb-Umbach, and Bhiksha Raj. A hierarchical system for word discovery exploiting DTW-based initialization. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2013.

Clifford Weinstein, Stephanie S. McCandless, Lee Mondshein, and Victor Zue. A system for acoustic-phonetic analysis of continuous speech. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23, 1975.

David Weiss, Benjamin Sapp, and Ben Taskar. Structured prediction cascades. *CoRR*, abs/1208.3279, 2012.

S. J. Young, J. J. Odell, and P. C. Woodland. Tree-based state tying for high accuracy acoustic modelling. In *Workshop on Human Language Technology*, 1994.

Jiahong Yuan, Neville Ryant, Mark Liberman, Andreas Stolcke, Vikramjit Mitra, and Wen Wang. Automatic phonetic segmentation using boundary models. In *Annual Conference of the International Speech Communication Association*, 2013.

Alan L Yuille and Anand Rangarajan. The concave-convex procedure. *Neural computation*, 15, 2003.

Shi-Xiong Zhang and Mark JF Gales. Structured SVMs for automatic speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 21, 2013.

Victor Zue, James Glass, Michael Philips, and Stephanie Seneff. Acoustic segmentation and phonetic classification in the SUMMIT system. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1989.

Geoffrey Zweig. Classification and recognition with direct segment models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2012.

Geoffrey Zweig and Patrick Nguyen. A segmental CRF approach to large vocabulary continuous speech recognition. In *IEEE Workshop on Automatic Speech Recognition & Understanding*, 2009.

Geoffrey Zweig, Patrick Nguyen, Dirk Van Compernolle, Kris Demuynck, Les Atlas, Pascal Clark, Greg Sell, Fei Sha, Meihong Wang, Aren Jansen, Hynek Hermansky, Damianos Karakos, Keith Kintzley, Samuel Thomas, Sivaram GSVS, Sam Bowman, and Justine Kao. Speech recognition with segmental conditional random fields: Final report from the 2010 JHU Summer Workshop. 2010.

Geoffrey Zweig, Chengzhu Yu, Jasha Droppo, and Andreas Stolcke. Advances in all-neural speech recognition. In *IEEE Internation Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.