Practical and Efficient Incorporation of Syntactic Features into Statistical Language Models

by

Ariya Rastrow

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

May, 2012

Abstract

Automatic Speech Recognition (ASR) and Statistical Machine Translation (SMT), among other natural language processing applications, rely on a language model (LM) to provide a strong linguistic prior over word sequences of the often prohibitively large and complex hypothesis space of these systems. The language models deployed in most state-of-the-art ASR and SMT systems are n-gram models. Several statistical frameworks have been proposed to build more complex models and "put (the syntactic structure of) language back into language modeling." Yet, n-gram models, despite being *linguistically naïve*, are still favored, because estimating them from text is well understood, they are computationally efficient, and integrating them into ASR and SMT systems is straightforward. This dissertation proposes novel algorithms and techniques that make it practical to estimate and apply more complex language models in ASR and SMT tasks, in particular syntactic modes for speech recognition.

While yielding significantly better performance than n-gram models, the syntactic structured language models (SLM) can not be efficiently trained on a large amount of text data due to the impractical size of the resulting model. A general information-

theoretic pruning scheme is proposed to significantly reduce the size of the SLM while maintaining its prediction accuracy, therefore enabling efficient maximum likelihood estimation of the SLM parameters.

The SLM, and other long-span language models, can not be directly applied during decoding or word lattice rescoring. Instead, these models are limited to an N-best rescoring framework, which as a search algorithm suffers from several known deficiencies and inefficiencies. Leveraging the theory and efficient algorithms for finite-state automata (FSA), an effective hill-climbing algorithm is developed for rescoring ASR lattices using long-span language models. It is shown that integrating the SLM into an ASR system in this manner significantly improves the WER over the computationally comparable N-best rescoring technique.

Discriminative training of language models with long-span features, such as syntactic dependencies, triggers, and topic information, is limited to N-best lists for similar reasons. The FSA based hill climbing algorithm, proposed for the application of long-span models to ASR lattice rescoring, also paves the way for efficient discriminative training of long-span language models on speech lattices and thus, alleviates the shortcomings of N-best training.

Long span language models—regardless of whether they are trained via maximum likelihood or trained discriminatively, and during both training and application—rely on auxiliary tools to extract nonlocal features. In the case of the SLM the syntactic features include a part of speech (POS) tagger and a parser for extracting features for

each hypothesis during rescoring. These tools also slow down the application of such models in deployed systems. A general principle is presented wherein substructures common to multiple hypotheses are efficiently shared in the transition based structured prediction algorithms employed by these auxiliary tools. It is shown that the proposed substructure sharing algorithm results in substantial speedup when utilizing these tools in automatic speech recognition.

The four methods and algorithms described above and detailed in this dissertation are evaluated using a state-of-the-art ASR system. It is demonstrated that these techniques could make the use of syntactically informed language models practical and hence, widespread. While this dissertation focuses on methods to efficiently apply syntactic language models to automatic speech recognition, many of the developed techniques may be used to efficiently utilize other complex language models in ASR and other applications such as machine translation.

Readers: Sanjeev Khudanpur and Hynek Hermansky

Committee: Sanjeev Khudanpur, Hynek Hermansky, Abhinav Sethy, and Daniel Povey

Acknowledgments

I have many people to acknowledge for their support and help towards finishing this PhD dissertation. I want to first start by mentioning a person who had the most impact on my academic life and, in a way, the person I am now: Frederick Jelinek. I had the privilege to have Fred as my academic/research advisor for four years until his unfortunate passing in 2010. He had a huge impact on building my research interest in the field. The many lessons I have learned from Fred cannot all be explained here. What I know for certain, however, is that they are lessons that I will carry with me throughout my life. His words of advice, of encouragement, and of criticism have been deeply rooted in me and will continue to be a part of my life.

Although Fred's death was really difficult for me to come to terms with, the support that I got from Sanjeev Khudanpur afterwards helped me a great deal with the transitionining process and in coping with the new situation. In fact, working with Sanjeev was not something new to me. I had always had his support from the day I joined the PhD program. I would not be wrong if I said he was also my advisor from day one (In fact, he had mentioned to me a several times that he was the main

reason Fred had noticed my PhD application and accepted me as his student). I would like to mention one valuable aspect of having Sanjeev as my advisor was his genius in immediately following and understanding the new ideas I would discuss with him. This combined with his broad knowledge of the field helped shaped many of the ideas presented in this dissertation. I also thank him for taking the pain of going through countless versions of this dissertation and editing them word-by-word.

I am also grateful to my co-advisor, Mark Dredze. His mentorship was extremely helpful in finishing this dissertation. His enthusiasm for research and publishing is contagious. One of the main advantages of having Mark as my advisor was his background and knowledge in machine learning. This particularly helped me to find connections between many of the ideas I had for speech recognition and language modeling and those proposed/used in the machine learning community. I also thank him for taking his time to carefully read this dissertation.

I am grateful to Bhuvana Ramabhadran and Abhinav Sethy who were my mentors during my internship at IBM. Thank you for the great opportunity and the support you provided during the rest of my PhD. Also, thanks to the rest of the IBM speech group for sharing their state-of-the-art software and models without which many of the experiments carried out in this dissertation would have been impossible.

I owe my incredible graduate study experience mainly to the Center for Language and Speech Processing (CLSP) with its highly intellectual atmosphere. To me CLSP means the best of the best faculties, smart and friendly students and helpful/kind

staffs. There are many people in CLSP I would like to thank. Hynek Hermansky, Jason Eisner and Damianos Karakos are among those faculties with whom I had the privilege to either collaborate or discuss and learn new ideas. Thanks to the brilliant friends and thriving students at the CLSP: Zhifei Li, Balakrishnan Varadarajan, Ehsan Variani, Nick Andrews, Ann Irvine, Scott Novotney, Sam Thomas, Sivaram Garimella, Sriram Ganapathy, Sridhar Nemala, Keith Kintzley, Mike Carlin. Thank you to the senior CLSPers Erin Fitzgerald, Arnab Goshal, Lambert Mathias, and Chris White for helping me in my transition as an undergrad to a grad student and introducing me many valuable tips and tricks to have an enjoyable experience as a grad student. Special Thanks to Markus Dryer, Carolina Parada and Jason Smith who were always there for me whether I had a technical question or if I wanted to discuss a problem with them. Markus was specially helpful in teaching me different computer science related techniques and algorithms. I still remember those nights we would stay late in the lab with Carolina to finish our projects and assignments. Jason was the one who I would go to if I needed to optimize the speed and memory usage of my code or to just sit down and have a cup of coffee. I am also thankful to the wonderful staff of CLSP and the ECE department: Desiree Cleves, Monique Folk, Debbie Race and Felicia Roane. They were all kind and helpful in resolving administrative issues.

Special thanks to the Human Language Technology Center of Excellence (HLT-COE) for funding my PhD and this dissertation. I am grateful to have been part

of the COE from its beginnings which gave me the opportunity to meet many great researchers and to attend interesting and informative discussions.

Thanks to my wonderful non-CLSP friends: Navid Shiee, Sahar Soleimanifard, Shahin Sefati, Ehsan Elhamifar and Safa Motesharrei. A great deal of my non-academic life was spent with these great friends. I am grateful for having them by my side and sharing many enjoyable moments with them.

Finally, I want to thank my parents, my sister and my brother. Despite being far away from them for most of my PhD, their emotional support and encouragement is the main reason I have made it thus far.

Dedication

In loving memory of my academic father, Frederick Jelinek.

This dissertation is dedicated to my parents Farah and Amir Abbas.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem

Automatic Speech Recognition (ASR) and Statistical Machine Translation (SMT), among other natural language applications rely on a language model (LM) to generate coherent natural language text. The LM plays a crucial role in identifying the correct word sequences by providing a strong *prior* over word sequences, in the often prohibitively large and complex hypothesis space of these systems.

Several advanced language modeling techniques and approaches have been proposed during the last two decades. Yet, most state-of-the-art ASR and SMT systems rely on the *n-gram* language modeling technique. While n-gram models are simple and efficient, it is widely believed that limiting the context to only the $(n-1)$ most recent words ignores the *linguistic* structure of language. Perhaps, the main rea-

sons for n-gram models to still be regarded as the dominant approach despite being *linguistically naïve* are that

- estimating them from text, large or small, is well understood, straightforward, and efficient, and
- the computational complexity vs. performance trade-off has long been in favor of n-gram models due to the prohibitive complexity of linguistically motivated language models.

The theoretical insufficiency of simple Markov models has been known for many years, e.g. Chomsky [3] proves that English cannot be modeled by a Markov chain because of long distance syntactic dependencies. Brill et al. [4] performed a series of experiments where humans are asked to correct and edit the output of an ASR system; additionally, the human subjects were asked to determine what types of information were used in their decision making process. They concluded that not only were humans able to make significant improvements in ASR over n-gram models[1], but furthermore the improvements were largely due to the use of *linguistic information* and *proficiencies*.

There has been a continuous push among researchers in the language modeling community to remedy the weaknesses of linguistically impoverished n-gram models and several statistical frameworks have been proposed to build *liguistically well-motivated* models which take syntactic, semantic, and other long-dependencies into

[1] They carried out the experiments over different domains.

account; In particular, there have been numerous efforts to incorporate syntactic information into language modeling [5] [6] [7] [2] [8] [9] [10]. The reason behind the use of syntactic knowledge in language modeling is quite simple and obvious: It is widely believed that natural language has syntactic structure. In addition, for domains and languages for which the availability of data is limited, the deep linguistic information captured by syntactic models may be the only way to improve upon n-gram models which rely entirely on co-occurrence statistics for their estimates[2].

One of the first successful attempts to build a statistical language model using syntactic structure and analysis was the structured language model (SLM) of Chelba and Jelinek [5], which showed a lot of promise and opened a new exciting area of language modeling. The basic idea behind structured language modeling is that the longer distance dependencies—such as the dependency between a verb and its subject—captured using a statistical parser are more useful, in terms of predicting words, than the recent previous context words. Unfortunately, practically issues such as its training scalability and the added complexity of applying it to ASR or SMT have prevented the widespread use of linguistically-informed SLMs in real applications.

First, the use of a *probabilistic* parser to reveal the dependencies in the history results in a mixture of possible equivalence classifications of history for each word position, which ultimately makes the framework impractical for estimating the parameters

[2]Although building robust and reliable statistical NLP tools such as POS taggers and parsers needed by syntactic language models for domains and languages with limited data availability is by itself a great challenge, we now have access to machine learning based methods and algorithms in the NLP community to build such high quality tools in *semi-supervised* and *unsupervised* scenarios.

of the model on a large amount of text data, and affects its training scalability.

Second, SLMs, like other proposed complex *long-span* language models, do not follow the simple Markov assumption and often rely on information far back in the context; Therefore, these models are not easily amenable to the left-to-right dynamic programming that is crucial in the (first pass) decoding for ASR or SMT. Instead, these complex models are applied in an N-best *rescoring* framework, where an n-gram model is used to generate a first pass search space, i.e. lattices in case of ASR task, and then the complex model is applied to rescore the top N hypotheses in the resulting search space. N-best rescoring, however, suffers from several well-known deficiencies and inefficiencies. To fully realize the superior power of utilizing long-span language models for the desired task, N needs to be sufficiently large, which increases the complexity of applying such models.

Moreover, the non-local features used in these models are usually extracted from a set of ASR hypotheses via auxiliary tools[3], which in the case of syntactic features include part of speech (POS) taggers and parsers. Finally, separately applying the auxiliary tools to each N-best list hypothesis leads to major inefficiencies as many hypotheses differ only slightly. Techniques for exploiting their commonalities are therefore in order.

This dissertation proposes strategies and algorithms to address such issues con-

[3]Some of the long-span language models integrate these tools as part of the model and are categorized as *joint* models. As an example, in the original SLM of Chelba and Jelinek [5] a POS tagger and a constituent statistical parser are part of the generative model of language. Other models such as the ones used in [9] and [6] rely on external auxiliary tools to incorporate long-span features into their language model.

cerning the practicality of complex models, particularly syntactic models. First, a framework is presented for training an efficient SLM based on dependency structures, derived using a state-of-the-art incremental parser. We demonstrate a new technique to maintain model compactness on a large training corpus using a general information-theoretic pruning method to reduce the size of the model. We then propose an efficient rescoring algorithm based on the well-known hill climbing idea to address the issues concerning the inefficient N-best rescoring presently used for applying long-span models. In fact, we show that our SLM when combined with the proposed hill climbing rescoring algorithm significantly improves over the baseline n-gram model while still remaining practical for the ASR task. Finally, we propose a general modification to the auxiliary tools used by syntactic language models to utilize the commonalities among the set of hypotheses being processed by the model, either in N-best rescoring or the proposed hill climbing algorithm.

The net result of the efficient estimation and application of SLMs via hill-climbing, while taking advantage of common substructures across the encountered hypotheses set, is arguably the first syntactic LM that is both fast enough accurate enough to be used in *real time* ASR applications.

While we mainly focus on syntactic language models and efficient ways to apply them in speech recognition, many of the developed techniques in this dissertation may be used to efficiently utilize other complex language models in ASR task and in other tasks such as machine translation.

Chapter 2

Background

This chapter provides the general background information for the problem of language modeling in automatic speech recognition (ASR) task. In the first part we introduce the task and review prior work on language modeling. Our emphasize is mainly on n-gram language models as they are still very much the state-of-the-art due to the simplicity and good performance. Those readers familiar with the speech recognition task and language modeling may skip this part.

The second part of this chapter provides an empirical study which shows that n-gram language models are weak word-predictors for the positions in a sentence where syntactic relations go beyond the limited n-gram context. The founding of this study suggests the need for explicit syntactic information in language models and motivates our efforts for building practical and efficient frameworks for incorporating such models into ASR systems.

2.1 A Statistical Formulation of the Speech Recognition

In the current statistical formulation of automatic speech recognition (ASR) systems [11], the recognizer seeks to find the word string

$$\widehat{W} = \arg\max_{W} P(W|A) = \arg\max_{W} P(A|W)P(W)$$

where A denotes the speech signal[1], $P(A|W)$ captures the probability that signal A is observed when the word string W is uttered, and $P(W)$ is the *a priori* probability that the speaker will utter W. The estimate for these probabilities are computed using statistical models where the parameters are estimated from data. The search for the maximizer $\widehat{W}$ is carried over all possible word sequences W,

$$W = w_1 w_2 \cdots w_m \quad w_i \in V$$

where V is the vocabulary used by the ASR system.

The probability $P(A|W)$ for any sequence of acoustic features $A = a_1 a_2 \cdots a_T$ and word sequence $W = w_1 w_2 \cdots w_m$ is estimated using an **acoustic model**[2]. The acous-

[1]The acoustic signal is a continuous signal and is not directly modeled. Instead, the acoustic signal is processed using the front-end pre-processing stage where a sequence of acoustic feature vectors are extracted and represented by A.

[2]Since the focus of this dissertation is mainly on language modeling, we briefly describe the most common technique for acoustic modeling and refer readers to [12] for details.

tic models in the state-of-the-art ASR systems are hidden Markov Models (HMM), where the HMM for $P(A|W)$ is composed by concatenating the HMM's for the individual words $w_1, w_2, \cdots, w_m$; The HMM for each w_i in turn is composed by concatenating the HMMs of the subword units, typically phonemes, that make up w_i. The HMM for each phoneme typically has a 3 state left-to-right topology[3]. The sequence of input acoustic features A are aligned to the states of the concatenated HMM for W, where observing an input acoustic vector by each state is modeled using a mixture of Gaussian distributions, also known as Gaussian Mixture Model (GMM). The parameters of the acoustic model are efficiently trained from large amount of transcribed speech using the Forward-Backward algorithm [14].

For the rest of this chapter, we discuss and explain the role of the second main component of an ASR system, i.e. **language model**, that estimates the probability $P(W)$. We start by giving a general description of a language model.

[3]To model the effect of neighboring phones, context dependent HMM based phones are used instead of context independent HMMs where the phone sequence is converted to to context dependent phones such as tri-phones. Moreover, to avoid the sparsity issue as a result of using context dependent HMMs (the number of tri-phone HMMs is $(\#phones)^3$), context dependent HMM states which have same acoustic properties are tied in practice, throughout the use of decision trees [13].

2.2 The Statistical Language Model

The aim of a language model is to assign a probability to any sequence of words $W = w_1 w_2 \cdots w_n$, where $w_i \in V$. Using the chain-rule, $P(W)$ may be factored as[4]

$$P(W) = \prod_{i=1}^{n} p(w_i | w_{i-1}, \cdots, w_1) \tag{2.1}$$

Note that in this factorization, we only need to define a conditional probability for each word (from a finite vocabulary V) given its sentence prefix (history).

Even with a moderate vocabulary size, the number of conditional probabilities $p(w_i | w_{i-1}, \cdots, w_1)$ that need to be estimated becomes unboundedly large and many of the word sequences in the history $w_1 w_2 \cdots w_{i-1}$ will never be seen in the training data used for training $P(W)$. In order to overcome the problem of sparsity for estimating $p(w_i | w_{i-1}, \cdots, w_1)$, the *history* $W_{-i} = w_1, w_2, \cdots, w_{i-1}$ is mapped to an equivalence class determined by a function $\phi : \mathcal{V}^* \rightarrow \{1, 2, \cdots, M\}$, so that one may set

$$P(W) \approx \prod_{i=1}^{n} p(w_i | \phi(W_{-i})). \tag{2.2}$$

Different language modeling techniques arise from applying different equivalence mappings, along with different parameterization of the conditional probabilities in the

[4]Although, there have been attempts to avoid this factorization by using whole sentence models [15], in this dissertation we use this factorization for generative language modeling. Another example of language models which do not use this factorization is the parser-based language model of Charniak [7], where the probability of a sentence is computed by marginalizing the probability of parse trees.

equation above (resulting from the chosen mapping). State-of-the-art ASR systems use a simple $(n-1)$-gram equivalence classification for the language model

$$\phi(W_{-i}) = w_{i-n+1}, w_{i-n+2}, \cdots, w_{i-1} \tag{2.3}$$

along with regular conditional probabilities as parameters, which results in an n-gram language model. According to the mapping above, histories that end in the same last $n-1$ words are equivalent. Almost all state-of-the-art ASR systems use this simple but but effective equivalence classification. We discuss different n-gram language modeling techniques in Section 2.2.2. However, let us first formally define metrics for measuring the quality of a language model, which is pivotal for evaluating different language modeling techniques, in the following section.

2.2.1 Performance Measure for Language Models

As it is the case with other statistical models, the quality of language models can vary due to differences in the amount of training data, quality/domain of the training data, parameterization of the model, etc. Therefore, formal evaluations are inevitable for comparing the quality of different language models. Here, we introduce two popular metrics used for measuring the quality of language models. The first is an intrinsic metric based on the divergence between the distribution implied by the language model and the ground truth distribution. The second is an extrinsic metric

of the impact of the language model on the desired application, which in this thesis is an ASR system.

2.2.1.1 Perplexity

Language models are typically evaluated through their *perplexity* on test data, an information-theoretic assessment of their predictive power [16, 17],

$$\begin{aligned} \mathrm{PPL}(\mathrm{p_M}) &= \exp\left(-\frac{1}{|T|}\sum_i \log p_M(w_i|W_{-i})\right) \\ &= \frac{1}{(\prod_i p_{-M}(w_i|W_{-i}))^{\frac{1}{|T|}}} \end{aligned} \tag{2.4}$$

where $|T|$ is the size of the test data and $p_M(w_i|W_{-i})$ is the conditional probability calculated by the language model at each word position. PPL is just the inverse of the (geometric) average probability assigned to each word in the test set by the model.

Intuitively, perplexity is the "branching factor", or the geometric average number of choices the LM has when predicting the next word. Hence, low PPL is desirable. PPLs from different models can be compared provided that:

- They are both calculated on a same test set.
- Both models are using the same vocabulary.
- Conventions such as counting the end of sentence symbol, `</s>`, as a word during PPL calculation should be consistent for the models being compared.

The reason PPL measures the goodness of a LM is due to the fact that it is related to the distance between the estimated probability distribution (through the LM) and the empirical distribution (of a test set). To observe this, one can re-write the Eqn. 2.4 as,

$$\text{PPL} = \exp\left(\sum_{w,h} p^*(w,h) \log p_M(w|h)\right)$$

where h indicates the history of the word-position w and $p^*(w,h)$ is the empirical (joint) distribution of (w,h) estimated on test set. The exponent above may be regarded as a cross-entropy of the test distribution and the LM distribution. Adding the exponent above to the negative entropy of the empirical distribution (which remains the same for any LM) results in,

$$D(p^*||p) = \sum_{w,h} p^*(w,h) \log p^*(w|h) + \sum_{w,h} p^*(w,h) \log p_M(w|h).$$

Therefor, a lower PPL results in lower Kullback-Leibler divergence between the underlying probability distribution implied by the LM and the empirical distribution of the test data.

Despite the fact that perplexity has been a popular comparison measure (because it allows language model research to develop independently of ASR systems, and it has many information-theoretically motivated properties), researcher have observed, that perplexity of different LMs does not always correlate well with their relative

performance in the desired task such as ASR [18, 19][5]. In case of ASR, this is mainly because perplexity fails to take into account the acoustic confusibility between words and other search issues in a recognizer.

2.2.1.2 Word Error Rate

In speech recognition, language models are evaluated by the word-error rate (WER) that results from using the LM in a automatic speech recognition (ASR) system. WER is the edit-distance (Levenshtein distance) between a *reference* transcription of the speech and the automatic transcript (hypothesis),

$$\text{WER} = \frac{I + D + S}{\#\text{ words in reference}} * 100 \tag{2.5}$$

where I, D, and S are the number off of word insertions, deletions, and substitutions, respectively, required to transform the automatic transcript (hypothesized by the ASR system) into the reference transcription.

2.2.2 n-gram Language Models

The most popular language models are based on a simple equivalence classification that uses an $(n-1)$-th order Markovian assumption, i.e. that the word w_i only

[5] Although experimental results for ASR in [18,19] suggests that perplexity can predict word-error rate quite well for n-gram models trained on in-domain data, perplexity is a poor predictor for other language models such as class n-gram models or the language models are trained on out-of-domain data.

depends on the immediately preceding $n-1$ words (Eqn. 2.3). The parameters of this model, i.e. the conditional probabilities, are estimated using the maximum likelihood criterion on a collection of training text data. It can be shown that the maximum likelihood solution has the following form:

$$p_{\mathrm{ML}}(w_i|w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^{i})}{C(w_{i-n+1}^{i-1})}, \tag{2.6}$$

where $w_j^i = w_j, w_{j+1} \cdots, w_i$ and $C(w_{i-n+1}^i)$ is the number of times the n-gram $w_{i-n+1}, w_{i-n+2}, \cdots, w_{i-1}$ occurs in the training data and $C(w_{i-n+1}^{i-1}) = \sum_{w_i \in V} C(w_{i-n+1}^i)$.

The maximum likelihood solution above is problematic as it assigns **unseen** n-grams a probability of 0 and does not generalize well[6]. To address this problem, *smoothing* is used, where the probabilities of observed n-grams are *discounted* and the discounted probability mass is re-distributed to assign non-zero probabilities to unseen n-grams. There have been many proposed smoothing algorithms for n-gram language models. In the following subsections, we describe two popular and most frequently used methods, namely Katz smoothing [20] and Kneser-Ney [21] smoothing[7]. It has been experimentally shown that Kneser-Ney smoothing consistently outperforms other smoothing techniques for the ASR task [17]. We refer readers to the thorough empirical survey of Chen and Goodman [17] for measuring the performance of different

[6]When we move to higher order n-gram models, the problem of unseen n-grams becomes more severe. While these models enable us to capture more context, their estimated conditional probabilities becomes more sparse and less robust.

[7]Another famous smoothing technique is Jelinek-Mercer smoothing [22] which is thoroughly described in Chapter 3.1 and used as a modeling tool for building our structured language model.

smoothing techniques.

2.2.2.1 Katz Smoothing

Katz smoothing uses a form of discounting in which the amount of discounting is proportional to that predicted by the Good-Turing estimate [23]. Good-Turing estimates are applied in a backoff scheme:

$$p_{\text{katz}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{C(w_{i-n+1}^{i})}{C(w_{i-n+1}^{i-1})} & r > k, \\ d_r \frac{C(w_{i-n+1}^{i})}{C(w_{i-n+1}^{i-1})} & k \geq r > 0, \\ \alpha(w_{i-n+1}^{i-1}) p_{\text{katz}}(w_i|w_{i-n+2}^{i-1}) & \text{otherwise.} \end{cases} \tag{2.7}$$

where k is some empirically chosen count above which no discounting is applied, $r = C(w_{i-n+1}^{i})$ is the (training) count of the n-gram and d_r is the discounting ratio, defined as follows:

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_k+1}{n_1}}{1 - \frac{(k+1)n_k+1}{n_1}} \tag{2.8}$$

where n_r is the count of n-grams (in training data) occurring r times and r^* is the adjusted count for n-grams that occur r times, given by the Good-Turing formula[8].

[8]The Good-Turing estimate states that for any n-gam that occurs r times, it's expected count is r^* using count-counts statistics.

That is,

$$r^* = (r+1)\frac{n_r+1}{n_r}.$$

According the Katz formulation in Eqn. 2.7, n-gram counts with $r > k$ are not discounted and are considered reliable. In addition, $\alpha(w_{i-n+1}^{i-1})$, the backoff weights, are calculated to ensure proper probabilities, i.e. $\sum_{w_i \in V} p_{\text{katz}}(w_i|w_{i-n+1}^{i-1}) = 1$. It is worth mentioning that the total number of counts discounted from infrequent n-grams with $1 \leq r \leq k$ in the Katz smoothing is equal to the total number of counts that should be assigned to n-grams with zero counts according to the Good-Turing estimate.

2.2.2.2 Kneser-Ney Smoothing

The original Kneser-Ney (KN) smoothing [21] is based on the idea that the lower-order model is significant only when the count of n-gram is small or zero in the higher-order model, and so should be optimized for that purpose. As an example consider the word `york` which is a fairly frequent word in English. So, in a uni-gram language model, it would be given a fairly high probability. Therefore, using any backoff language model, the bi-gram probability of "`york`$|w$", for w's for which "w `york`" does not occur at all in the training data, is assigned a high probability through the lower order uni-gram probability. However, when `york` occurs, it almost

always directly follows the word `new`. Therefore, we expect `york` to be very likely after we have seen new, but otherwise it should be assigned a very low probability. Recall that in the backoff scheme, the unigram model is only used if the bigram model is inconclusive, i.e. for unseen bigrams. Therefore, in a backoff unigram model, it is desirable to give the word `york` much less probability than its raw count based estimate. Kneser-Ney smoothing utilizes this idea by taking into account the **diversity of histories** and replacing the raw counts with the count of unique histories preceding a word for estimating unigram probabilities[9].

Chen and Goodman [17] proposes a interpolated version of the original Kneser-Ney smoothing method, which they show empirically outperforms the original backoff version:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\left(C(w_{i-n+1}^{i}) - D, 0\right)}{C(w_{i-n+1}^{i-1})} + \gamma(w_{i-n+1}^{i-1})p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}), \qquad (2.9)$$

where $\gamma(w_{i-n+1}^{i-1})$ is calculated as follows to ensure a valid probability:

$$\gamma(w_{i-n+1}^{i-1}) = D\frac{N_{1+}(w_{i-n+1}^{i-1}, \bullet)}{C(w_{i-n+1}^{i-1})}. \qquad (2.10)$$

where $N_{1+}(w_{i-n+1}^{i-1}, \bullet)$ is the number of unique words (types) that has been seen in

[9]The motivation used in the original text is totally different than the one described here [17].

the training data to follow a given context w_{i-n+1}^{i-1}, i.e.

$$N_{1+}(w_{i-n+1}^{i-1}, \bullet) = |\{w_i : C(w_{i-n+1}^{i}) > 0\}|. \tag{2.11}$$

We argued that for lower order n-grams it is better to base the estimation of the probability distribution on the count of unique histories in which word w_i may appear, instead of raw counts. To this end, KN smoothing replaces the number of observed n-grams $C(w_i|w_{i-n+2}^{i-1})$ with the number of *unique* words preceding w_{i-n+2}^{i} for estimating the lower order distribution $p_{\text{KN}}(w_i|w_{i-n+2}^{i-1})$. Ney et al. [24] suggests estimating the discounting factor, D, using deleted estimation on the training data which results in the value

$$D = \frac{n_1}{n_1 + 2n_2}, \tag{2.12}$$

where n_1 and n_2 are the number of n-gram with count 1 and 2 in the training data, respectively. Alternatively, the D parameter can be optimized using held-out data.

Additionally, Chen and Goodman [17] argued that better results can be achieved by using not a fixed discounting factor D for all n-gram counts, but a discounting value that depends on the n-gram itself. They propose using three different discount

values based on the count of the event under consideration:

$$D\left(C(w_{i-n+1}^{i})\right) = \begin{cases} D_1 & \text{if } C(w_{i-n+1}^{i}) = 1 \\ D_2 & \text{if } C(w_{i-n+1}^{i}) = 2 \\ D_{3+} & \text{if } C(w_{i-n+1}^{i}) \geq 3 \end{cases} \tag{2.13}$$

It turns out that optimal discounting parameters D_1,D_2, and D_{3+} have the following values (using deleted estimation on the training data),

$$D_1 = 1 - 2D\frac{n_2}{n_1}$$
$$D_2 = 2 - 3D\frac{n_3}{n_2}$$
$$D_{3+} = 3 - 4D\frac{n_4}{n_3}$$

Using the discounting factors above, the modified interpolated KN smoothing assigns the following n-gram probabilities:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\left(C(w_{i-n+1}^{i}) - D\left(C(w_{i-n+1}^{i})\right), 0\right)}{C(w_{i-n+1}^{i-1})} + \gamma(w_{i-n+1}^{i-1})p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}). \tag{2.14}$$

Again, $\gamma(w_{i-n+1}^{i-1})$ is set so that a valid probability distribution is obtained. That is,

$$\gamma(w_{i-n+1}^{i-1}) = \frac{\sum_{j\in\{1,2,3+\}} D_j N_j(w_{i-n+1}^{i-1}, \bullet)}{C(w_{i-n+1}^{i-1})}. \tag{2.15}$$

The modified interpolated KN has been shown to outperform most of other smoothing techniques and is widely regarded as the-state-of-the-art n-gram language modeling technique in ASR systems. Throughout this dissertation, we use the modified interpolated KN smoothing for building the **baseline n-gram models**.

2.2.3 Other Language Models

There have been many different statistical methods, besides the regular n-gram models, developed for language modeling in the past couple of decades [25] [26][10]. In the following subsections, we briefly review a few of these techniques and refer interested readers to an extensive survey by Goodman [26] for details. In particular, we describe discriminative language models, as part of this dissertation (Chapter 5) attempts to address challenges faced in incorporating long-span features using the discriminative framework.

[10]Perhaps, the main reason the language modeling field has attracted many researchers during this era is due to it's direct impact on automatic speech recognition, machine translation, spelling correction and many more applications.

2.2.3.1 Maximum Entropy Models

One approach to estimating the LM's parameters relies on Maximum Entropy (MaxEnt) learning [27]. In recent years, MaxEnt learning has become a popular approach for estimating language model parameters from a large corpus of text [28]. MaxEnt models allow for the inclusion of arbitrary features, providing a unified framework for combining a range of different dependencies [29]. Additionally, MaxEnt learning provides a well formed model to which new machine learning techniques can be naturally applied, i.e. different machine learning based techniques can be used to perform domain adaptation of MaxEnt LM parameters [30, 31] or L1 and L2-regularization techniques can be easily applied to perform feature selection and smoothing of the parameters [32]. In contrast, n-gram language models must satisfy back-off constraints and ensure that the model parameters (conditional probabilities) sum to one. MaxEnt models often yield comparable or better performance compared to standard n-gram models [28, 29].

Under the MaxEnt framework, the probability of word w given the history h has a log-linear (exponential) form[11],

$$P_\Lambda(w|h) = \frac{1}{Z_\Lambda(h)} \exp\left(\sum_j \lambda_j f_j(w,h)\right). \tag{2.16}$$

[11]The maximum entropy principle constructs a joint probability distribution that satisfies certain constraints (on training data) while staying ignorant about everything the constraints do not specify, i.e. it seeks most uniform probability distribution that satisfies the constraints. It is easy to show – using Lagrange multipliers – that these maximum entropy assumptions result in an exponential distribution [11].

$Z_\Lambda(h)$ is the normalization factor for the given history h,

$$Z_\Lambda(h) = \sum_{w' \in V} \exp\left(\sum_j \lambda_j f_j(w', h)\right). \tag{2.17}$$

$f_j(w, h)$ is the j-th feature function based on word w and history h. Each λ_i represents the weight of the corresponding feature and the set of feature weights $\Lambda = \{\lambda_1, \lambda_2, \cdots\}$ forms the parameters of the model, estimated during LM training. In the case of n-gram MaxEnt models, each n-gram corresponds to a single features. For instance, a bigram (a, b) feature would take the form:

$$f_j(w, h) = \begin{cases} 1 & \text{if } w = b \text{ and } h \text{ ends in } a \\ 0 & \text{otherwise} \end{cases}$$

for some a and b. Typically, for an n-gram MaxEnt model the feature set and parameters are defined to include all the n-grams ($n = 1, 2, \cdots N$ where N is the order of the LM) seen in the training data. The parameters of a MaxEnt LM are estimated to fit the training sentences using a Maximum Likelihood (ML) criterion[12]. There are several approaches to maximizing the ML objective, such as Generalized Iterative Scaling (GIS) [33], or gradient based methods, such as L-BFGS [34]. In addition, an L_2 regularization term $||\Lambda||_2^2$ with weight γ is added to the ML objective function to prevent overfitting and provide smoothing [32]. γ is usually chosen empirically using

[12]It can be shown that the ML criterion using the exponential parametrization of Eqn. 2.16 is the dual objective fo the maximum entropy objective function.

a development set.

A significant disadvantage of MaxEnt LM training is the need to compute the normalizer, also called the partition function $Z_\Lambda(h)$ in (2.17), for which one must sum over *all possible* words $w \in V$ for every history h. In the naive implementation of an n-gram MaxEnt LM, the complexity for computing normalization factors (and feature expectations) for a single iteration, therefore, is $O(|H| \times |V|)$, where $|H|$ is the number of history tokens seen in training data and $|V|$ is the vocabulary size, typically on the order of tens of thousands. Wu [29] proposed a hierarchical method for nested and non overlapping features, e.g., n-gram features. The hierarchical training procedure reduces the complexity for calculating normalization factors and n-gram feature expectations (for one iteration) to $O(\sum_{n=1}^{N} \#n\text{-grams})$, the same complexity as training the corresponding back-off n-gram LM.

Despite the fact that MaxEnt LMs introduces a unified framework for incorporating arbitrary features such syntactic features, its computational complexity makes their application challenging specially for large-scale applications[13].

2.2.3.2 Discriminative Language Models

All the language model techniques covered so far are categorized as *generative*. An alternative method for language modeling is *discriminative* models. Unlike generative models where the training objective is to compute maximum likelihood estimate on

[13]The hierarchical approach of Wu [29] loses its effectiveness when working with arbitrary non-nested features.

a huge amount of text data, discriminative models directly attempt to minimize the recognition error while estimating the parameters of the LM. Therefor, the parameters of a discriminative language model are directly optimized for the task, i.e ASR. Discriminative language models do not attempt to estimate the probability of word sequences – in contrast to generative models; Instead, they are often used as a corrective model to rank a given set of hypotheses of an ASR system and to complement generative LMs used in ASR systems[14].

A popular framework for discriminative language modeling is through the use of *global linear* discriminative models [37]:

$$\hat{W} = \underset{W \in \mathbf{GEN}(A)}{\arg\max} \langle \alpha, \Phi(A, W) \rangle \tag{2.18}$$

where $\Phi(A, W)$ is a vector of language model features, and α is the corresponding weights. $\mathbf{GEN}(A)$ is the set of ASR hypotheses for the acoustic input A. Roark et al. [37] proposed a feature vector $\Phi(A, W)$ that is only based on n-grams and used the perceptron algorithm to train the weights α. This framework provides the flexibility to extend beyond the local n-gram context to incorporate long range dependencies such as features based on full-sentence parser tree, morphological features, and trigger features. As an example, Collins et al. [2] uses the framework to train a syntactic discriminative model where they include feature based on the syntactic parse trees of

[14]We distinguish discriminative language models from discriminatively trained generative language models where a parameters of a generative model are trained using discriminative criteria such as Minimum Classification Error (MCE) [35, 36] or Minimum Bayes Error (MBR).

word sequences in the hypotheses set. However, global-linear discriminative models with long-span features can not be efficiently trained and used with ASR lattices. We address this important issue in this dissertation and propose an efficient algorithm for training and applying discriminative language models with long-span features.

2.3 Weaknesses of n-gram LMs

Despite all the efforts made by researchers to incorporate longer-dependency features such as syntactic features [2, 5, 6, 8, 9], statistical language models used in deployed systems for speech recognition, machine translation and other human language technologies are almost exclusively n-gram models. [25]. While n-gram models have been outperformed by many of the proposed complex models and are regarded as *linguistically naïve*, their simplicity and the ease of estimating them from any amount of text, have made them the dominant approach in the state-of-the-art ASR and MT systems. This unusual resilience of n-grams, as well as their weaknesses, are examined in this section. It is demonstrated that n-grams are good word-predictors, even linguistically speaking, in a large majority of word-positions. However, they lose their predictive power in word-positions where syntactic relation spans beyond the context captured by n-gram models and that the lack of predictive power for these positions *can not* be overcome by merely increasing training data size. The founding of our empirical study is yet another encouraging evidence that shows to improve

over n-grams, one must explore syntax-aware (or other) language models that focus on positions where n-grams are weak (i.e. the proposed structured language model in Chapter 3 or discriminative syntactic language model of Chapter 5).

Consider the following sentence, which demonstrates why the $(n-1)$-gram equivalence classification of history in n-gram language models may be insufficient:

`<s> i asked the vice president for his endorsement </s>`

In an n-gram LM, the word `for` would be modeled based on a 3-gram or 4-gram history, such as `<vice president>` or `<the vice president>`. Given the *syntactic* relation between the preposition `for` and the verb `asked` (which together make a compound verb), the strongest evidence in the history (and hence the best classification of the history) for word `for` should be `<asked president>`, which is beyond the 4-gram LM. Clearly, the *syntactic relation* between a word position and the corresponding words in the history spans beyond the limited $(n-1)$-gram equivalence classification of the history.

This is but one of many examples used for motivating *syntactic features* [5, 9] in language modeling. However, it is legitimate to ask if this deficiency could be overcome through sufficient data, that is, accurate statistics could be somehow gathered for the n-grams even without including syntactic information. We empirically show that $(n-1)$-gram equivalence classification of history is not adequate to predict these cases. Specifically, n-gram LMs lose predictive power in the positions where the *head-word* relation, exposed by the syntactic structure, goes beyond $(n-1)$ previous words

(in the history.)

We postulate the following hypotheses:

Hypothesis 1 *There is a substantial difference in the predictive power of n-gram LMs at positions within a sentence where syntactic dependencies reach further back than the n-gram context versus positions where syntactic dependencies are local.*

Hypothesis 2 *This difference does not diminish by increasing training data by an order of magnitude.*

In the following section (Section 2.3.1), we present a set of experiments to support the hypotheses above.

2.3.1 Experimental Evidence

In this section, we explain our experimental evidence for supporting the hypotheses stated above. First, Section 2.3.1.1 presents our experimental design where we use a statistical constituent parser to identify two types of word positions in a test data, namely positions where the headword syntactic relation spans beyond recent words in the history and positions where the headword syntactic relation is within the n-gram window. The performance of an n-gram LM is measured on both types of positions to show substantial difference in the predictive power of the LM in those positions. Section 2.3.1.3 describes the results and analysis of our experiments which supports our hypotheses.

Throughout the rest of this dissertation, we refer to a position where the headword syntactic relation reaches further back than the n-gram context as a *syntactically-distant* position and other type of positions is referred to as a *syntactically-local* position.

2.3.1.1 Design

Our experimental design is based on the idea of comparing the performance of n-gram LMs for *syntactically-distant* vs. *syntactically-local* . To this end, we first parse each sentence in the test set using a constituent parser, as illustrated by the example in Figure 2.1. For each word w_i in each sentence, we then check if the "syntactic heads" of the preceding constituents in the parse of $w_1, w_2, \cdots, w_{i-1}$ are within an $(n-1)$ window of w_i. In this manner, we split the test data into two disjoint sets, $\mathcal{M}$ and $\mathcal{N}$, as follows,

$$\mathcal{M} = \{\text{word positions } i \text{ where } (h.\text{w}_{-1}, h.\text{w}_{-2} = w_{j-1}, w_{j-2})\}$$

$$\mathcal{N} = \{\text{word positions } i \text{ where } (h.\text{w}_{-1}, h.\text{w}_{-2} \neq w_{j-1}, w_{j-2})\}$$

Here, h_{-1} and h_{-2} correspond, respectively, to the two previous *exposed headwords* at position i, based on the syntactic structure. Therefore, $\mathcal{M}$ corresponds to the positions in the test data for which two previous *exposed heads* match exactly the two previous words. Whereas, $\mathcal{N}$ corresponds to the position where at least on of the

exposed heads is further back in the history than the two previous words, possibly both.

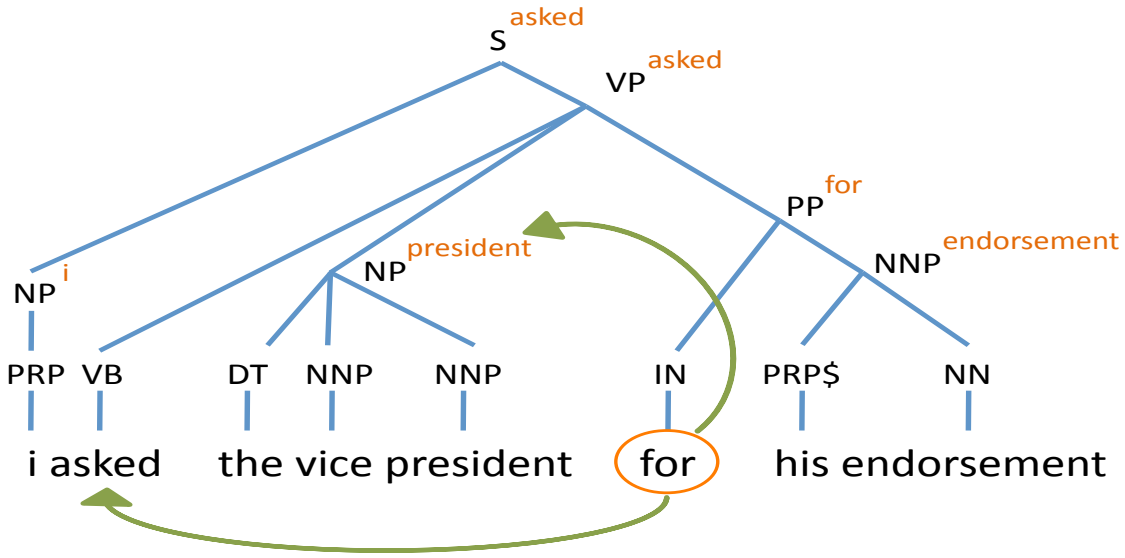


Figure 2.1: An example of a position in a test sentence for which two previous exposed headwords are further back in the history than the two previous words

To extract the exposed headwords at each position, we use a constituent parser to obtain the syntactic structure of a sentence followed by *headword percolation* procedure to get the headwords of corresponding syntactic phrases in the parse tree. The following method, described in [9], is then used to extract exposed headwords from the history of position i from the full-sentence parse trees:

1. Start at the leaf corresponding to the word position (w_i) and the leaf corresponding to the previous context word (w_{i-1}).

2. From each leaf, go up the tree until the two paths meet at the lowest common ancestor (LCA).

3. Cut the link between the LCA and the child that is along the path from the context word w_{i-1}. The head word of the the LCA child, the one that is cut, is chosen as previous exposed headword $h.\mathrm{w}_{-1}$.

These steps may be illustrated using the parse tree shown in Figure 2.1. Let us show the procedure for our example from Section 2.3. Figure 2.1 shows the corresponding parse tree of our example. Considering word position w_i=`for` and w_{i-1}=`president` and applying the above procedure, the LCA is the node $\mathrm{VP}^{\texttt{asked}}$. Now, by cutting the link from $\mathrm{VP}^{\texttt{asked}}$ to $\mathrm{NP}^{\texttt{president}}$ the word `president` is obtained as the first exposed headword ($h.\mathrm{w}_{-1}$).

After the first previous exposed headword has been extracted, the second exposed headword also can be obtained using the same procedure, with the constraint that the node corresponding the second headword is different from the first [9]. More precisely,

1. set $k = 2$
2. Apply the above headword extraction method between w_i and w_{i-k}.
3. if the extracted headword has previously been chosen, set $k = k + 1$ and go to step (2).
4. Otherwise, return the headword as $h.\mathrm{w}_{-2}$.

Continuing with the example of Figure 2.1, after `president` is chosen as $h.\mathrm{w}_{-1}$, `asked` is chosen as $h.\mathrm{w}_{-2}$ of position `for` by applying the procedure above. Therefore, in

this example the position corresponding to word `for` belongs to the set $\mathcal{N}$ as the two extracted exposed headwords (`asked,president`) are different from the two previous context words (`vice,president`).

After identifying sets $\mathcal{N}$ and $\mathcal{M}$ in our test data, we measure *perplexity* of n-gram LMs on $\mathcal{N}$, $\mathcal{M}$ and $\mathcal{N} \cup \mathcal{M}$ separately. That is,

$$\begin{aligned}
\text{PPL}_{\mathcal{N}\cup\mathcal{M}} &= \exp\left(-\frac{1}{|\mathcal{N}\cup\mathcal{M}|}\sum_{i\in\mathcal{N}\cup\mathcal{M}}\log p(w_i|w_{i-n+1}^{i-1})\right) \\
\text{PPL}_{\mathcal{N}} &= \exp\left(-\frac{1}{|\mathcal{N}|}\sum_{i\in\mathcal{N}}\log p(w_i|w_{i-n+1}^{i-1})\right) \\
\text{PPL}_{\mathcal{M}} &= \exp\left(-\frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}}\log p(w_i|w_{i-n+1}^{i-1})\right)
\end{aligned}$$

where $p(w_i|w_{i-1}w_{i-2}\cdots w_{i-n+1})$ is the conditional probability calculated by an n-gram LM at position i and $|.|$ is the size (in number of words) of the corresponding portion of the test.

In addition, to show the performance of n-gram LMs as a function of training data size, we train different n-gram LMs on 10%,20%,$\cdots$,100% of a large corpus of text and report the PPL numbers using each trained LM with different training data size. For all sizes less than 100%, we select 10 random subset of the training corpus of the required size, and report the average perplexity of 10 n-gram models. This will enable us to observe the improvement of the n-gram LMs on as we increase the training data size. The idea is to test the hypothesis that not only is there significant gap between predictive power of the n-gram LMs on sets $\mathcal{N}$ and $\mathcal{M}$, but also that

this difference does not diminish by adding more training data. In other words, we want to show that the problem is not due to lack of robust *estimation* of the model parameters but due to the fact that the included features in the model (n-grams) are not *informative* enough for the positions $\mathcal{N}$.

2.3.1.2 Setup

The n-gram LMs are built on $400M$ words from various Broadcast News (BN) data sources including [38]: 1996 CSR Hub4 Language Model data, EARS BN03 closed captions, GALE Phase 2 Distillation GNG Evaluation Supplemental Multilingual data, Hub4 acoustic model training scripts (corresponding to the 300 Hrs), TDT4 closed captions, TDT4 newswire, GALE Broadcast Conversations, and GALE Broadcast News. All the LMs are trained using modified Kneser-Ney smoothing (as described in Section 2.2.2.2). To build the LMs, we sample from each source and build a source specific LM on the sampled data. The final LMs are then built by interpolating those LMs. Also, we do not apply any pruning to the trained LMs, a step that is often necessary for speech recognition but not so for perplexity measurement. The test set consists of the NIST rt04 evaluation data set, dev04f evaluation set, and rt03 evaluation set. The test data includes about $70K$ words.

We use the parser of [39], which achieves state-of-the-art performance on broadcast news data, to identify the word poisons that belong to $\mathcal{N}$ and $\mathcal{M}$, as was described in Section 2.3.1.1. The parser is trained on the Broadcast News treebank from Ontonotes

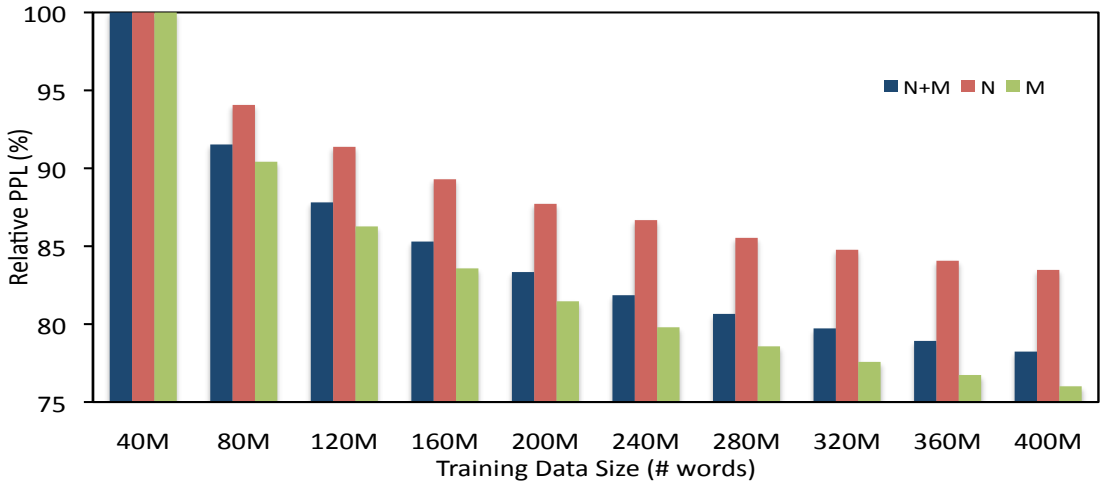
[1] and the WSJ Penn Treebank [40] along with self-training on 1996 Hub4 CSR [41] utterances.

2.3.1.3 Analysis

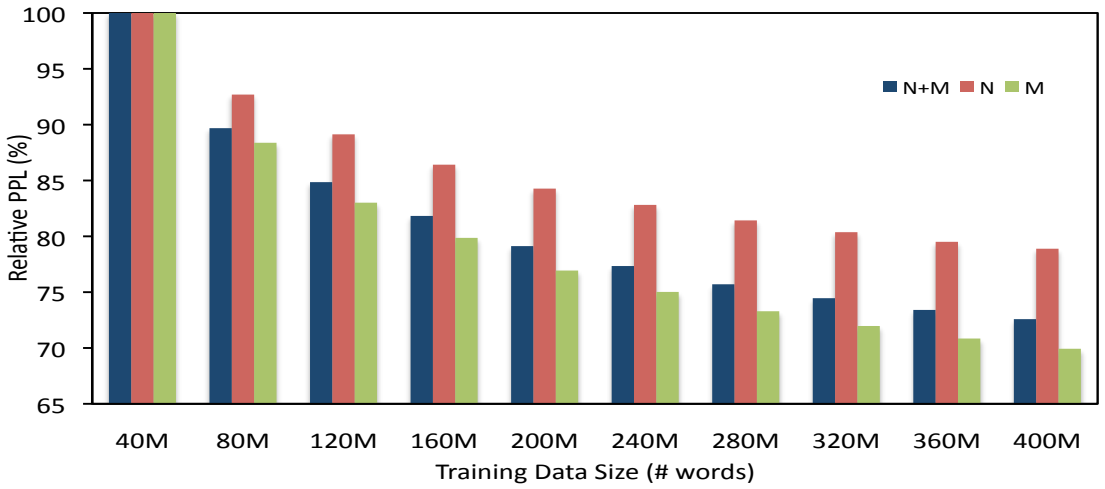
We found that $\frac{|\mathcal{N}|}{|\mathcal{N} \cup \mathcal{M}|} \approx 0.25$ in our test data. In other words, two previous exposed headwords go beyond 2-gram history for about 25% of the test data.

We train 3-gram and 4-gram LMs on 10%,20%,$\cdots$,100% of the BN training data, where each 10% increase corresponds to about $40M$ words of training text data. Figure 2.2 shows reduction in perplexity with increasing training data size on the entire test set $\mathcal{N} + \mathcal{M}$, on its *syntactically local* subset $\mathcal{M}$, and the *syntactically distant* subset $\mathcal{N}$. The figure basically shows relative perplexity instead of absolute perplexity — 100% being the perplexity for the smallest training set size — so the rate of improvement for 3-grams and 4-gram LMs can be compared. As can be seen from Figure 2.2, there is a substantial gap between the improvement rate of *perplexity* in syntactically distant positions compared to that in syntactically local positions (with 400M woods of training data, this gap is about 10% for both 3-gram and 4-gram LMs). In other words, increasing the training data size has much more effect on improving the predictive power of the model for the positions included in $\mathcal{M}$. Also, by comparing Figure 2.2(a) to 2.2(b) one can observe that the gap is not overcome by increasing the context length (using 4-gram features).

Also, to better illustrate the performance of the n-gram LMs for different portions



(a)



(b)

Figure 2.2: Reduction in perplexity with increasing training data size on the entire test set $\mathcal{N} + \mathcal{M}$, on its *syntactically local* subset $\mathcal{M}$, and the *syntactically distant* subset $\mathcal{N}$. The figure shows relative perplexity instead of absolute perplexity — 100% being the perplexity for the smallest training set size — so that (a) 3-gram and (b) 4-gram LMs may be directly compared.

of our test data, we report the absolute values of PPL results in Table 2.1. It can be seen that there exits a significant difference between *perplexity* of sets $\mathcal{N}$ and $\mathcal{M}$ and that the difference gets larger as we increase the training data size.

Position in Test Set	Training Data Size			
	40M words		400M words	
	3-gram	4-gram	3-gram	4-gram
$\mathcal{M}$	166	153	126	107
$\mathcal{N}$	228	217	191	171
$\mathcal{N} + \mathcal{M}$	183	170	143	123
$\frac{PPL_{\mathcal{N}}}{PPL_{\mathcal{M}}}$	138%	142%	151%	161%

Table 2.1: Perplexity of 3-gram and 4-gram LMs on *syntactically local* ($\mathcal{M}$) and *syntactically distant* ($\mathcal{N}$) positions in the test set for different training data sizes, showing the sustained higher perplexity in distant v/s local positions.

Taken together, the results of Table 2.1 and Figure 2.2 suggest that predicting the next word is about 50% more difficult when its syntactic dependence on the history reaches beyond n-gram range. They also suggest that this difficulty does not diminish with increasing training data size. If anything, the difficulty in predicting word positions with nonlocal dependencies relative to those with local dependencies appears to *increase* with increasing training data and n-gram order.

Chapter 3

Efficient Structured Language Modeling

3.1 Introduction

It is widely held that simple Markov assumption of n-gram language models, by limiting the contextual information only to $(n-1)$ recent words, is not sufficient for modeling the natural language. We showed, in Section 2.3, that the performance of n-gram models is substantially lower when evaluated on *syntactically-distant* positions. Many researchers have developed statistical modeling frameworks to incorporate the "syntactic structure of language back into language modeling"[1].

One of the first successful attempts to build a statistical language model based on

[1] Attributed to Fred Jelinek at the closing session of the 1995 Johns Hopkins Center for Language and Speech Processing workshop.

syntactic structure and analysis was proposed and demonstrated in [5]. The proposed framework is referred to as Structured Language Modeling (SLM). The SLM utilizes additional types of information by constructing more complexly structured models than the n-gram model and can be loosely categorized a joint model of the word sequence W and its syntactic structure T. The SLM assigns a joint probability $P(W,T)$ to every word sequence W and every possible binary parse tree T, where T's terminals are the words W, along with their part-of-speech (POS) tags, and its internal nodes comprise non-terminal labels and lexical "heads" of phrases. The SLM consists of three components: a word-predictor, a parser and a POS-tagger which were all modeled using a conditional (left-to-right) parameterization. In another attempt, [42] and [6] use the exposed headwords from syntactic structure, along with other sources of information such as topic, in a maximum-entropy based language modeling framework and show substantial improvements over regular n-gram LMs. [9] also uses exposed headwords from full-sentence parse tree in a neural network based LM to substantially reduce the WER in an Arabic speech recognition task.

The main idea behind structured language modeling can also be cast in a Bayesian framework where instead of having a deterministic (unique) equivalence classification $\phi(W_{-i})$ of the history W_{-i} at each word position, as is the case with n-gram models, there is an entire set of possible equivalence classification at each position weighted by their corresponding prior probability $p\left(\phi(W_{-i})|W_{-i}\right)$. The set of possible $\phi(W_{-i})$ and the corresponding distribution on them are obtained by identifying partial parse

trees (also known as prefix trees) at each position using a incremental (left-to-right) shift-reduce constituent parser in [5].

The framework we present in this chapter is closely related to the SLM. However, our goal is to build framework which results in a robust, accurate and practical SLM. This requires our framework to have the capability to train SLMs on large data, and to be able to build a robust model which fully takes advantages of syntactic structures and features. In addition, to be able to practically integrate SLMs into ASR systems, we need to build SLMs which are space/memory efficient. Finally, to motivate the usefulness of SLMs for state-of-the-art ASR system, we evaluate our SLM against 4-gram baseline model (which is the de facto standard in most current ASR systems)[2]. We distinguish our work from past SLM framework in the following directions:

- Our framework is based on dependency structures derived using a state-of-the-art probabilistic shift-reduce parser [43], in contrast to the original SLM [5] which parametrizes the parser component with a conditional model. Our parser is also a left-to-right incremental parser. The quality of the identified syntactic structures plays a key role in the performance of a SLM. In fact, it was pointed out in [5], Section 4.4.1, that if the final best tree is used to capture the partial trees (prefix-trees) at each position (just one partial tree at each position imposed by the final best tree), SLM can reduce the perplexity using the ex-

[2]In contrast to the original SLM framework where all the evaluations and experimental performance measurements are reported against 3-gram models.

posed headword information by about 40%[3]. This observation is used in [44] to enhance PPL and WER performance of the SLM by enriching the parser component of the model with more refined syntactic features. We believe that the use of a state-of-the-art probabilistic shift-reduce parser in our framework, enables us to build a SLM which significantly improves the baseline n-gram model.

- We propose a novel hierarchical interpolation of syntactic parameters that achieves better performance without significant model complexity.

- While the SLM significantly improves over the baseline n-gram model, the size of a SLM is much bigger than regular n-gram LMs when trained on same training data. This is mainly due to the fact that at each word position there are multiple possible equivalence classifications (as mentioned in the above) of the history and these separate equivalence classifications each might contribute a separate set of activated features to the final model. This issue was addressed in the original SLM by heavily pruning the space of partial trees locally, to reduce the number of possible equivalence classifications at each position, only those $\phi(W_{-i})$ were permitted which had large $p\left(\phi(W_{-i})|W_{-i}\right)$. We instead propose to estimate the SLM first without pruning and then apply a general information-theoretic algorithm [45] for pruning resulting SLM instead, which substantially

[3]This is referred to as non-casual perplexity in [5] and is not a valid perplexity because the prediction of a word position is conditioned on the best final tree which has been produced by looking at the whole sentence.

reduces the size of the LM, enabling us to train on large data. Under our procedure, highly predictive equivalence classifications $\phi(W_{-i})$, which have low $p\left(\phi(W_{-i})|W_{-i}\right)$ but occur at many positions i, have a better chance to survive the pruning.

- Unlike the original SLM framework, we do not attempt to re-estimate the parser parameters through the expectation-maximization (EM) algorithm [46] and we stick to the parameters which are estimated on a labeled treebank style data[4]. This is equivalent to the initialization stage in the original SLM framework. The reason is that the main focus of our work is to make the SLM framework efficient and practical both in terms of the complexity of the trained model and its ultimate application to the task of speech recognition. Moreover, as it is shown in [5] (and we will also show later in this chapter) the full power of SLM is realized when *interpolated* with the baseline n-gram model and although the re-estimation strategy improves the SLM, it has less effect on the perplexity of the interpolated model. Nevertheless, we emphasize that integrating the re-estimation stage into our framework is an obvious step and can be easily done.

[4]The details of training procedure of our parser is described in Section 3.3

3.2 Dependency Parser

Syntactic information can be encoded in terms of headwords of phrases and their POS tags, which are extracted from a syntactic analysis of a sentence [5, 9]. Dependency parsing is an approach to automatic syntactic analysis of natural language inspired by the theoretical linguistic formalism of dependency grammars [47]. The fundamental notion of *dependency structure* is based on the assumption that the syntactic structure of a sentence consists of binary, asymmetrical relations called *dependencies*. A dependency relation holds between a word in the sentence, called the *dependent* (or *modifier*), and another word on which it depends, called the *head* (or *governor*). A word can have multiple modifiers but it can serve as a head of at most one word in a sentence. The dependency structure and the corresponding relations are represented by a *dependency tree* which is a directed graph where each edge (arc) encodes a relation between a head and it's dependent. The edges of the tree can be labeled to represent the specific role or relation between the head and modifier. Figure 3.1 shows the (unlabeled) dependency tree structure of our example sentence from Section 2.3. In this example, the verb `asked` serves as the head of words `president` and `for`.

The information encoded in a dependency structure representation is different from the information captured in a *constituent phrase structure* representation. While the dependency structure represents head-dependent relations between words, classified by functional categories such as subject (SBJ) and object (OBJ), the phrase

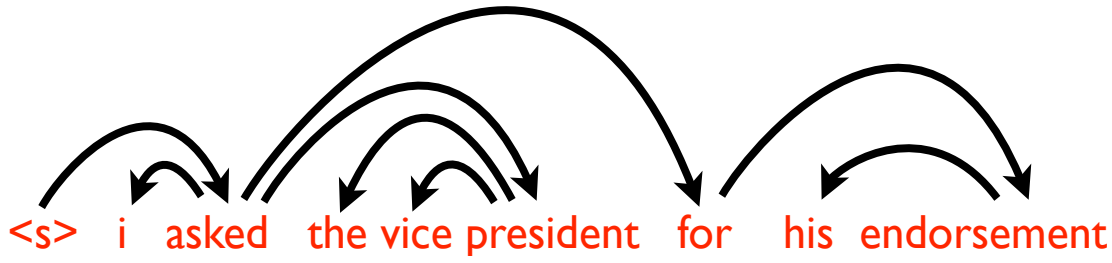


Figure 3.1: An example of a dependency tree

structure represents the grouping of words into phrases, classified by structural categories such as noun phrase (NP) and verb phrase (VP). Nevertheless, it is worth mentioning that a dependency structure can in fact be extracted from a constituent parse structure through a set deterministic headword percolation rules[5] [48].

The task of dependency parsing is to automatically analyze the dependency structure of a given input sentence. More formally, given an input sentence $W = w_0 w_1 \cdots w_k$ a dependency parser maps W to its dependent tree (or graph). We always assume that $w_0 =$ `<s>`, which serves as the `ROOT` of the sentence, to simplify the computational implementation, this way every word of the sentence must have a syntactic head (in Figure 3.1 `<s>` is the head of the verb `asked`).

Modern data-driven[6] dependency parsers are mostly divided into two classes *transition-based* and *graph-based*. As their name suggests, Transition-based parsers are based on a transition system which starts in an initial state and gradually pre-

[5]The most famous toolkit for this matter is Stanford converter which converts treebank style constituencies to labeled dependencies

[6]There exist another category of dependency parsing methods which are called Grammar-based which rely on a formal grammar.

dicts the dependency structure of an input sentence. The prediction at each step of parsing is performed by a model (classifier) which is trained on a set of training sentences annotated with gold-standard dependency graphs [43, 49–51]. Broadly speaking, transition-based dependency parsers fall in the family of "history-based" structure predictors and hence, permit the use of rich non-local expressive feature representation in the model [43]. "Graph-based" methods instead search over the space of complete dependency graphs for a sentence, given the induced model. Transition-based models, compared to graph-based dependency parsing, typically offer linear time inference complexity as they use a greedy search, also known as *deterministic* search, where at each step the most likely move (action), based on the classifier, is chosen to transition to the next configuration[7].

In this dissertation and for the purpose of structured language modeling, we use a shift-reduce transition-based dependency parser. The ultimate goal is to use the parser to assign better equivalence classification of the history for our language model. The incorporation of a SLM into an ASR system will hence be dependent on the parser. The choice of a linear-time shift-reduce parser is suitable for our task as one of the main constraints in applying new complex models to ASR is to have a reasonable run-time.

The following section presents a formal definition and description of a shift-reduce dependency parser. We then briefly presents metrics used for evaluating dependency

[7]Other type of heuristics such as beam-search or best-first algorithm can also be applied as we will them discuss in Section 3.2.1. Nevertheless, the complexity of these parsing algorithms remains linear at the expense of losing the guarantees for returning the most likely tree.

parsers.

3.2.1 Shift-reduce dependency parsing

Transition-based dependency parsing methods produce dependency structure of an input sentence by making sequence of transitions proposed by a learning algorithm (classifier). This is sometimes referred to as *shift-reduce* dependency parsing, since the overall approach is inspired by deterministic shift-reduce parsing for context-free grammars. In this Section, we first develop formal notation to describe a shift-reduce dependency parser and then, we walk through an example sentence to clarify the notation.

In a typical transition-based parser, the input words are put into a queue in their given order, and partially built structures are organized by a stack. A set of shift-reduce actions are defined, which consume words from the queue and build the output parse. Formally, the state of a shift-reduce parser is defined as $\pi = \{S, Q\}$: S is a stack of subtrees $s_0, s_1, \ldots$ (s_0 is the top tree) and Q is the queue of words in the input word sequence. We denote the space of all possible states with Π and all states along the way are chosen from this space. A transition (or action) $\omega \in \Omega$ advances the parser (decoder) from state to state, where the transition ω_i changes the state from π_i to π_{i+1}. A sequence of valid transitions, starting in the initial state, π_0 for a given sentence and ending in one of several terminal (final) states π_f, defines a feasible dependency tree for the input sentence. Dependency trees are built by processing the

words left-to-right and the sequence of states $\{\pi_0 \dots \pi_i, \pi_{i+1} \dots \pi_f\}$ are mapped to final output structure. The initial state $\pi_0 = \{\emptyset, \{w_0, w_1, \dots\}\}$ is an empty stack and a queue of all input words. The final states occur when Q is empty which is when all input words have been processed and S contains a single tree (the output).

Given the current state of the parser $\pi = \{\{s_0 s_1 \cdots\}, \{w_j, w_{j+1} \cdots\}\}$, Ω is determined by the set of possible dependency labels $r \in \mathcal{R}$ and one of three transition types:

- `Shift`: remove the head of Q (w_j) and place it on the top of S as a singleton tree (only w_j.)
- `Reduce-Left`(r): replace the top two trees in S (s_0 and s_1) with a tree formed by combining s_0 and s_1. s_0 is attached as the rightmost child of s_1, i.e. the head of s_0 becomes a dependent of the head of s_1 with label r.
- `Reduce-Right`(r): same as `Reduce-Left`(r) except reverses s_0 and s_1. That is, the head of s_1 becomes a dependent of the head of s_0 with label r.

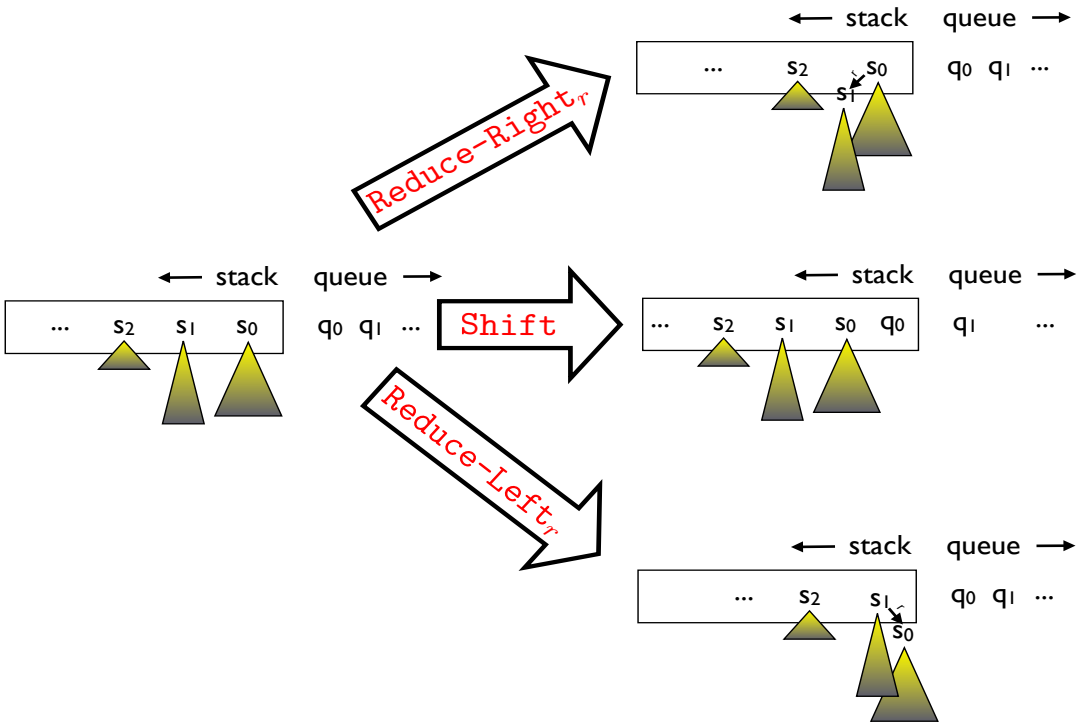


Figure 3.2: Set of possible shift-reduce dependency parser actions and their effect on a parser state.

In the following, we summarize the state definition and the set of possible actions:

$$\text{Transition System:} \begin{cases} \text{State defnition:} & \pi = \{S, Q\} \\ \text{Initial state:} & \pi_0 = \{\emptyset, \{w_0, w_1, \ldots\}\} \\ \text{Final state(s):} & \pi_f = \{\{s\}, \emptyset\} \end{cases}$$

$$\text{Actions:} \begin{cases} \texttt{Shift} \\ \texttt{Reduce-Left}(r) \\ \texttt{Reduce-Right}(r) \end{cases} \tag{3.1}$$

Figure 3.2 shows the effect of taking each possible action on a parser state. To utilize

a probabilistic data-driven dependency parser, a classifier g based on a maximum-entropy (also knowns as a locally normalized *log-linear*) model or a support vector machine (SVM) [52] is used. Given the previous k actions up to state ϕ_i, the classifier $g : \Pi \times \Omega^k \rightarrow \mathbb{R}^{|\Omega|}$ assigns a score (or a probability) to each possible action. We note that state definition can encode the k previous actions which simplifies the score (which can be a probability) to $p_g(\omega_i|\pi_i)$. The score of the new state is then

$$p(\pi_{i+1}) = p_g(\omega_i|\pi_i) \cdot p(\pi_i) \tag{3.2}$$

In other words, the probability of a state can be computed simply as the product of the probabilities of each action in the sequence of actions that resulted in reaching that state from the initial state. Examples of transition-based dependency parsing with a SVM classifier can be found in [49, 50]. [43] uses a maximum-entropy based dependency parser for the CoNLL-X shared task on multilingual dependency parsing [53].

Throughout this dissertation, we use the maximum-entropy shift-reduce dependency parser[8] of [43]. The main reason to work with this parser is that the maximum-entropy classifier assigns real probabilities to the states during decoding which we will make a use of for the purpose of language modeling as we will see later in Section 3.3. In addition, the parser is fast and easy to implement.

[8]The constituent shift-reduce parsing version is proposed in [54]

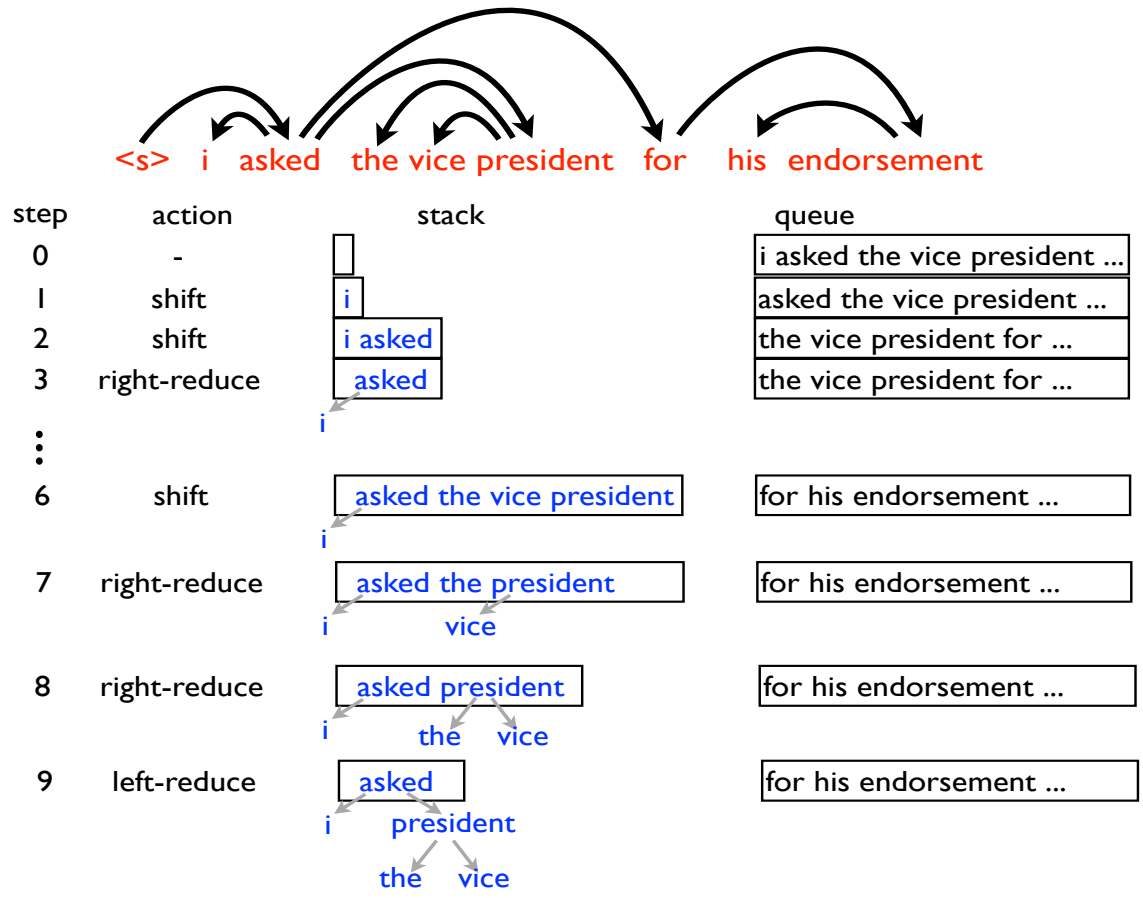


Figure 3.3: An example of shift-reduce dependency parsing actions.

3.2.2 Evalution Metrics

The most widely used metric for evaluating dependency parsing is **attachment score** which is the percentage of words whose heads (or governors) have been correctly predicted by the parser [55]. The parser quality can be evaluated using a single accuracy metric thanks to the single-head constraint of dependency trees. The attachment score can be unlabeled (only looking at heads) or labeled (looking at heads and labels) in which case it is called *unlabeled attachment score* (UAS) or *labeled*

attachment score (LAS), respectively.

3.3 Structured Language Modeling Using Dependency Parses

Syntactic information can be encoded in terms of headwords of phrases and their POS tags, extracted from a syntactic analysis of a sentence [5,9], such as a *dependency structure*. In fact, our motivating experiments in Section 2.3 showed that regular n-gram LMs are poor(er) at predicting the next words at positions where the headword is located deep in the history (not in the window of $(n-1)$-gram recent words). In this section, we describe our language modeling framework which incorporates such features using dependency structures.

We use the shift-reduce incremental dependency parser of [43], which constructs a tree from a transition sequence governed by a maximum-entropy classifier (described in details in Section 3.2.1). Shift-reduce parsing places input words into a queue Q and partially-built structures are organized in a stack S. An incremental parser can provide partial syntactic analyses of the history at each word position. Parser states corresponding to the ith word w_i are *history-states*, denoted $\Pi_{-i} = \{\pi^0_{-i}, \pi^1_{-i} \cdots, \pi^{K_i}_{-i}\}$, where K_i is the total number of such states (others may be eliminated via beam pruning.). These are the states which have word w_i as the top element of their queue.

That is,

$$\forall \pi \in \Pi_{-i} \quad q_0(\pi) = w_i.$$

Figure 3.4 depicts the history-states for each word position of our running example. We note that history-states at position i are generated, during decoding with the parser, without any knowledge of the words that appear after position i, They are merely derived from the structure in the history of the position. It is also worth mentioning that each π^j_{-i} is either a result of applying a `shift` action to a history-state $\pi^k_{-(i-1)}$ or one of two `reduce` actions to some other history-state $\pi^{j'}_{-i}$. Given the set of history-states $\Pi_{-i} = \{\pi^0_{-i}, \pi^1_{-i} \cdots, \pi^{K_i}_{-i}\}$, the probability assignment for w_i is,

$$p(w_i|W_{-i}) = \sum_{j=1}^{|\Pi_{-i}|} p\left(w_i|\phi(\pi^j_{-i}, W_{-i})\right) p_g(\pi^j_{-i}|W_{-i}) \tag{3.3}$$

where,

- W_{-i} : is the word history $w_1, \ldots, w_{i-1}$ at i^{th} position
- π^j_{-i} : the j^{th} history-state of position i
- $p_g(\pi^j_{-i}|W_{-i})$: is the normalized probability assigned to state π^j_{-i} by the parser. The normalization is applied over all the states in Π_{-i}.
- $\phi(\pi^j_{-i}, W_{-i})$: denotes an equivalence classification of word-history and parser history-state, capturing features from π^j_{-i} and W_{-i} that are most useful for predicting w_i.

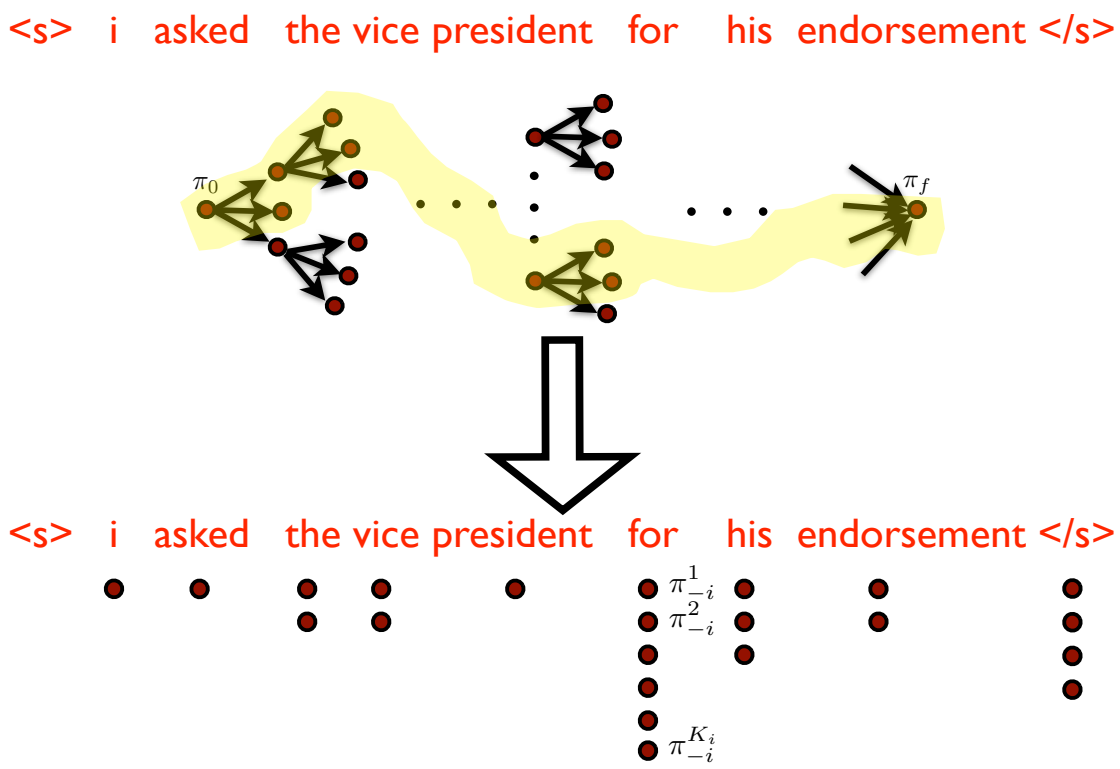


Figure 3.4: History-states corresponding to each position of an input sentence, after applying beam-pruning while parsing.

We emphasize that each π_{-i}^j and the corresponding $p_g(\pi_{-i}^j|W_{-i})$ are obtained by only looking[9] at the words W_{-i} in the history of position i. Therefore, the probability assignment in Eqn. 3.3 is a valid left-to-right probability which can be used for calculating PPL (This is referred to as the L2R-word-predictor in [5]).

$\phi(\pi_{-i}^j, W_{-i})$ indicates what type of information and features we want to extract from the history-states pair π_{-i}^j and W_{-i}. Due to the constraint that we can only

[9]In other words, the probability distribution imposed by the maximum-entropy classifier g of the parser is only due to features in the stack for extending each state and we drop all features extracted from the queue used in [43] for calculating this distribution (these features are called look-ahead features) . We will discuss this in more details later in Chapter 6, Section 6.7, and analyze its effect on the parser accuracy.

use features from the history for predicting word w_i, the equivalence classification $\phi(\pi^j_{-i}, W_{-i})$ is restricted to be based on $\{s_0 s_1 \cdots\}$, the partial trees in the stack. One example of such equivalence classification is,

$$\phi(\pi^j_{-i}, W_{-i}) = \text{h.w}_{s^j_0}\text{h.t}_{s^j_0} \tag{3.4}$$

where $\text{h.w}_{s^j_0}$ and $\text{h.t}_{s^j_0}$ indicate the word and POS tag, respectively, of the head of s^j_0, the top tree in the stack of π^j_{-i}. For example, in Figure 3.3, such a function applied to the parser state after step 8 yields the conditional probability $p(\texttt{for}|\texttt{president}, \texttt{NN})$ for predicting the word `for` (given that `president` has POS tag `NN`).

Given the choice of $\phi(.)$, the parameters of the model $p(w_i|f(\pi^j_{-i}, W_{-i})))$ are estimated to maximize the log-likelihood

$$\log P(\mathcal{T}) = \sum_i \log \left(\sum_{j=1}^{|\Pi_{-i}|} p\left(w_i|\phi(\pi^j_{-i}, W_{-i})\right) p_g(\pi^j_{-i}|W_{-i}) \right) \tag{3.5}$$

of the training data $\mathcal{T}$. It is easy to show that the objective function above can be maximized using a EM-style algorithm[10] [46] . The key to showing this is the fact that the objective function in Eqn. 3.5 can be thought of as a objective function induced by a hidden Markov model (HMM) with fixed transition probabilities $p_g(\pi^j_{-i}|W_{-i})$ and

[10] Although, there is no hidden (latent) variable (our parser parameters are fixed) as we have a separate trained dependency model for predicting the tree structures, one still can use the EM-style algorithm to iteratively approximate the objective function with an objective function that can be maximized with a closed form solution. The approximated objective function at each step has a form $\sum \log(.)$ instead of original form of $\log(\sum(.))$ and hence, easier to maximize [56].

Algorithm 1 Estimating conditional parameters using **Baum-Welch** algorithm

Initialization: set $p(\mathbf{w}|\boldsymbol{\phi}) = \frac{1}{|V|}$ for all $\mathbf{w} \in V$ and all possible $\boldsymbol{\phi}$.
do
  for each sentence $W \in \mathcal{T}$
    update expected counts as follows:

$$c(\mathbf{w}, \boldsymbol{\phi}) = \sum_{i:w_i=\mathbf{w}} \sum_{j:\phi(\pi^j_{-i}, W_{-i})=\boldsymbol{\phi}} \frac{p(w_i|\phi(\pi^j_{-i}, W_{-i}))p_g(\pi^j_{-i}|W_{-i})}{\sum_{j'} p(w_i|\phi(\pi^{j'}_{-i}, W_{-i}))p_g(\pi^{j'}_{-i}|W_{-i})} \tag{3.6}$$

  end for
  normalize expected counts: $p(\mathbf{w}|\boldsymbol{\phi}) = \frac{c(\mathbf{w},\boldsymbol{\phi})}{\sum_{w\in V} c(w,\boldsymbol{\phi})}$
until objective function in Eqn. 3.5 converges

HMM-state observation probabilities specified by $p\left(w_i|\phi(\pi^j_{-i}, W_{-i})\right)$ [5]. The steps for estimating the parameters of the model are shown in Algorithm 1.

To obtain better estimations for word probabilities and assign non-zero probability to unseen events, we use the **Jelinek-Mercer smoothing** technique [22]. In the following section, we describe the smoothing technique and the training procedure for building the final LM.

3.3.1 Jelinek-Mercer Smoothing

The *maximum likelihood* (ML) estimation procedure described in Section 3.3 tends to over-fit the data and *does not* generalize well (same as ML estimation of regular n-gram LMs); Indeed, any syntactic event $(\mathbf{w}, \mathbf{f})$ that is not observed in the training data would have a zero probability ($p(\mathbf{w}|\mathbf{f}) = 0$). To address this problem, *smoothing* is used to adjust the ML estimate of probabilities to more reliable (more robust)

probabilities [17]. There have been many smoothing techniques proposed for n-gram models such as Jelinek-Mercer [22], Katz [20], Witten-Bell [57] and Kneser-Ney [21]. While most of the mentioned smoothing techniques rely on the notation of n-gram *counts* (and hence, can not be used in our framework as we are dealing with *fractional counts* in Eqn. 3.6), the Jelinek-Mercer technique provides a flexible smoothing method which directly incorporates the estimates produced in Eqn. 3.6 .

Jelinek-Mercer smoothing for regular n-gram LMs is based on the idea that it is useful to interpolate higher-order n-gram models with lower-order n-gram models to account for the sparsity issue of estimated probabilities in the higher order LM [17]. We use the same idea for interpolating different type of equivalence classifications functions of history-states at each word position. Let

$$\phi_M(\pi_{-i}, W_{-i}) \rightarrow \phi_{M-1}(\pi_{-i}, W_{-i}) \rightarrow \cdots \rightarrow \phi_2(\pi_{-i}, W_{-i}) \rightarrow \phi_1(\pi_{-i}, W_{-i}) \qquad (3.7)$$

be a set of M different "fine-to-coarse" equivalence classification of history state π_{-i}. Jelinek-Mercer smoothing addresses zero probabilities (and in general unreliable estimates) in the probability estimates by the higher-order context ϕ_m, by recursive interpolation with probabilities estimated from a lower order context ϕ_{m-1} as:

$$\begin{aligned} p_{\text{interp}}(w_i|\phi_m(\pi_{-i}W_{-i})) &= \lambda_{\phi_m(\pi_{-i}W_{-i})} p_{\text{ML}}(w_i|\phi_m(\pi_{-i}W_{-i})) + \\ & \quad (1 - \lambda_{\phi_m(\pi_{-i}W_{-i})}) p_{\text{interp}}(w_i|\phi_{m-1}(\pi_{-i}W_{-i})), \end{aligned} \qquad (3.8)$$

for $1 \leq m \leq M$, where the 0-th order context is assumed to result in a uniform distribution:

$$p_{\text{interp}}(w_i|\phi_0(\pi_{-i}W_{-i})) = p_{\text{unif}}(w_i) = \frac{1}{|V|}.$$

A reasonable assumption in the scheme above is to have the interpolation order from fine to coarse equivalence classification functions. In other words, it is reasonable to assume that equivalence classification function $\phi_m(\pi_{-i}W_{-i})$ extracts finer information from the state π_{-i} than $\phi_{m-1}(\pi_{-i}W_{-i})$. One way to quantify this criterion is to constrain f_m to be better in terms of predicting the training data. That is,

$$\log P(\mathcal{T}|\phi_m) \geq \log P(\mathcal{T}|\phi_{m-1}) \tag{3.9}$$

where $\log P(\mathcal{T}|\phi_m)$ is the log-likelihood of the training data using ϕ_m equivalence classification of the history-states and is defined in Eqn. 3.5.

Given fixed $p_{\text{ML}}(.)$ for each equivalence classification function ϕ_m, coefficients $\lambda_{\phi_m(\pi_{-i}W_{-i})}$ are estimated on a held-out set using the Baum-Welch algorithm [56]. This held-out data need to be different form the training data used for estimating $p_{\text{ML}}(.)$, otherwise the criterion in Eqn. 3.9 forces the coefficients of the finest equivalence classification function $\lambda_{\phi_M(\pi_{-i}W_{-i})}$ to be one and the rest to be zero. One way to get such heldout data is to reserve a section of the training data for this purpose, where this heldout data is not used in calculating the $p_{\text{ML}}(.)$.

We note that because the number of parameters $\lambda_{\phi_m(\pi_{-i}W_{-i})}$ can be very large, estimating each $\lambda_{\phi_m(\pi_{-i}W_{-i})}$ independently would require a huge amount of held-out data. Jelinek and Mercer [22] proposed tying them by dividing $\lambda_{\phi_m(\pi_{-i}W_{-i})}$ into a reasonable number of *buckets*. We use the bucketing algorithm suggested in [58] which is based on the idea that those $\lambda_{\phi_m(\pi_{-i}W_{-i})}$ should be tied together that we believe should be similar [17]. Specifically, Bahl [58] suggests bucketing the $\lambda_{\phi_m(\pi_{-i}W_{-i})}$'s based on the count of $\phi_m(\pi_{-i}W_{-i})$'s in the training data, as this count should intuitively correlate with how much weight should be assigned to the finer higher-order equivalence classification function ϕ_m. In our case, the history-state $\phi(\pi_{-i}^j)$ is a latent variable with probability assignment $p_g(\pi_{-i}^j|W_{-i})$. The count of $\boldsymbol{\phi}$, a specific context features $\phi_m(\pi_{-i}W_{-i})$ in the training data, can be therefore calculated as

$$c(\boldsymbol{\phi}) = \sum_i \sum_{j:\phi_m(\pi_{-i}^j W_{-i})=\boldsymbol{\phi}} p_g(\pi_{-i}^j|W_{-i}). \tag{3.10}$$

We make the assumption that whether a bucket is large enough for robust parameter estimation on held-out data depends only on the expected count of the features whose λ parameters fall in the bucket. The expected count of training tokens in a bucket B is therefore

$$E_{\text{heldout}}[|B|] = \sum_{\boldsymbol{\phi}:\lambda_{\boldsymbol{\phi}}\in B} c_{\text{heldout}}(\boldsymbol{\phi}) \tag{3.11}$$

$c_{\text{heldout}}(\boldsymbol{\phi})$ is calculated analogous to Eqn. 3.10, but on held-out data and $E_{\text{heldout}}[|B|]$

refers to expected count of coefficients falling in the bucket, calculated on held-out data. The steps of building the buckets for tying λ_{ϕ_m}'s are as follows:

- Create a special bucket for λ_{ϕ_m}'s that have never occurred in the training data, i.e. $c(\boldsymbol{\phi_m}) = 0$. We make an exception for this bucket and set all λ_{ϕ_m}'s in this bucket to be zero, as there is no information in the higher-order distribution[11].
- Start from the lowest observed value of $c(\boldsymbol{\phi_m}) > 0$, and put increasing values of $c(\boldsymbol{\phi_m})$ into the same bucket B^m until $E_{\text{heldout}}[|B|]$ exceeds some threshold $c_{\min}$.
- Repeat the previous step until all λ_{ϕ_m}'s are bucketed.
- If for the last bucket $E_{\text{heldout}}[|B|] < c_{\min}$, then merge it with the preceding bucket.

We perform the bucketing steps above for each order $\phi_1, \phi_2, \cdots, \phi_M$ separately and estimate the bucketed $\lambda_{c(\phi_m)}$'s using the Baum-Welch algorithm [56] to maximize the likelihood of the held-out data.

Given the hierarchical equivalence classification functions $\phi_1, \phi_2, \cdots, \phi_M$ and estimated λ_{ϕ_m}'s, the word probability assignment in Eqn. 3.3 is replaced with,

$$p(w_i|W_{-i}) = \sum_{j=1}^{|\Pi_i|} p_{\text{interp}}\left(w_i|\phi_M(\pi_{-i}^j, W_{-i})\right) p_g(\pi_{-i}^j|W_{-i}) \tag{3.12}$$

where $p_{\text{interp}}\left(w_i|\phi_M(\pi_{-i}^j, W_{-i})\right)$ is defined recursively using Eqn. 3.8 (λ_{ϕ_m} is replaced with $\lambda_{c(\phi_M)}$ due to bucketing).

As discussed in Section 3.3, we restrict the equivalence classification functions ϕ_m

[11] In other words, $p_{\text{interp}}(w_i|\phi_m(\pi_{-i}W_{-i})) = p_{\text{interp}}(w_i|\phi_{m-1}(\pi_{-i}W_{-i}))$ for such cases.

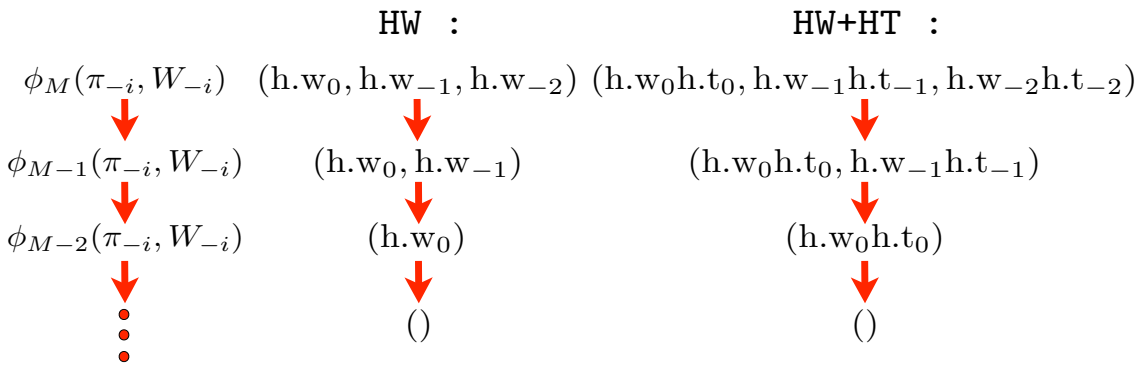


Figure 3.5: Examples of hierarchal interpolation schemes.

to only look at features in the stack of the history-states π_{-i}. Furthermore, we will only use headword and headtag features of partial trees in the stack throughout this dissertation. We emphasize that this is mainly because it has been observed in studies that these features carry most of the information for word prediction [5, 59].

To further simplify the notation, we use $\mathrm{h.w}_0, \mathrm{h.w}_{-1}, \cdots$ ($\mathrm{h.t}_0$,$\mathrm{h.t}_{-1}$,$\cdots$) to denote the headwords (headtags) of partial trees in the stack of a history-state π_{-i}^j of a word position i, respectively. We use two hierarchical interpolation schemes from [5] (Figure 3.5) for smoothing equivalence classification of history-states by looking only at the headwords (`HW`) and head-tags (`HW+HT`) of the first 3 partial trees in the stack. "()" refers to an equivalence classification where no information is used from history (uni-grams). Compared to [5], we use a deeper context for both (`HW`) and (`HW+HT`).

3.3.2 Preliminary Results

Before proceeding to our new methods, we establish a baselined perplexity of the SLM using the two hierarchical schemes shown in Figure 3.5. We apply these schemes to two different tasks, namely *Broadcast News* (BN) and *Wall Street Journal* (WSJ). The following summarizes the setup used for each task:

- `BN` **setup** : EARS BN03 corpus, which has about $42M$ words serves as our training text. We also use rt04 ($45K$ words) as our evaluation data. Finally, to interpolate our structured language models with the baseline 4-gram model, we use rt03+dev04f (about $40K$ words) data sets to serve as our development set. The vocabulary we use in the `BN` experiments has about $84K$ words.
- `WSJ` **setup** : The training text consists of about $37M$ words. We use dev93 ($9K$ words) to serve as our development set for interpolating SLMs with the baseline 4-gram model and eval92+eval93 ($10K$ words) as our evaluation set.

In both cases, we hold out an additional $20K$ sentences from the training text to use as our heldout data for applying the bucketing algorithm and estimating λ's. To apply the dependency parser, all the data sets are first converted to Treebank-style tokenization and POS-tagged[12] using the tagger of [60]. Both the POS-tagger and the shift-reduce dependency parser are trained on the Broadcast News treebank from Ontonotes [1] and the WSJ Penn Treebank (after converting them to dependency

[12]Again, to make sure we have a proper LM, the POS-tagger only uses features from history to tag a word position. All lookahead feature in [60] are dropped.

trees) which consists of about $1.2M$ tokens. Finally, we train the modified Kneser-Ney 4-gram LMs on the tokenized training texts to serve as our baseline LM, for both experiments.

Tables 3.1 and 3.2 show the perplexity measurements for the `BN` and `WSJ` experiments, respectively. As seen in the tables, headtags in the equivalence classifications used in the `HW+HT` interpolation significantly improve the perplexity of the SLM. It also improves the performance of the interpolated LM. However, neither `HW` nor `HW+HT` match KN performance 4-gram on their own. To improve over the baseline 4-gram LMs, the SLMs need to be interpolated with the corresponding 4-gram LMs.

LM		Dev	SLM Weight	Eval
baseline 4-gram		165	0.00	158
SLM	`HW`	183	1.00	174
	`HW+HT`	174	1.00	163
Interp.	`HW`	154	0.35	149
	`HW+HT`	**150**	0.43	**144**

Table 3.1: The perplexity of different `BN` language models.

LM		Dev	SLM Weight	Eval
baseline 4-gram		149	0.00	121
SLM	`HW`	165	1.00	140
	`HW+HT`	155	1.00	132
Interp.	`HW`	138	0.33	116
	`HW+HT`	**135**	0.42	**114**

Table 3.2: The perplexity of different `WSJ` language models.

3.4 Improved Hierarchical Interpolation

The original SLM hierarchical interpolation scheme is aggressive in that it drops both the tag and headword from the history. However, in many cases the headword's tag alone is sufficient, suggesting a more gradual coarsening of the context. We propose a new interpolation where instead of dropping both the headword and headtag of a partial tree in the stack of a history-state at each level of the `HW+HT` hierarchical interpolation scheme, we first drop only the headword. E.g., in the `HW+HT` hierarchy of Figure 3.5, we use $(\text{h.w}_0\text{h.t}_0, \text{h.w}_{-1}\text{h.t}_{-1}, \text{h.w}_{-2}\text{h.t}_{-2}) \rightarrow (\text{h.w}_0\text{h.t}_0, \text{h.w}_{-1}\text{h.t}_{-1}, \text{h.t}_{-2}) \rightarrow (\text{h.w}_0\text{h.t}_0, \text{h.w}_{-1}\text{h.t}_{-1})$.

Keeping the headtag adds more specific information and at the same time is less sparse. We refer to this interpolation hierarchy as $\texttt{HW+HT}_2$. A similar idea is found, e.g., in the back-off hierarchy of n-gram language model [61] where instead of backing off from the n-gram to the $(n-1)$-gram, a more gradual backoff — by considering a hierarchy of fine-to-coarse classes for the furthest word in the history— is used. To empirically show that the added levels to the hierarchy of $\texttt{HW+HT}_2$ indeed provides an advantage, we compare the log-likelihood of training data using the added $(\text{h.w}_0\text{h.t}_0, \text{h.w}_{-1}\text{h.t}_{-1}, \text{h.t}_{-2})$ level to those of the levels before and after the added level. The average log-likelihood of training data using the equivalence classifications above are show in Table 3.3. It can be observed that the new added level of hierarchy has significantly better predictive power, at least on training data, than the one which simultaneously drops the tag and headword from the history.

Equivalence Classification	Average Log-likelihood (PPL)
$(\text{h.w}_0\text{h.t}_0, \text{h.w}_{-1}\text{h.t}_{-1}, \text{h.w}_{-2}\text{h.t}_{-2})$	-0.8337 (7)
$(\text{h.w}_0\text{h.t}_0, \text{h.w}_{-1}\text{h.t}_{-1}, \text{h.t}_{-2})$	-1.0903 (12)
$(\text{h.w}_0\text{h.t}_0, \text{h.w}_{-1}\text{h.t}_{-1})$	-1.3249 (21)

Table 3.3: The average log-likelihood of different equivalence classification on training data.

LM	Eval	Dev
BN		
SLM HW+HT$_2$	159	168
Interp. HW+HT$_2$	**142**	**147**
WSJ		
SLM HW+HT$_2$	125	149
Interp. HW+HT$_2$	**110**	**132**

Table 3.4: The perplexity performance of the LMs using the proposed HW+HT$_2$ interpolation scheme.

Table 3.4 presents the perplexity results for both BN and WSJ tasks. it confirms that the new scheme, when compared with Tables 3.1 and 3.2, improves both interpolated and non-interpolated models.

3.5 Pruning the SLM

A significant drawback of the SLM is its final size. Since the latent state π_i can take several values, and since the transition from π_i to π_{i+1} results in multiplicative growth in the number of possibilities, an exponential growing number of possible history-states are hypothesized during parsing. In comparison, when building a regular n-gram model at each position, there will be only $n-1$ added contextual features. Let us clarify this point through the example of Figure 3.4. In this example at the word

position `for` there are 6 history-states (that have survived the beam). Assuming the interpolation scheme of `HW`, Figure 3.5, 18 (6×3) can be potentially added to the final LM at the word position `for` .

This ultimately prevents us from being able to train linguistically motivated SLMs on a huge amount of text data. To address this issue, one can apply one of the following solutions:

- **More aggressive parsing beam-pruning**: One way to reduce the size of a SLM, is aggressive pruning of the parser states during training. This can be performed by reducing the beam-threshold in the beam-pruning strategy or, as it is suggested in [5], by explicitly limiting the maximum number of parser states (the latter pruning strategy is referred to as maximum-stack-depth pruning in [5]).
- **Pruning strategy on the LM level**: Another way to reduce the size of a SLM — and make it comparable in size with n-gram LMs — is to wait till all the training text is processed and prune the trained model $p_{\text{interp}}\left(w_i|\phi_M(\pi^j_{-i}, W_{-i})\right)$.

In order to measure the effect of the first solution on the quality of trained LM, one would need to train a new LM each time a pruning parameter of the parser is changed and hence, it can be a tedious procedure to optimize pruning parameters to satisfy the tradeoff between LM performance and size. The second solution, on the other hand, works directly on the LM level. Therefore, the effect of pruning on the resulting model can be measured without the need to re-train the model. In addition, since the pruning is applied directly on the SLM parameters (as we will

see later in Section 3.5.1), the pruning takes into account the accumulated training statistics, as opposed to the first strategy which prunes parser states locally for each training sentence. In other words, there might be equivalence classifications which are pruned locally by the parser—because they have a low prior probability assigned by the parser—they, however, might occur enough times in the training data and therefore, they might capture a useful information for language modeling.

We propose to investigate the latter solution in this dissertation. We start in Section 3.5.1 by proposing a general information-theoretic strategy for pruning Jelinek-Mercer LMs. We then present the results and the effect of our pruning method on the size of the SLM and its performance in Section 3.5.2.

3.5.1 Relative Entropy Pruning of the Jelinek-Mercer LM

Increasing the size of training data, usually improves the performance of a language model. However, as the size of training data increases, LM size increases, which can lead to models that are too large to be practically applicable. This is more severe for our SLM due to the reasons mentioned in Section 3.5.

Here, we present an algorithm for pruning the underlying Jelinek-Mercer LM. Our algorithm is based on the pruning scheme proposed in [45], which was used for pruning n-gram backoff language models. The goal is is to minimize the *distance*

between the distribution implied by the original model and that by the pruned model. As mentioned in [45], the proposed algorithm has desired properties such as,

- It is an information-theoretic based criterion directly related to minimizing the difference between the language modeling performance of the original and pruned model.
- It is linear in size of the original model and hence, practically efficient.
- It is performed by merely considering the distribution encoded by the language model. In other words, there is no need to have access to the raw data used to train the model.

Let us first revisit the structure of a Jelinek-Mercer LM. The m^{th}-level model in a hierarchical Jelinek-Mercer interpolated LM consists of the following parts:

1. $\boldsymbol{\phi_m} \in \{\phi_m\}$: The m^{th}-level contextual features which are observed in the training data. We will refer to these as "context-features".

2. $c(\boldsymbol{\phi_m})$: The (empirical expected) count of the context-feature $\boldsymbol{\phi_m}$ in training. These counts are used for bucketing and tying λ's.

3. $\lambda_{\boldsymbol{\phi_m}}$: The interpolation weights which are functions of $c(\boldsymbol{\phi_m})$ and are used to interpolate m^{th}-level ML probabilities with the $(m-1)^{th}$-level probabilities.

4. $p_{\mathrm{ML}}\left(w|\boldsymbol{\phi_m}\right)$: Explicit maximum-likelihood estimates of the probability of a word w given the context-feature $\boldsymbol{\phi_m}$.

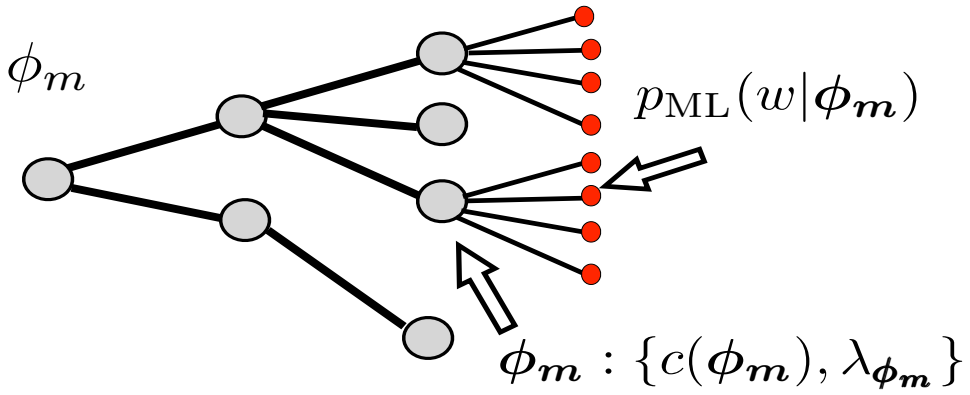


Figure 3.6: The data structure used for representing m^{th} level parameters of a Jelinek-Mercer LM.

Using the notation developed above, the probability of a word w given the highest order context-feature $\boldsymbol{\phi_m}$ is calculated through the recursive interpolation scheme:

$$p_{\mathrm{interp}}(w|\boldsymbol{\phi_m}) \quad = \quad \lambda_{c(\boldsymbol{\phi_m})} p_{\mathrm{ML}}\left(w|\boldsymbol{\phi_m}\right) + (1 - \lambda_{c(\boldsymbol{\phi_m})}) p_{\mathrm{interp}}(w|\boldsymbol{\phi_{m-1}}) \tag{3.13}$$

for $1 \leq m \leq M$, where $\boldsymbol{\phi_M} \rightarrow \boldsymbol{\phi_{M-1}} \rightarrow \cdots \rightarrow \boldsymbol{\phi_1}$ represents the context-features used in each level of the hierarchical interpolation scheme. Figure 3.6 depicts the data structure used for representing m^{th} level parameters of a Jelinek-Mercer LM. The context-features $\boldsymbol{\phi_m}$ and probabilities $p_{\mathrm{ML}}\left(w|\boldsymbol{\phi_m}\right)$ are efficiently represented using a *trie*, which is a prefix-tree data structure. We refer to pre-leaf and leaf nodes in the trie as context-nodes and probability-nodes, respectively.

Our goal is to reduce the number of parameters in the Jelinek-Mercer LM by removing explicit context-features and their corresponding explicit estimated proba-

bilities. For a given m^{th} level context-feature $\mathbf{f_m}$, let

$$\mathcal{S}\left(\boldsymbol{\phi_m}\right) = \{w | w \in V \text{ s.t } p_{\text{ML}}\left(w|\boldsymbol{\phi_m}\right) > 0\} \tag{3.14}$$

represent all the words in the vocabulary for which there exist an explicit estimated probability given the context-feature, i.e. $p_{\text{ML}}\left(w|\boldsymbol{\phi_m}\right) > 0$. In our pruning scheme, we want to remove those $\boldsymbol{\phi_m}$'s and their corresponding explicit probabilities $p_{\text{ML}}\left(w|\boldsymbol{\phi_m}\right)$ which have the least impact on the LM distribution. Following [45], we use *Kullback-Leibler* (KL) divergence [62] (also known as relative-entropy) to measure the distance between the original and pruned LM. Let $p_{\text{interp}}(.|.)$ denote the conditional probabilities encoded by the original Jelinek-Mercer LM and $p'_{\text{interp}}(.|.)$ the probabilities of the pruned model. The KL divergence between the two models — using m^{th} level probabilities — is,

$$\begin{aligned} D(p||p') &= \sum_{\phi_m} \hat{p}(\phi_m) D\left(p(.|\phi_m)||p'(.||\phi_m)\right) \\ &= \sum_{\phi_m, w} \hat{p}(\phi_m)\, p_{\text{interp}}(w|\phi_m) \log \frac{p_{\text{interp}}(w|\phi_m)}{p'_{\text{interp}}(w|\phi_m)}, \end{aligned} \tag{3.15}$$

where $\hat{p}(\phi_m) = \frac{c(\phi_m)}{\sum_{\phi'_m} c(\phi_m)}$ is the empirical probabillty of a context-feature ϕ_m.

Ideally, we should minimize Eq. 3.15 over all possible subsets of $\boldsymbol{\phi_m}$ that are candidates for removal, but this is computationally intractable. Instead, we assume that each $\boldsymbol{\phi_m}$ effects $D(\cdot)$ independently and greedily prune those with the smallest

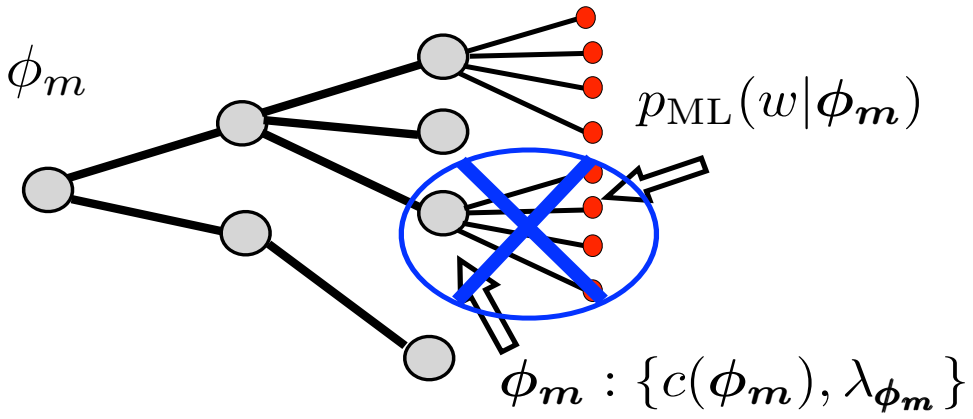


Figure 3.7: Deleting a context-node and all of its explicit probability-nodes from the m^{th} trie of a Jelinek-Mercer LM.

impact. It is shown in [45] that relative change in the model perplexity is related to $D(p||p')$ through

$$\frac{\mathrm{PPL}(\mathrm{p}'_{\mathrm{interp}}) - \mathrm{PPL}(\mathrm{p}_{\mathrm{interp}})}{\mathrm{PPL}(\mathrm{p}_{\mathrm{interp}})} = \exp\left(D(p||p')\right) - 1 \tag{3.16}$$

In the proposed algorithm, we prune $\boldsymbol{\phi_m}$'s that results in an increase of relative perplexity (3.16) by less than a threshold.

Let us now show mathematically how to compute $D(p||p')$ from pruning a single $\boldsymbol{\phi_m}$. Figure 3.7 pictorially shows the effect of removing a context-node $\boldsymbol{\phi_m}$ and all its leaf-nodes $w \in \mathcal{S}\left(\boldsymbol{\phi_m}\right)$. Removing $\boldsymbol{\phi_m}$ changes the model(s) as follows:

(i) For $w \in \mathcal{S}\left(\boldsymbol{\phi_m}\right)$, the m^{th}-level probability is completely replaced by the lower-level interpolated model. That is,

$$p_{\mathrm{interp}}(w|\boldsymbol{\phi_m}) \;=\; \lambda_{c(\boldsymbol{\phi_m})} p_{\mathrm{ML}}\left(w|\boldsymbol{\phi_m}\right) + (1 - \lambda_{c(\boldsymbol{\phi_m})}) p_{\mathrm{interp}}(w|\boldsymbol{\phi_{m-1}}),$$

$$p'_{\text{interp}}(w|\boldsymbol{\phi_m}) \;=\; p_{\text{interp}}(w|\boldsymbol{\phi_{m-1}}). \tag{3.17}$$

This is because when the context-feature $\boldsymbol{\phi_m}$ is removed, it is as if it has not been observed in the training data and $c(\boldsymbol{\phi_m})$ becomes zero, forcing $\lambda_{c(\boldsymbol{\phi_m})} = 0$.

(ii) For $w \notin \mathcal{S}(\boldsymbol{\phi_m})$, $p_{\text{ML}}(w|\boldsymbol{\phi_m})$ is zero, so that

$$\begin{aligned} p_{\text{interp}}(w|\boldsymbol{\phi_m}) &= \lambda_{c(\boldsymbol{\phi_m})} p_{\text{ML}}(w|\boldsymbol{\phi_m}), \\ p'_{\text{interp}}(w|\boldsymbol{\phi_m}) &= p_{\text{interp}}(w|\boldsymbol{\phi_{m-1}}). \end{aligned} \tag{3.18}$$

(iii) Since we assume $c(\boldsymbol{\phi_m}) = 0$ after removing the context-node $\boldsymbol{\phi_m}$, the buckets used to tie λ's—and therefore, the maximum-likelihood estimates of λ's on the heldout data—will also be affected. This could affect $p'_{\text{interp}}(w|\boldsymbol{\phi'_m})$ even for $\boldsymbol{\phi'_m} \neq \mathbf{f_m}$ and is hence computationally difficult to measure. Therefore, we use an iterative algorithm: each iteration measures only the changes in $D(p||p')$ due to effects (i) and (ii), ignoring (iii). After pruning candidate $\boldsymbol{\phi_m}$'s, we run the bucketing algorithm and re-estimate the λs for the new pruned LM on held out data.

By ignoring (iii) all estimates not involving context-feature $\boldsymbol{\phi_m}$ remain unchanged, so the KL divergence resulting from removing $\boldsymbol{\phi_m}$ is:

$$D(p||p') = \hat{p}(\boldsymbol{\phi_m}) \sum_{w \in V} p_{\text{interp}}(w|\boldsymbol{\phi_m}) \log \frac{p_{\text{interp}}(w|\boldsymbol{\phi_m})}{p'_{\text{interp}}(w|\boldsymbol{\phi_m})} \tag{3.19}$$

Furthermore, the sum above over all the words in the vocabulary, can be split into

two parts as

$$D(p||p') = \hat{p}(\boldsymbol{\phi_m}) \left(\sum_{w \in \mathcal{S}(\boldsymbol{\phi_m})} p_{\text{interp}}(w|\boldsymbol{\phi_m}) \log \frac{p_{\text{interp}}(w|\boldsymbol{\phi_m})}{p'_{\text{interp}}(w|\boldsymbol{\phi_m})} + \sum_{w \notin \boldsymbol{\phi_m}} p_{\text{interp}}(w|\boldsymbol{\phi_m}) \log \frac{p_{\text{interp}}(w|\boldsymbol{\phi_m})}{p'_{\text{interp}}(w|\boldsymbol{\phi_m})} \right) \tag{3.20}$$

For the first and second sum we use probabilities in Eq. 3.17 and Eq. 3.18 respectively. The above thus simplifies to

$$\begin{aligned} D(p||p') &= \hat{p}(\boldsymbol{\phi_m}) \left(\sum_{w \in \mathcal{S}(\boldsymbol{\phi_m})} p_{\text{interp}}(w|\boldsymbol{\phi_m}) \log \left[\lambda_{c(\boldsymbol{\phi_m})} \frac{p_{\text{ML}}(w|\boldsymbol{\phi_m})}{p_{\text{interp}}(w|\boldsymbol{\phi_{m-1}})} + (1 - \lambda_{c(\boldsymbol{\phi_m})}) \right] \right. \\ &\quad \left. + \left(1 - \sum_{w \in \mathcal{S}(\boldsymbol{\phi_m})} p_{\text{interp}}(w\boldsymbol{\phi_m}) \right) \log(1 - \lambda_{c(\boldsymbol{\phi_m})}) \right) \end{aligned} \tag{3.21}$$

Note that a single iteration over all $w \in \mathcal{S}(\boldsymbol{\phi_m})$ is needed to calculate both the first sum in Eq. 3.21 and $\sum_{w \in \mathcal{S}(\boldsymbol{\phi_m})} p_{\text{interp}}(w|\boldsymbol{\phi_m})$.

Algorithm 2 summarizes our proposed pruning procedure. In each pass, we iterate over all context-features, from level M to $M_{\min}$, and remove nodes for which $(e^{D(p||p')} - 1) < \frac{\theta}{T}$. Dividing the threshold θ by T, the number of passes, reduces pruning in each iteration and distributes pruning across T passes, where we re-estimate λ's after each pass to take into account changes due to (iii) above.

Algorithm 2 Relative-entropy pruning of Jelinek-Mercer LM

Input:
θ: pruning threshold
$M_{\min}$:minimum hierarchical level for pruning
T: number of iterations

Algorithm:
for *iter* in $1..T$ **do**
  for m in $M..M_{\min}$ **do**
    for $\boldsymbol{\phi_m} \in \phi_m$ **do**
      calculate $D(p||p')$ due to removing $\boldsymbol{\phi_m}$ according to Eqn. 3.21
      if $e^{D(p||p')} - 1 < \frac{\theta}{T}$
        prune $\boldsymbol{\phi_m}$ and make $c(\boldsymbol{\phi_m}) = 0$
    end for
  end for
  if ***anything pruned***
    recompute buckets and λ's on the held-out data (as described in Section 3.3.1)
  else exit
end for

3.5.2 Pruning Experiments and Results

We evaluate the effect of pruning using the setup described in Section 3.3.2. We apply pruning to our best model, the Jelinek-Mercer SLM with $\texttt{HW+HT}_2$, with $M = 7$, $T = 5$ and $M_{\min} = 4$. The lowest hierarchy level that we considered for pruning is $(\text{h.w}_0\text{h.t}_0, \text{h.t}_{-1})$. We prune the non-interpolated SLM and then interpolate it with the Kneser-Ney 4-gram model. Our goal is to reduce the SLM to a comparable size to the baseline model, which stores $39M$ n-grams in 1.7GB memory in `BN` experiments and stores $43M$ n-grams in 1.9GB memory in `WSJ` setup.

Pruning reduces the size of the `BN` SLMs significantly, as seen in Table 3.5. SLM size drops from 12.5GB to 2.8GB without much loss in test-set perplexity. In fact, when interpolated with the Kneser-Ney 4-gram, the Dev-data perplexity is almost

Threshold	Eval	Dev	# Param.	Mem. Size
SLM				
0	159	168	413.5 M	12.5 GB
1×10^{-7}	159	175	143.3 M	4.4 GB
5×10^{-7}	161	183	90 M	2.8 GB
Interpolated				
0	142	147	(413.5+39) M	(12.5+1.7) GB
1×10^{-7}	141	148	(143.3+39) M	(4.4+1.7) GB
5×10^{-7}	141	149	(90+39) M	(2.8+1.7) GB

Table 3.5: Preplexity and memory size of the HW+HT$_2$ SLMs for the BN task.

Threshold	Eval	Dev	# Param.	Mem. Size
SLM				
0	125	149	495M	15.2GB
1×10^{-7}	129	153	147.2M	4.3GB
5×10^{-7}	135	158	100.1M	3.2GB
Interpolated				
0	110	132	(495+43)M	(15.2+1.9)GB
1×10^{-7}	111	133	(147.2+43)M	(4.3+1.9)GB
5×10^{-7}	112	134	(100.1+43)M	(3.2+1.9)GB

Table 3.6: Preplexity and memory size of the HW+HT$_2$ SLMs for the WSJ task.

unchanged. For the Eval-data, the performance is slightly better, possibly due to over-fitting of the unpruned LM. Table 3.6 confirms that our pruning scheme is also effective for WSJ experiments. Once again, SLM size drops significantly while the performance of the interpolated LM is retained. We now compare our pruning method with the pruning method that was used in the original SLM framework, where parser states are pruned locally. Table 3.7 shows the perplexity results of the HW+HT$_2$ SLM using the dependency parser in deterministic mode, where we only capture the best state among states with same number of actions during parsing a sentence. The deterministic dependency parser is essentially a parser with a beam where only the

LM	Eval	Dev	# Param.	Mem. Size
BN				
SLM	167	182	154.2M	4.7 GB
Interp. SLM	145	151	(154.2+39) M	(4.7+1.7) GB
WSJ				
SLM	135	159	175.9 M	5.4 GB
Interp. SLM	114	136	(175.9+39) M	(5.4+1.9) GB

Table 3.7: Preplexity and memory size of the $\texttt{HW+HT}_2$ SLMs using deterministic dependency parser for the BN and WSJ tasks.

top state is retained in the beam. Comparing the perplexity results in Table 3.7 with those of Tables 3.5 and 3.6, it can be observed that our pruning method, which is applied directly on the LM level as described in Section 3.5, results in better perplexity performance. As an example the BN SLM using the deterministic parser requires about 4.7GB of memory and has perplexity of about 167 on the evaluation set. Our punning method with $\theta = 5 \times 10^{-7}$, on the other hand, requires only 2.8GB of memory and has a better perplexity of about 161.

3.6 Analysis

The main motivation behind using syntactic structure in our SLM is the fact that n-gram models *can not* learn long-distance dependencies and are shown to be problematic for predicting word positions where syntactic relations go beyond recent words in the history (Section 2.3). In this section, we aim to empirically show that, in fact, the syntactic dependencies modeled through our SLM parameterization improves the predictive power of the LM in those problematic regions, i.e. *syntactically-distant*

positions[13]. To this end, we calculate the following (log) probability ratio for each position in the test data,

$$\log \frac{p_{\text{KN+SLM}}(w_i|W_{-i})}{p_{\text{KN}}(w_i|W_{-i})}, \tag{3.22}$$

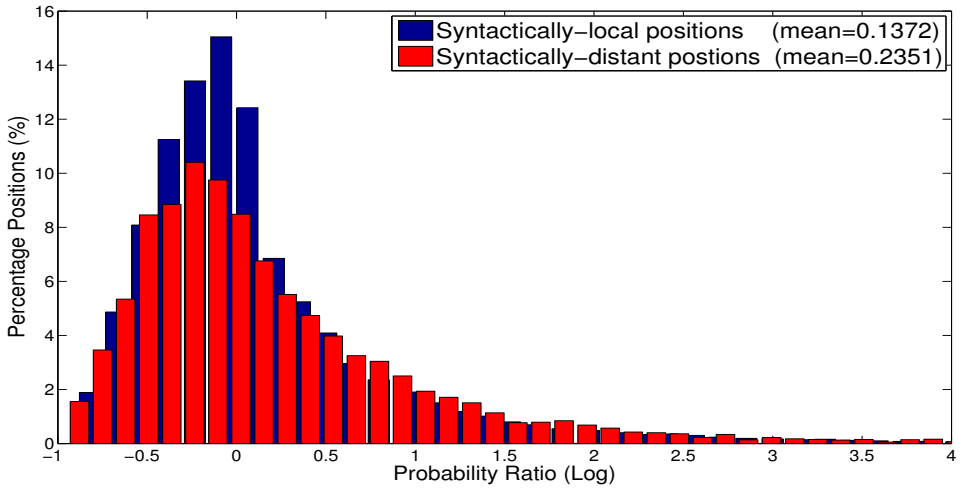
where p_{SLM} is the word probability assignment of SLM with $\texttt{HW+HT}_2$ interpolation scheme at each position and $p_{\text{ngram}}(w_i)$ is the probability assigned by the baseline 4-gram model. The quantity above measures the improvement (or degradation) gained as a result of using the SLM parameterization[14].

Figures 3.8(a) and 3.8(b) illustrate the histogram of the probability ratio (3.22) for all the word positions in evaluation data of `BN` and `WSJ` tasks, respectively. In these figures the histograms for *syntactically-distant* and *syntactically-local* positions are shown separately to measure the effect of the SLM for the two position types. For both tasks the percentage of positions with $\log \frac{p_{\text{KN+SLM}}(w_i|W_{-i})}{p_{\text{KN}}(w_i|W_{-i})}$ near zero is much higher for *syntactically-local* (blue bars) than the *syntactically-distant* (red bars). To confirm this, we calculate the average $\log \frac{p_{\text{KN+SLM}}(w_i|W_{-i})}{p_{\text{KN}}(w_i|W_{-i})}$ —this is the average log-likelihood improvement, which is directly related to perplexity improvement— for each position type in the figures.

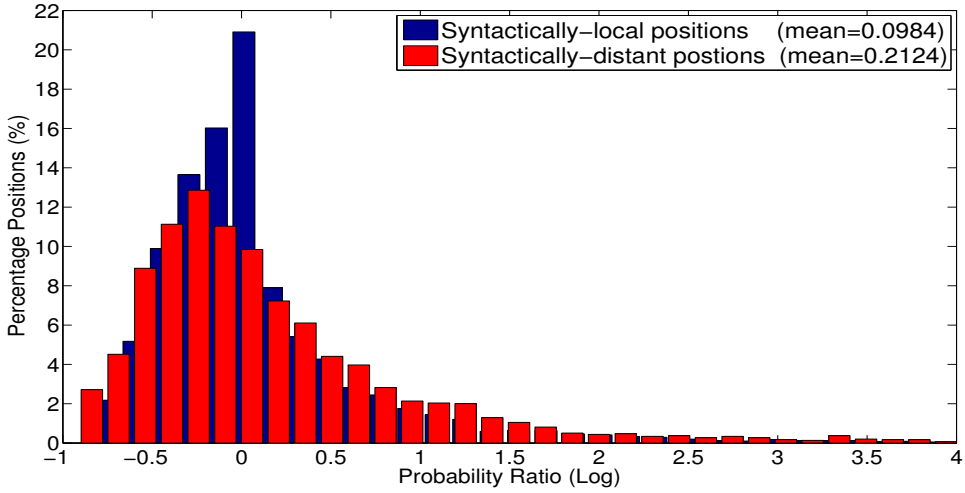
Table 3.8, reports the perplexity performance of each LM (baseline 4-gram, SLM

[13]In other words, we want to show that perplexity improvements shown in Section 3.4 are mainly due to improving the predictive power in those problematic regions.

[14]If $\log \frac{p_{\text{KN+SLM}}(w_i|W_{-i})}{p_{\text{KN}}(w_i|W_{-i})}$ is greater than zero, then we can conclude that the SLM has a better predictive power for word position w_i. This is a meaningful comparison due to the fact that the probability assignment using both SLM and n-gram is a proper probability (which sums to one over all words at each position).



(a) BN



(b) WSJ

Figure 3.8: Probability ratio historgram of SLM to 4-gram model for (a) BN task (b) WSJ task.

and interpolated SLM) on different positions of the evaluation data for BN and WSJ tasks. As observed, the use of long-span dependencies in the SLM partially fills the gap between the performance of the basleline 4-gram LM on *syntactically-distant* postions $\mathcal{N}$ versus *syntactically-local* positions $\mathcal{M}$. In addition, it can be seen that

Test Set Position	4-gram	SLM	4-gram + SLM
	BN		
$\mathcal{M}$	146	152	132
$\mathcal{N}$	201	182	171
$\mathcal{N}+\mathcal{M}$	158	159	142
$\frac{PPL_{\mathcal{N}}}{PPL_{\mathcal{M}}}$	138%	120%	129%
	WSJ		
$\mathcal{M}$	114	120	105
$\mathcal{N}$	152	141	131
$\mathcal{N}+\mathcal{M}$	121	125	110
$\frac{PPL_{\mathcal{N}}}{PPL_{\mathcal{M}}}$	133%	117%	125%

Table 3.8: Perplexity on the BN and WSJ evaluation sets for the 4-gram LM, SLM and their interpolation. The SLM has lower perplexity than the 4-gram in *syntactically distant* positions $\mathcal{N}$, and has a smaller discrepancy $\frac{PPL_{\mathcal{N}}}{PPL_{\mathcal{M}}}$ between preplexity on the distant and local predictions, complementing the 4-gram model.

the SLM itself fills the gap substantially, however, due to its underlying parameterization based on Jelinek-Mercer smoothing it has a worse performance on regular *syntactically-local* positions (which account for the majority of the positions) compared to the Kneser-Ney smoothed LM[15]. Therefore, to improve the overall performance, the interpolated SLM takes advantage of both the better modeling performance of Kneser-Ney for *syntactically-local* positions and the better features included in the SLM for improving predictive power on *syntactically-distant* positions.

[15]This is merely due to the superior modeling power and better smoothing of the Kneser-Ney LM [17].

3.7 Conclusion

In this chapter, we presented useful extensions to the original structured language modeling framework of Chelba and Jelinek [5] to make it more accurate and space efficient. A simple hierarchal interpolation scheme, which improves the perplexity of the SLM, is introduced. Furthermore, a general information-theoretic pruning algorithm is developed to reduce the model size in the Jelinek-Mercer LM smoothing framework. Our experiments showed that the proposed SLM significantly improves over the state-of-the-art modified Kneser-Ney 4-gram LMs. Additionally, our analysis confirmed that the syntactic dependencies modeled through the proposed SLM parameterization is improving the predictive power of the LM in the position where the baseline 4-gram LM is weak, i.e. positions where syntactic relations go beyond recent words in the history.

Chapter 4

Hill Climbing on Speech Lattices: an Efficient Rescoring Framework

4.1 Introduction

In chapter 3, we introduced the SLM framework and showed that the equivalence classification of the histories based on the dependency structures resulted in a LM which improves over regular n-gram models in terms of perplexity. Our proposed SLM is one of the many statistical approaches that aim to address the weakness of n-gram history classification in representing linguistic facts by using long-distance contextual dependencies as well as the syntactic structure of the word string. In fact, due to the availability of large amounts of training data and computational resources during the past decade, building more complex models with sentence level knowledge

and longer dependencies has been an active area of research in automatic speech recognition (ASR) [5, 6, 9]. Yet, due to the complexity of the speech recognition task and its large search space, integration of many of these complex and sophisticated knowledge sources into the first decoding pass is not feasible. Many of these long-span models cannot be represented as weighted finite-state automata (WFSA), making it difficult even to incorporate them in a direct lattice rescoring pass. Even if they can be represented as WFSA, the direct lattice rescoring under them usually requires an extensive state expansion which can be potentially problematic in terms of memory requirements. Instead, an *N-best rescoring* strategy is employed to (partially) realize their superior modeling power.

There have been many works on long-span language modeling, besides SLMs [5, 8, 44]: Whole sentence exponential language models are used in [15, 63]. [64] uses a recurrent neural network framework to capture long-span contextual information. A non-incremental, head-driven statistical parser is used in [7] to produce word sequence probabilities. A maximum-entropy based language model which uses both syntactic and topic information is proposed in [6]. A constraint dependency grammar and a finite-state tagging model were used by [65] to exploit syntactic dependencies. Yet, most of these language models are used in an N-best rescoring framework, due to the reasons mentioned in above.

N-best rescoring suffers from several known deficiencies and inefficiencies. As a thought experiment, one could sort all the hypotheses in a lattice using the simpler

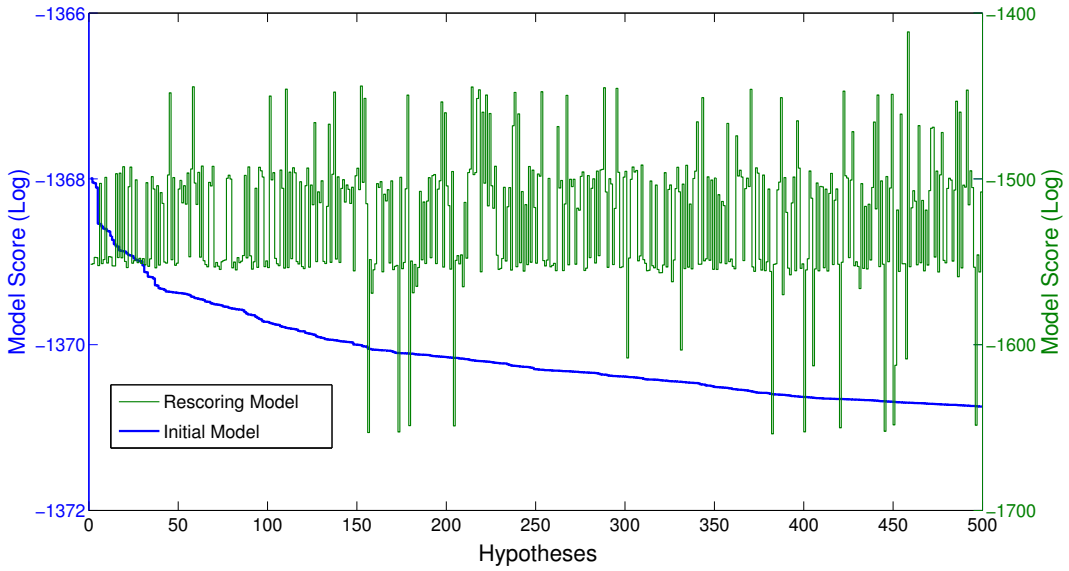


Figure 4.1: N-best hypotheses (under an initial model) and their corresponding score using a new rescoring model for an example speech utterance.

lattice-generating models, rescore each with the more complex models and ask where the highest-scoring word sequence *ranks*. N-best rescoring will suffer from search errors if N is smaller than this rank. But while considering a large number of hypotheses mitigates search errors, it is often computationally expensive, especially for complex models—with sentence level knowledge and longer dependency information—for which evaluating each hypothesis is computationally costly. The solution is to use the more complex models to aid hypothesis selection, as opposed to considering the top N hypotheses chosen by the simpler models. Figure 4.1 illustrates the problem with N-best rescoring for an example speech utterance. In this figure the x-axis corresponds to the 500-best solutions (hypotheses) under some initial model where these hypotheses are ordered based on their rank under the initial model score. The y-axis

denotes the model scores of each of these hypotheses using the initial model and a rescoring model, respectively. As seen from this figure, there is not a strong correlation between the rank of a hypothesis under the initial model and its rank under the new model[1]. Moreover, the best scoring hypothesis (in this set of hypotheses) under the new model is ranked around 450^{th} using the initial model, and to reach that hypothesis sequentially, one needs to evaluate hypotheses that score poorly under the new model. Looking at Fig 4.1, one may ask if it is necessary to evaluate the hypotheses ranked 1 to 450 sequentially, or whether, having found a hypothesis that scores well under the new model, one could "jump" in the N best list to a similar hypothesis and check if its new model score is even higher. In this manner, one could leap-frog to the 450^{th} hypothesis without necessarily scoring all the preceding ones.

In this chapter, we propose a **hill climbing** technique that operationalizes the idea above. Hill climbing examines a neighborhood of an initial point, finds the direction in which the score under the long-span LM is increasing most steeply, moves a suitable step in this direction to reach a new point, and iterates. For a broad class of problems, hill climbing is guaranteed to reach a local maximum of the function.

Let us summarize the steps of a hill climbing algorithm as a general *greedy search*

[1]We emphasize that the example provided here by no means represents all possible examples in a speech corpus and is just serving as an illustrative example. There may be utterances for which the correlation between the rank of hypotheses under the initial model and their rank under the new model is high. In fact, if one plots the same figure but instead averages the scores across many utterances in a corpus, it may be observed that there is a high correlation between the ranks using initial model and the new model in average. Here, we are addressing the problem with N-best rescoring on individual utterance basis. The N-best rescoring although is effective in average, one could provide a much more efficient rescoring framework that dynamically adapts to each single utterance.

algorithm:

1. **Initialize** to a starting point in the search space.
2. Examine a **neighborhood** of the current solution.
3. Take a step in the direction where the **search function increases**.
4. **Iterate** using the new selected solution, until the current solution attains a local maximum of the search function.

To apply hill climbing ideas to lattice rescoring with complex models, we therefore need to view the alternatives in the lattice as points in a domain-space equipped with a notion of a neighborhood. A natural domain is to consider each word sequence in the lattice as a point, and the *edit distance* (Levenshtein distance) between two word sequences as the metric for defining neighborhoods. Starting with a word sequence in the lattice, we evaluate hypotheses in a small neighborhood with the complex model, step to the best one, and iterate, until a hypothesis scores higher than all the alternatives in its neighborhood.

Hill climbing for ASR has been investigated previously in [66] for rescoring confusion networks [67], which provide posterior probabilities for individual words in the hypotheses based on all the lattice-generating models. Scores from the complex models may be combined with these (total) probabilities during rescoring. Rescoring confusion networks, while very efficient, has the disadvantage of losing some information in the lattice, such as timing information and individual acoustic and LM score.

The technique presented here evaluates explicit paths in the original lattice, permitting greater flexibility, e.g. for discriminatively combining the individual models used in lattice generation and rescoring [68]. Also, as a result of working on speech lattices, the timing information and acoustic scores (encoded in the lattice) can be directly used by the rescoring model (if needed).

The rest of this chapter develops the hill climbing idea. Section 4.2 describes our hill climbing technique, and efficient construction of the neighborhoods using finite-state automata (FSA) techniques. We then discuss few directions for extending the proposed algorithm in Section 4.3 followed by Section 4.4 which addresses the issue of non-convex optimization, i.e. avoiding local maxima. Sections 4.5 and 4.6 describe our experimental setup and results where we empirically show the effectiveness of the proposed hill climbing algorithm. In Section 4.7, we present speech recognition experiments using the SLM in conduction with the proposed hill climbing algorithm to fully and efficiently realize the power of the long-span SLM.

4.2 Hill Climbing on Speech Lattices

The ASR output from the first decoding pass is typically encoded as a lattice: a directed acyclic graph (DAG) with unique start and end nodes, nodes time-stamped w.r.t. the speech signal A, and edges labeled with words w. Each path in the DAG from start to end corresponds to a candidate time-aligned transcription $W =$

$w_1\, w_2\, \ldots\, w_n$ of A. Each edge in the DAG is also labeled with component model scores, e.g. log-probabilities from the acoustic model $P(A|W)$ and language model $P(W)$. For simplifying the notation and to emphasize the generality of our algorithms, we use Γ and Λ to represent the acoustic model and the language model, respectively. The acoustic model probably and language model probability are denoted by $P(A|W,\Gamma)$ and $P(W|\Lambda)$, respectively. The DAG structure respects the conditional independence assumptions of the component models, so that the score $g(A,W;\Gamma,\Lambda)$ of an entire path is the sum of the scores of the edges along the path. A lattice, generated by the first pass of decoding, encodes an astronomically large number of hypotheses in a compact representation.

Our goal is to investigate the replacement of the lattice-generating language model $P(W|\Lambda)$ with a long-span language model $P(W|\Lambda_{\text{new}})$[2]. We will use hill climbing to find the path W in the lattice that maximizes the new total score,

$$g(A,W;\Gamma,\Lambda_{\text{new}}) = \gamma \log P(A|W,\Gamma) + \log P(W|\Lambda_{\text{new}}), \tag{4.1}$$

where $P(A|W,\Gamma)$ denotes the acoustic score of the path corresponding to word sequence W and $P(W|\Lambda_{\text{new}})$ is the new language model score of W. γ is the inverse of the language model (LM) scaling factor. Assuming hypothesis W is generated by taking the path $d = e_1 e_2 \cdots e_n$ in the lattice, where the e_i's represent the corresponding

[2]We emphasize that the technique we develop in this chapter can also be used for rescoring speech lattices with long-span sentence level acoustic score. However, we only focus on using long-span language models.

edges, the acoustic score of W may be decomposed as,

$$\log P(A|W,\Gamma) = \sum_{i=1}^{n} s_{\text{acoustic}}(e_i) \tag{4.2}$$

where $s_{\text{acoustic}}(e_i)$ is the acoustic score of the i^{th} edge in the path.

In order to apply hill climbing to our problem, we need to define a *neighborhood* for each word sequence W in the lattice. Since our search space consists of a set of word sequences, it is natural to define the neighborhood function using the *edit distance* metric.

Specifically, we define the neighborhood of W at position i to be all the paths in the lattice whose word sequence may be obtained by editing — deleting, substituting or inserting a word to the left of — w_i. We will use $\mathcal{N}(W,i)$ to denote this "distance 1 at position i" neighborhood of W. We explain in Section 4.2.1 how the set of paths in $\mathcal{N}(W,i)$ can be efficiently generated from the representation of the lattice as a finite-state automaton (FSA) using the intersection operation.

We propose to undertake hill climbing as follows. At each step of the algorithm, a position i in the current word sequence W is selected. All paths in the lattice corresponding to word sequences $W' \in \mathcal{N}(W,i)$ are then extracted, along with the original acoustic scores on each edge. The new scores $g(A,W';\Gamma,\Lambda_{\text{new}})$ are computed and the hypothesis $\hat{W}'(i)$ with the highest score becomes the new W. A new position i in the new W is then selected[3] and the process continues, until $W = \hat{W}'(1) = \ldots =$

[3]We have experimented with selecting i sequentially from left to right at successive steps, return-

$\hat{W}'(n)$. In other words, the search terminates when W itself is the highest scoring hypothesis in its 1-neighborhood at all positions in i.

Section 4.2.2 described the hill climbing algorithm formally, and Section 4.2.3 discusses alternative ways of selecting the position i.

4.2.1 Efficient Generation of Neighborhoods

As mentioned above, we need to efficiently generate the neighborhood set of a given word sequence W at a specific position i.

To this end, note that the set of all word sequences that can be generated from W with one deletion, insertion or substitution at position i can be represented by a unweighted FSA. We will call this machine $LC(W, i)$. Before we show how to construct this machine, let us first introduce two special arcs in a FSA:

- **epsilon arc** (ϵ): This arc can be matched to to any input symbol and taking it, does not consume any symbol from the input alphabet. It is used to represent situations where we need to delete a position from an input word sequence.

- **sigma arc** (σ): This is a special arc which is used to represent all possible input symbols of a FSA. In other words, this arc can be matched with any symbol from the input alphabet (any w in the vocabulary) and consumes one symbol from the input. It is only used to efficiently account for situations where we need to have an arc for each input symbol in the alphabet.

ing to the beginning (i=1) from the end of W.

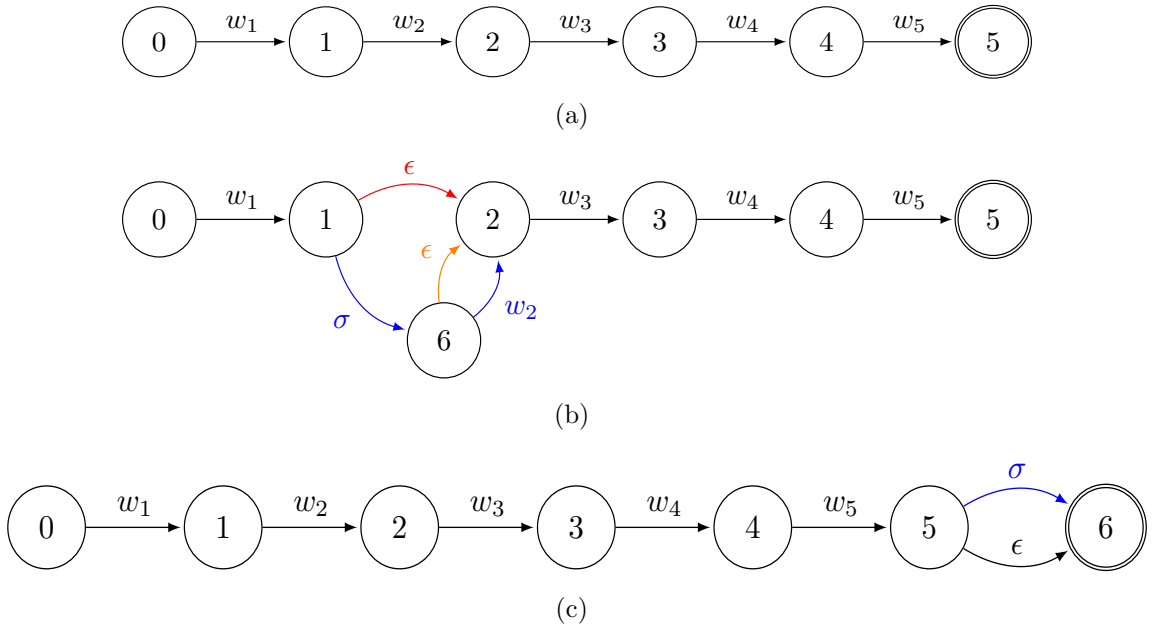


Figure 4.2: The FSA representation of the neighborhood set of a given path: (a) Original path. (b) Neighborhood for a specific position. (c) Neighborhood for the last position.

Using these special arcs, Figure 4.2 illustrates the construction of the FSA $LC(W, i)$. A word sequence $W = w_1\, w_2 \ldots w_5$ is represented as a FSA in Figure 4.2(a) and Figure 4.2(b) represents $LC(W, 2)$, the **distance 1 neighbors** of W at position 2 . The ϵ arc $(1 \to 2)$ accounts for the neighbor with w_2 deleted, the arc with σ followed by ϵ $(1 \to 6 \to 2)$ corresponds to substituting w_2 with any word, including w_2 itself, i.e. W always belongs to $\mathcal{N}(W, i)$, and the path with σ followed by w_2 corresponds to one insertion to the left of w_2. Since the decision to insert only to the left of a position i is arbitrary, we also define the FSA $LC(W, n+1)$, which permits insertions to the right of the last word in W. $LC(W, 6)$ for the example above is shown in Figure 4.2(c).

Next, to restrict the set of all the word strings represented by $LC(W, i)$ to word sequences actually present in the lattice (our search space), $LC(W, i)$ is intersected with a **weighted** FSA representation of the lattice, $\mathcal{L}_{\text{acoustic}}$, the weights on arcs are the scaled acoustic scores, $\gamma s_{\text{acoustic}}(e)$, from the initial lattice:

$$LN(W, i) \leftarrow LC(W, i) \circ \mathcal{L}_{\text{acoustic}}. \tag{4.3}$$

$LN(W, i)$ therefore is a weighted FSA representation of the subset of word sequences W' in $\mathcal{N}(W, i)$ that are also present in the initial lattice. The weights associated with the words in $LN(W, i)$ are the acoustic scores in the original lattice, which we will need for combining with the new language model scores $\log P(W'|\Gamma_{\text{new}})$ according to (4.1). Note also that although $\mathcal{L}_{\text{acoustic}}$ could be a huge WFSA, the intersection is fast and efficient because $LC(W, i)$ is (practically) deterministic, and has a very small number of states. Figure 4.3 illustrates an example of an intersected $LN(W, i)$ (with lattice) where the current word sequence W is "`AND CERTAINLY AND THAT IS WHY IT IS HARD`" and neighborhood hypotheses are generated for the 4-th position ($i = 4$).

4.2.2 The Hill Climbing Algorithm

We now introduce the initial version of our algorithm formally using the notation developed previously. The three major steps are shown in Algorithm 3.

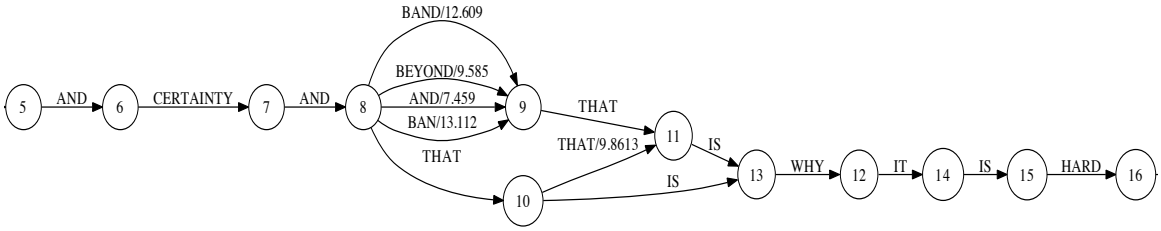


Figure 4.3: An example of $LN(W, i)$, word sequences present in lattice with edit-distance 1 to W at position i.

- **Initialization**, where the highest scoring word sequence (the Viterbi path) from the initial lattice is selected;
- **Neighborhood Generation**, discussed in Section 4.2.1;
- **Neighborhood Rescoring**, which involves evaluating all the word sequences in the neighborhood set using (4.1), and selecting the word sequence with the maximum score for the next step of the algorithm.

It is worth mentioning that our algorithm is based on a variation of hill climbing, called *steepest ascent* hill climbing, in which the best possible move is made at each step, among all possible moves (in our case the best path after rescoring by the new model among paths in the neighborhood set). In the basic hill climbing algorithm, any move in the neighborhood which improves the objective function may be made, not necessarily the move with the most improvement.

Remark. The advantage of the proposed hill climbing algorithm is that we always have a full hypothesis during rescoring the neighborhood. This enables us to include

Algorithm 3 Steepest Ascent Hill Climbing for Rescoring

$\mathcal{L}_{\text{acoustic}} \leftarrow$ **WFSA** of the initial (first-pass) lattice
$W \leftarrow$Viterbi path of the initial lattice [**Initilization**]
$N \leftarrow$ **length**(W)
repeat
  $i \leftarrow 1$ [Start at position 1]
  while $i \leq N + 1$ **do**
    [**Neighborhood generation**]
    create **FSA** $LC(W, i)$
    $LN(W, i) \leftarrow LC(W, i) \circ L_{\text{acoustic}}$
    [**Neighborhood rescoring according to Eqn. 4.1**]
    $W \leftarrow \arg\max_{W' \in LN(W,i)} g(A, W'; \Gamma, \Lambda_{\text{new}})$
    [Adjust length and position]
    $N \leftarrow$**length**(W)
    if DELETION **then**
      $i \leftarrow i$-1
    end if
    $i \leftarrow i$+1
  end while
until W does not change [Stopping criterion]

features and model components that are **sentence-level** and capture global properties. In other words, there is no need to be restricted by the Markov assumption. Also, the proposed algorithm — as it is generally true for any hill climbing search method— is an **anytime method**. That is, we can stop the rescoring procedure if we are satisfied with the current solution or are bound by time constraints, e.g. a maximum of 20 seconds for rescoring per utterance.

4.2.3 Choosing Positions to Explore Via Hill Climbing

Once the neighborhood $\mathcal{N}(W, i)$ of the current position in W has been explored and W has been updated, the hill climbing algorithm has many choices for the next position (the new i) to explore. We have made the expedient choice of $i \leftarrow i + 1$ in Algorithm 3, i.e. the choice to explore the neighborhood(s) of W sequentially from left to right. We emphasize that this is by no means the only way to choose i, and possibly not even the optimal choice. For the local optimality of hill climbing, it suffices that the search terminates only if W is the highest scoring hypothesis in *all* its i neighborhoods. A possibly smarter choice of i may be based on obtaining the lattice entropy at each w_i. Positions of high entropy indicate higher uncertainty about the choice of w_i, and positions of likely errors in W [69]. Attacking such positions first, with other positions fixed, will likely focus the rescoring effort where it is most needed, and result in even faster convergence to the highest scoring hypothesis.

4.3 Extended Hill-Climbing Algorithm

The effectiveness and efficiency of the proposed hill climbing algorithm (like any other greedy local search algorithm) depends on several aspects, such as the neighborhood structure and initialization. In this section, we extend the hill-climbing algorithm described in Section 4.2.2 to make it more efficient while keeping it ef-

fective and accurate, in terms of avoiding search errors using the complex model. In Section 4.3.1, we introduce an extension to the original neighborhood definition (described in Section 4.2) to address the issue with consecutive errors in the output of a speech recognizer. In Section 4.3.2, we then introduce the proposed heuristic to make the algorithm more efficient, using the information provided by the initial lattice generating model. Finally, Section 4.3.3 presents the extended hill climbing algorithm.

4.3.1 Expanding the Neighborhood

An ASR system tends to make *consecutive errors*. This is mainly due to the fact that a erroneous word (region) causes errors in the neighboring words through affecting the context (which essentially affects the ability of the context dependent models to correctly predict neighboring words)[4]. Correcting these consecutive errors by applying a hill climbing algorithm that uses the 1 edit distance neighborhood definition, requires the new model to correct the errors one-by-one. This implies that the search algorithm can reach the final corrected solution (the one which has all consecutive errors corrected) *only if* the followings are satisfied simultaneously[5]:

(i) The target solution is *reachable* through 1-edit distance neighborhoods. That is, there exists a set of solutions $W^0, W^1, W^2, \cdots W^K$ in the search space (lattice)

[4]As an example, there are two errors per out-of-vocabulary (OOV) word occurrence [70].

[5]It is worth mentioning that these conditions are, in general, required for reaching to any desired target solution by applying the hill climbing algorithm.

where W^0 is the initial solution and W^K is the target solution and

$$W^m \in \mathcal{N}(W^{k-1}, k) \tag{4.4}$$

for $1 \leq k \leq K$. We call these solutions intermediate reachable solutions for target W^K.

(ii) There exists a set of intermediate reachable solutions on which the rescoring model scores better, consecutively. In other words, there exists an intermediate reachable solution set $W^0, W^1, W^2, \cdots W^K$ where,

$$g(A, W^k; \Gamma, \Lambda_{\text{new}}) > g(X, W^{k-1}; \Gamma, \Lambda_{\text{new}}) \tag{4.5}$$

for $1 \leq k \leq K$.

Considering consecutive errors in the ASR output, there are situations where even though condition (i) is satisfied, the new model has worse scores on the intermediate point (with consecutive edit distance 1) and hence, the errors can not be corrected. In other words, we need to first walk down the hill and then walk up (a different hill) to reach to the correct solution. Let us illustrate this point through the following example which shows an ASR 1-best output for an utterance and its corresponding

reference transcription:

Reference: YEAH FIFTY CENT GOT A NOMINATION WHICH WAS GREAT

ASR output: YEAH FIFTY CENT A GALLON NOMINATION WHICH WAS GREAT

One way to reach to the correct solution in the above example, is to first insert GOT to the left of A GALLON and then, to delete GALLON. That is the following intermediate solutions:

(1): YEAH FIFTY CENT GOT A GALLON NOMINATION WHICH WAS GREAT

(2): YEAH FIFTY CENT GOT A -- NOMINATION WHICH WAS GREAT

Assuming this is the only way in the lattice to reach the correct solution (reference), it is very likely that $g(1) < g(\text{ASR output})$ using the new language model but $g(2) > g(\text{ASR output})$. Therefore, even though the final solution scores better under the new model, it can not be reached through 1-edit distance neighbors with increasing scores.

We mitigate this problem by extending the definition of the neighborhood of W at position i, denoted by $\mathcal{N}(W, i)$, to include all the paths in the lattice which are within **edit distance 2** w.r.t to W. Firgure 4.4(a) shows how $LC(W, i)$ FSA corresponding to edit distance 2 definition can be constructed. This FSA basically represents the following word-sequences:

1. $w_1 w_4 w_5$ (deleting both w_2 and w_3)
2. $w_1 \sigma w_3 w_4 w_5$ (deleting w_2 and inserting to the left of w_3)
3. $w_1 \sigma w_4 w_5$ (deleting w_2 and substituting w_3)
4. $w_1 \sigma w_2 w_4 w_5$ (insertion to the left of w_2 and deleting w_3)
5. $w_1 \sigma w_2 \sigma w_4 w_5$ (insertion to the left of w_2 and substituting w_3)
6. $w_1 \sigma w_2 \sigma w_3 w_4 w_5$ (insertion to the left of both w_2 and w_3)
7. $w_1 \sigma \sigma w_4 w_5$ (substituting both w_2 and w_3)
8. $w_1 \sigma \sigma w_3 w_4 w_5$ (substituting w_2 and inserting to the left of w_3)

It is easy to see that word-sequences $w_1 \sigma w_4 w_5$ are represented, by $LC(W, 2)$ in Figure 4.4(a), through two distinct paths. This makes the FSA non-efficient when intersecting it with the lattice. Figure 4.4(b) shows the FSA representation of $LC(W, 2)$, in which the duplication above is avoided, hence making it more efficient for the intersection operation. Extending the structure of the neighborhood to include word sequences with edit distance 2 w.r.t to the current solution increases the *reachability* of the good solutions (based on the new model) in the search space (lattice) and as we show later in Section 4.6, it significantly improves the effectiveness of the search algorithm.

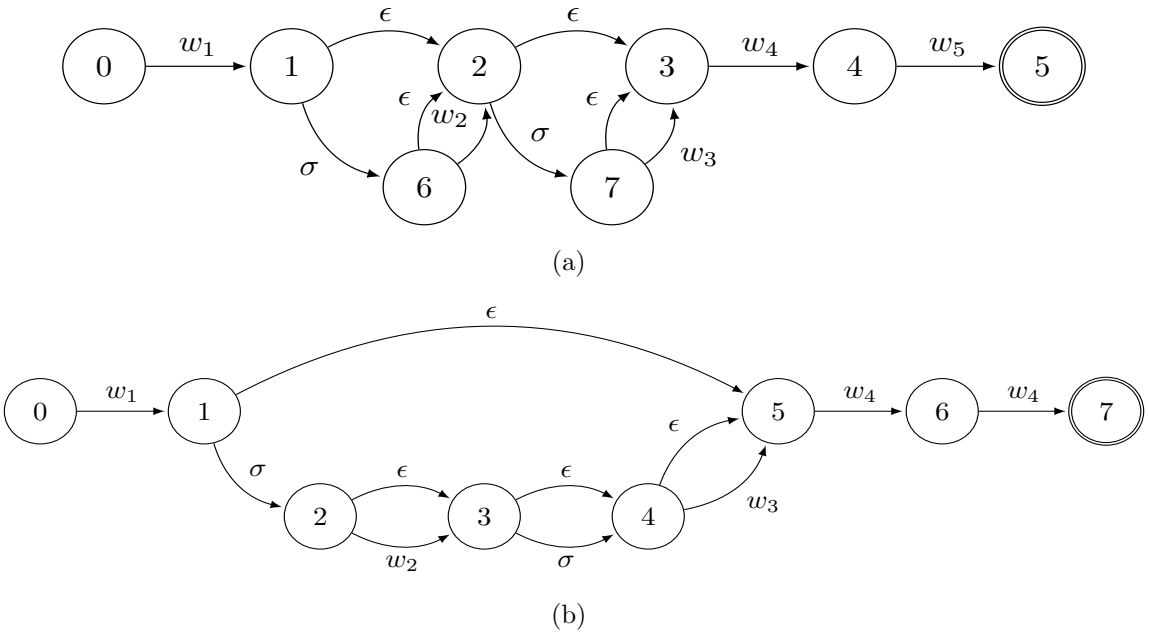


Figure 4.4: The FSA representations of the edit distance 2 neighborhood set.

4.3.2 Pruning the Neighborhood

The proposed hill climbing algorithm presents an efficient rescoring framework by using the more complex models to aid hypotheses selection for rescoring and to iteratively/locally improve an initial solution. Yet, rescoring all the paths in the neighborhood, using a complex long-span model, is computationally expensive and can lead to a waste of resources as some of the paths (hypotheses) in the neighborhood might have originated deep in the lattice and have scored poorly according to the baseline ASR score. To increase the efficiency of the algorithm, we propose reducing the number of paths in the neighborhood by **pruning** using one of the following strategies:

- **Histogram pruning**: limits the number of hypotheses, per neighborhood, to the top ones based on the initial model score.
- **Beam pruning**: only considers paths in the neighborhood which are within a *threshold* of the best path according to the initial model.

In this dissertation, we apply the **beam pruning** strategy as it makes the search algorithm focus the rescoring effort on **high-entropy** regions (under the initial model). In fact, it has been noted that having a high entropy region in the ASR output highly correlates with erroneous regions [69], and many methods use the entropy as a strong feature for predicting ASR errors [69, 71, 72]. Given a threshold θ and the neighborhood $LN(W, i)$ for the current solution W at position i, we define the pruned neighborhood Prune$(LN(W, i), \theta)$ to only include word sequence $W' \in LN(W, i)$ for which,

$$g^* - g(A, W'; \Gamma, \Lambda_{\text{old}}) \leq \theta. \tag{4.6}$$

Here, $g(A, W'; \Gamma, \Lambda_{\text{old}})$ denotes the original ASR score (combined acoustic score and the initial language model score) for W' and g^* is the best hypothesis in the neighborhood according the the initial score. That is, $g^* = \max_{W' \in LN(W,i)} g(A, W'; \Gamma, \Lambda_{\text{old}})$. Also, to ensure that the objective function *does not* decrease at any step of the algorithm, we avoid pruning the current solution W from the neighborhood even if $g^* - g(A, W; \Gamma, \Lambda_{\text{old}}) > \theta$, i.e. we always keep the current solution in the pruned

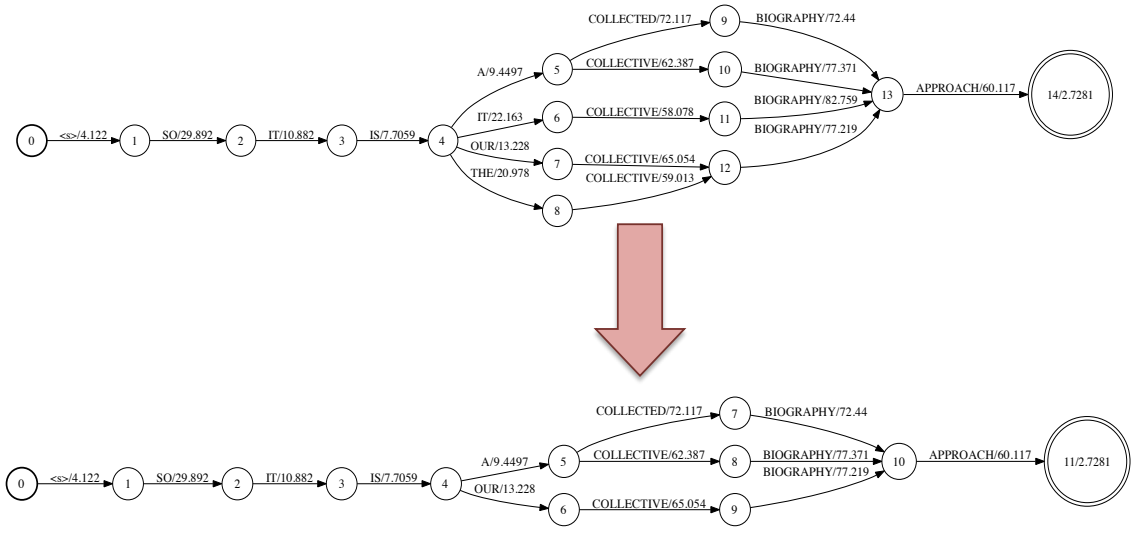


Figure 4.5: Pruning the weighted FSA representation of a neighborhood $LN(W, i)$.

neighborhood. Figure 4.5 shows the effect of applying this pruning scheme on the FSA representation of a neighborhood, which has been generated using edit distance 2 neighborhood definition. In Section 4.6, we show experimentally that the proposed beam pruning is very effective and can significantly reduces the computational complexity of the search algorithm without introducing significant *search errors*.

4.3.2.1 The Lexicographic Semiring

The proposed pruning scheme uses the total initial score from the ASR output — which is the combined acoustic and initial language model scores — in the pruning criterion. However, we need to have access to acoustic scores by themselves to combine them with the new language model score $P(W|\Lambda_{\text{new}})$ according to the Eqn. 4.1.

To utilize the pruning, we use **lexicographic** weights [73] in the wighted FSA

representation of the original lattices. A $\langle W_1, W_2 \rangle$-lexicographic weight is a tuple of two weights where each of the weight classes, W_1 and W_2, must observe the *path property* [74] that is: $a \oplus b$ is equal to either a or b for any pair a and b in the weight class. In this work, we use a pair of *tropical weights* to get the $\langle T, T \rangle$-lexicographic weight (T denotes the class of tropical weights). Given the real weights a_1, b_1, a_2 and b_2 the $\oplus$ and $\otimes$ operations — the intersection operation uses the $\otimes$ (times) operation to get the weights of the resulting weighted FSA — for $\langle T, T \rangle$-lexicographic semiring is defined as follows:

$$\begin{aligned} \langle a_1, b_1 \rangle \otimes \langle a_2, b_2 \rangle &= \langle a_1 + a_2, b_1 + b_2 \rangle \\ \langle a_1, b_1 \rangle \oplus \langle a_2, b_2 \rangle &= \begin{cases} \langle a_1, b_1 \rangle & \text{if } a_1 < a_2 \text{ or } (a_1 = a_2 \ \& \ b_1 < b_2) \\ \langle a_2, b_2 \rangle & \text{otherwise} \end{cases} \end{aligned} \tag{4.7}$$

As can be seen, the lexicographic semiring has the *path property* and therefore, the *natural order* is defined as follows:

$$\langle a_1, b_1 \rangle < \langle a_2, b_2 \rangle \quad \text{iff} \quad \langle a_1, b_1 \rangle \oplus \langle a_2, b_2 \rangle = \langle a_1, b_1 \rangle \tag{4.8}$$

We use $\mathcal{L}_{\langle g_{\Gamma,\Lambda_{\text{old}}}, g_\Gamma \rangle}$ to denote the $\langle T, T \rangle$-lexicographic weighted FSA representation of the lattices, where the first weight corresponds to the initial total score $g_{\Gamma,\Lambda_{\text{old}}}$ and the second weight is the acoustic score g_Γ. The $\langle T, T \rangle$-lexicographic weighted

neighborhood FSA, $LN(W, i)$, is then constructed as follows,

$$LN(W, i) \leftarrow LC(W, i) \circ \mathcal{L}_{\langle g_{\Gamma,\Lambda_{\text{old}}}, g_{\Gamma}\rangle}.$$

where the intersection uses the $\otimes$ operation defined in Eqn. 4.7. Therefore, we have access to both the total initial score and the acoustic score in the new $LN(W, i)$. Given the natural order definition and the threshold θ the pruning operation prune$(LN(W, i), \theta)$ is applied to the $LN(W, i)$. This operation deletes states and arcs in the input FST that do not belong to a successful path whose weight is no more (w.r.t the natural order) than the threshold θ times the weight of the best path (which due to natural order definition is selected based on the combined acoustic and initial language model scores) in the input FSA [75].

4.3.3 The Extended Algorithm

Applying the extensions described in Sections 4.3.1 and 4.3.2, we introduce the new hill climbing algorithm, which is shown in Algorithm 4. Let us distinguish the new version from the initial version of the algorithm (shown in Algorithm 3) through the following bullet points:

- The neighborhood definition is expanded to edit distance 2.
- We apply a beam pruning scheme to the neighborhoods using the $\langle T, T\rangle$-lexicographic weighted FSA representation of the lattices and the intersected neighborhoods.

Algorithm 4 The extended Hill Climbing Algorithm

$\mathcal{L}_{\langle g_\Gamma, \Lambda_{\text{old}}, g_\Gamma \rangle} \leftarrow$ **WFSA** of the initial (first-pass) lattice using LexicographicWeight.
$\theta \leftarrow$pruning threshold
$W \leftarrow$Viterbi path of the initial lattice [**Initilization**]
$N \leftarrow$ **length**(W)
repeat
  $i \leftarrow 1$ [Start at position 1]
  while $i \leq N + 1$ **do**
    [**Neighborhood generation with pruning**]
    create **FSA** $LC(W, i)$
    $LN(W, i) \leftarrow LC(W, i) \circ \mathcal{L}_{\langle g_\Gamma, \Lambda_{\text{old}}, g_\Gamma \rangle}$
    $LN(W, i) \leftarrow \text{prune}(LN(W, i), \theta)$
    [**Neighborhood rescoring according to Eqn. 4.1**]
    $W \leftarrow \arg\max_{W' \in LN(W,i)} g(A, W'; \Gamma, \Lambda_{\text{new}})$
    [Adjust length and position]
    $N \leftarrow$**length**(W)
    if DELETION **then**
      $i \leftarrow i$-1
    end if
    $i \leftarrow i$+1
  end while
until W does not change [Stopping criterion]

4.4 Mitigating Effects of Local Maxima

Our algorithm, as true in general of hill climbing algorithms, may yield a local maximum solution as there is no guarantee that it finds the global maximum when applied to a non-convex space. Two common solutions to overcome this problem are,

1. **Random-restart hill climbing** where hill climbing is carried out using different random starting points (word sequences)

2. **Simulated Annealing** in which one chooses a random move from the neighborhood (recall that hill climbing chooses the best move from all those available, at least when using the steepest ascent variant). If the move results in a better word sequence (in terms of the score under the new model) than the current word sequence then simulated annealing will accept it. If the move is worse then it will be accepted according to some probability [76].

This dissertation considers the random-restart technique. Algorithm 4 is repeated M times, each time with a different initial word sequence. At the end of each iteration, the score (under the new model) of the resulting path is stored. Hence, we will have M different stopping paths , $(v_1, v_2, \cdots, v_M)$, along with their corresponding scores, $(g_1, g_2, \cdots, g_M)$. The path with the maximum score is selected as the final output of the algorithm. The M starting paths (word sequences) are selected by sampling the initial lattices (based on the distribution imposed by the initial model). In addition, we make sure that the sampled paths are not repeated and also for the first iteration

we start from the viterbi path in the lattices. In Section 4.4.1, we discuss how sampling can be efficiently applied to FSA representation of lattices.

Remark. It is worth mentioning that the random-restart hill climbing can easily be distributed across multiple CPU threads. First, a set of M initial paths are sampled from the lattice and then, each thread can start the hill climbing algorithm using one of these initial paths. Once, all threads converge to their final solution, the final output can be decided by using the final score from each thread.

4.4.1 Sampling from a Weighted FSA

A weighted FSA (WFSA) encodes a distribution over a set of strings (hypotheses) and hence, sampling from a WFSA is equivalent to sampling from the corresponding distribution. Here, we describe a sampling algorithm. Let us first establish a notation for a WFSA L. Let us represent an edge in $\mathcal{L}$ with e. We also denote a state in $\mathcal{L}$ with v. The starting and ending states of an edge e are represented by $l(e)$ and $r(e)$ respectively. Foreach state v, we use $E(v)$ to denote the set of outgoing edges from v. Using the developed notation for WFSA $\mathcal{L}$, the sampling procedure can be applied using the following steps:

1 Run the backward algorithm to get the backward scores for each state v of L.

2. Start from the initial state of $\mathcal{L}$ and assign it as the current state v.

3. Among the outgoing edges $e \in E(v)$ of the current state v, sample one according

to the following score:

$$\frac{s(e)\beta(r(e))}{\beta(v)} \tag{4.9}$$

where $s(e)$ is score of the outgoing edge e. $\beta(r(e))$ is the backward score corresponding to the ending state of e and $\beta(v)$ is the backward score of the current state v. It is easy to see that the above score is between 0 and 1 as $\sum_{e \in E(v)} s(e)\beta(r(e)) = \beta(v)$.

4. Assign the ending state of the selected edge in step (3) as the current state v. If v is not the final state go to step (3). Otherwise, terminate.

The selected states and edges from the steps above implies a sampled path from the WFSA. Figure 4.6 depicts this procedure for a WFSA where the red edges correspond to the sampled path. The complexity of running the backward algorithm for an *acyclic* WFSA is $O(|E| + |V|)$, where $|E|$ and $|V|$ are the number of edges (arcs) and states of the WFSA respectively [74]. Therefore, running the sampling procedure to get M path requires $O(M.l + |E| + |V|)$[6] where l is the expected length of a successful path according to the distribution embodied by the WFSA.

[6]The backward algorithm needs to be run just once.

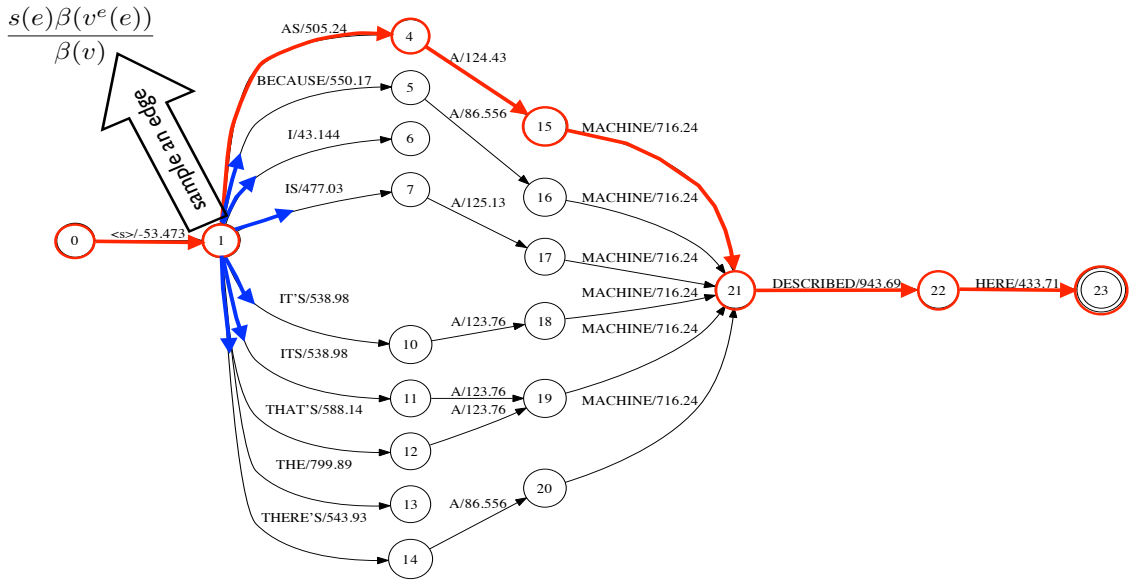


Figure 4.6: Sampling procedure applied to a WFSA assuming the backward scores are calculated for each state.

4.5 Experimental Setup

4.5.1 Corpora, Baseline and Rescoring Models

The ASR system used throughout this chapter is based on the 2007 IBM Speech transcription system for GALE Distillation Go/No-go Evaluation [38]. The acoustic models used in this system are state-of-the-art discriminatively trained models and are the same ones used for all experiments presented in this chapter.

As a demonstration of our proposed rescoring framework, we first build a (pruned) 3-gram language model with modified Kneser-Ney smoothing [17] on $400M$ word *broadcast news* LM training corpus which includes the following sources: 1996 CSR Hub4 Language Model data, EARS BN03 closed captions, GALE Phase 2 Distillation

GNG Evaluation Supplemental Multilingual data, Hub4 acoustic model training transcripts, TDT4 closed captions, TDT4 newswire, and GALE Broadcast Conversations and GALE Broadcast News. This 3-gram LM has about $2.4M$ n-grams and is used to generate initial lattices for all the experiments. For the rescoring experiments, we train two different models (to make sure our algorithm is applicable across different models) on the above LM text:

1. a (partially pruned) 4-gram LM with about $64M$ n-grams.
2. Model M shrinking based exponential LM [77]. This LM has been reported to have state-of-the-art ASR performance for the broadcast news task [78].

The evaluation set is the 4h rt04 evaluation set containing 45k words. The WER of the initial 3-gram LM on this data set is 15.5%. Additionally, we report rescoring results on dev04f using Model M to show the generalizability of the hill climbing algorithm across different data sets. The initial WER on dev04f using the lattice generating 3-gram LM is 17.0%.

4.5.2 Evaluation of the Efficacy of Hill Climbing

We evaluate the following aspect of our proposed hill climbing algorithm:

- Comparison of the proposed algorithm and N-best rescoring. For this, we calculate the average number of word sequence evaluations under the new models needed for both methods to get to a particular WER. In our algorithm during the

rescoring phase (of the neighborhood set), we need to query the LM score of each word sequence in the neighborhood set. In order to maximally expedite scoring, we maintain a **look-up table** of previously computed scores for word sequences, computing scores using the evaluation method for the rescoring model only for new word sequences which are not included in the look-up table. For each utterance, the effective number of performed evaluations by the rescoring model can be obtained from the size of the look-up table at the end of the hill climbing procedure. Also, for N-best rescoring we generate a list of word sequences in which all the paths are **unique** and hence the size of the generated list is essentially the number of effective evaluations.

Evaluating each hypothesis using a complex long-span model is computationally expensive. Therefore, a reduction in the number of evaluations effectively corresponds to a speedup in runtime of the resoring framework.

- An important factor, which directly affects the efficiency, is the amount of **overhead-time** spent by the proposed algorithm to utilize the search. The overhead-time for our random-restart hill climbing algorithm is due to the time spent to:

  1. Sample initial paths
  2. Generate neighborhoods (constructing $LC(W, i)$ and intersecting it with the lattice)
  3. Prune neighborhoods

4. Enumarate paths in the neighborhoods (to be resocred by the new model)

The overhead-time for N-best rescoring, on the other hand, is the time spent generating the N-best list. We compare the added overhead-time using each algorithm to reach a particular WER.

- The algorithms are evaluated based on the amount of **search errors** that they introduce. In other words, they are analyzed based on how close they can get to the WER of the optimal solution (global maximum) under the rescoring model on the lattices. To this end, the rescoring LMs are selected, for now, to be such that exact lattice rescoring is possible; Hence, the optimal WERs can be reported.

Therefore, a rescoring framework (as a search strategy) is considered efficacious if it reaches to solutions close enough to the optimal solution with a small number of hypothesis evaluations and a reasonable added overhead-time.

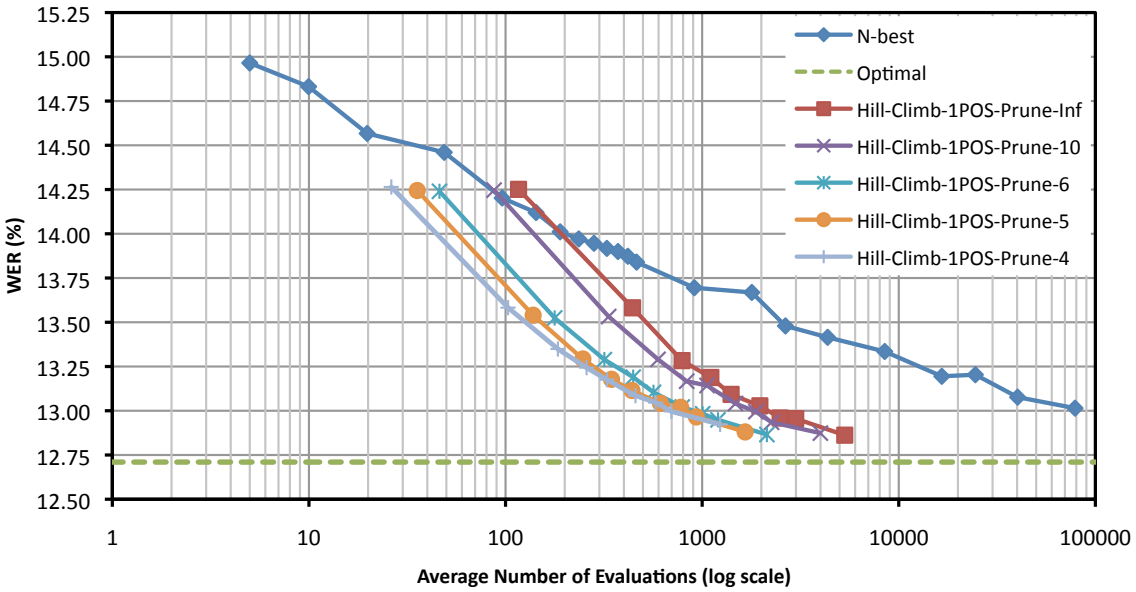
4.6 Results and Discussion

The results for comparing our hill climbing method to N-best rescoring on rt04 using Model M may be seen in Figure 4.7. The x-axis corresponds to the average number of unique hypothesis evaluations (on a log scale), calculated as described in Section 4.5.2 for each method. The y-axis shows the WER of the output of the algorithms for each experiment. For hill climbing, we experiment with different pruning thresholds, where $\theta = \infty$ corresponds to experiments with no prun-

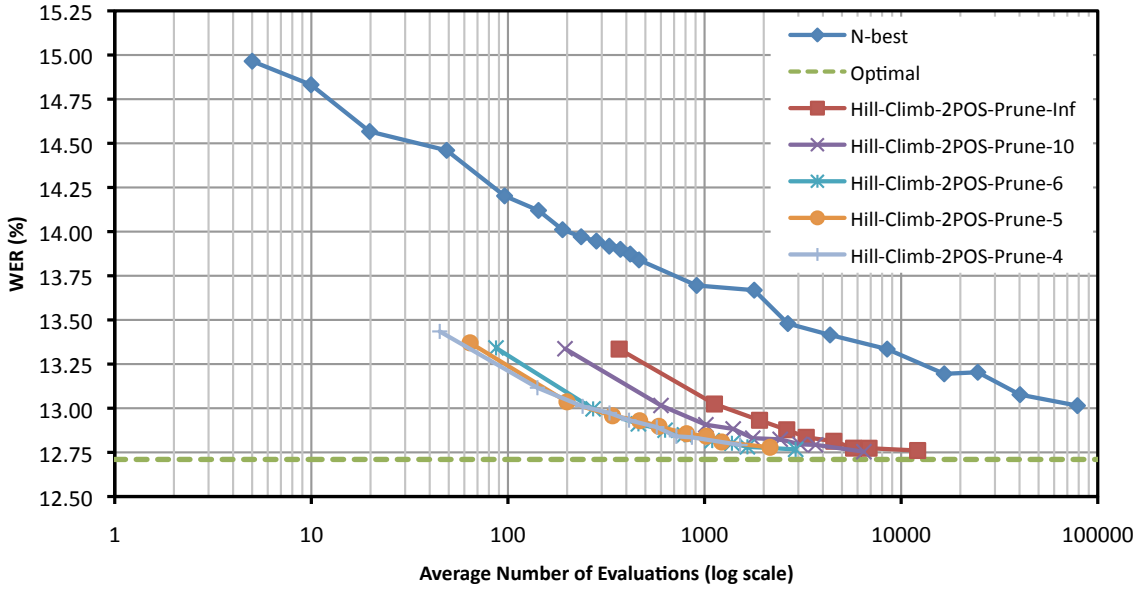
ing applied. In addition, for hill climbing experiments, we use random-restart with $M = \{1, 5, 10, 15, 20, 30, 40, 50, 100\}$ starting paths (which corresponds to the different points of the corresponding hill climbing line). We evaluate N-best rescoring with N ranging from 5 to 50000. The horizontal dashed line in each figure shows the WER of the best-scoring solution in the lattices using the rescoring model, Model M, which is about 12.7%. These optimal WERs can be calculated due to the fact that the rescoring models, which we used for our experiments, can be represented as **WFSA** and hence be composed with the lattices to extract the best path. However, we emphasize that this *can not* be done for models with longer dependencies and syntactic information, and therefore, the need for a rescoring framework is inevitable with those models[7].

The effect of using different neighborhood structures, namely 1 vs. 2 edit distance, on the efficiency of the hill climbing algorithm is shown in Figures 4.7(a) and 4.7(b), respectively (the N-best experiments are repeated in both figures for the propose of comparison). These figures show that our proposed hill climbing algorithm results in far fewer evaluations to reach competitive WERs, including the optimal WER, when reasonable pruning is applied. Moreover, comparing hill climbing experiments in Figure 4.7(a) with those of Figure 4.7(b), shows that the use of 2 edit distance neighborhood structure when combined with pruning substantially improves the efficiency of the algorithm to quickly reach to solutions close to the optimal solution.

[7]One of such models is our SLM, for which the rescoring WER results using the proposed hill climbing algorithm is presented later in Section 4.7.



(a)



(b)

Figure 4.7: Hill-Climbing vs. N-best rescoring on rt04 using Model M with different pruning thresholds $\theta = \{\infty, 10, 6, 5, 4\}$ and (a) 1 edit distance neighborhood (b) 2 edit distance neighborhood.

As an example, using 1 initial path, $\theta = 5$ and 1 edit distance neighborhoods results in 14.3% WER after evaluating about 36 hypotheses (per utterance) in average. 2 edit distance neighborhood structure with $\theta = 4$ and 1 initial path however, results in 13.4% WER after evaluating about 45 average hypotheses. That is about 0.9% absolute reduction in WER, using almost the same number of average evaluations. When compared to N-best framework, our extended algorithm produces similar WER results with speedups up to *two orders of magnitude* (or even more). This improvement is due to the characteristics of the hill climbing method where at each step the moves are selected (from neighborhood set) based on their quality under the new model (in contrast to N-best rescoring where the evaluating points are selected based on the initial model).

Figure 4.8 illustrates the rescoring experiments on rt04 using the 4-gram LM where the hill climbing is applied using 2 edit distance neighborhood structure. In this figure, the optimal solution of 4-gram LM has about 13.6% WER. An important observation, seen by comparing Figure 4.8 with Figure 4.7(b), is that the problem with N-best rescoring (non-efficiency in terms of effective evaluations) is more severe when the rescoring model is more different/orthogonal to the initial model. In our case, Model M LM is capturing knowledge which significantly differs from the initial 3-gram model (due to exponential-family parameterization used in Model M) , compared to the 4-gram vs. 3-gram LM. Hence, one can see in Figure 4.7(b) a large gap between the two rescoring methods in terms of the number of needed evaluations.

Finally, the results for comparing WER vs. overhead-time trade-off for both algorithms on rt04 using Model M is shown in Figure 4.9 where the hill climbing is applied with 2 edit distance and different number of initial paths. The overhead-time is calculated as described in Section 4.5.2. It can be observed from this figure that:

1. The proposed hill climbing algorithm has a better WER vs. overhead-time trade-off. This is due to the fact that our proposed hill-climbing algorithm can quickly reach good solutions in the space and, hence, the overhead-time can be compensated effectively. As an example, 200 seconds of overhead-time in N-best rescoring reaches 13.7% WER, whereas the same overhead-time in hill climbing results in 13.4%. This is on top of the fact that the 13.7% WER has been reached by evaluating about 2000 hypotheses on average compared to average 34 evaluations in the corresponding hill climbing algorithm.
2. Comparing the overhead-time of hill climbing with and without pruning reveals that the pruning scheme is very efficient and does not contribute significant added overhead to the algorithm.

4.7 SLM Speech Recognition Experiments

We conclude with experiments using our improved SLM for speech recognition. Since the SLM uses equivalence classification of histories based on dependency struc-

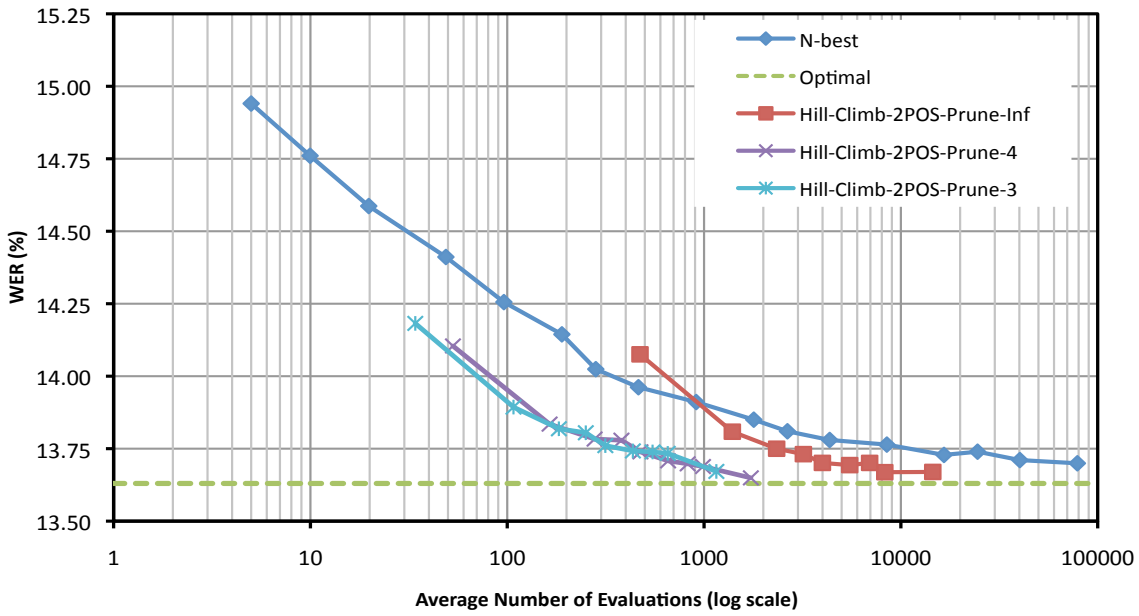


Figure 4.8: Hill-Climbing vs. N-best rescoring on rt04 using a huge 4-gram LM with different pruning thresholds $\theta = \{\infty, 4, 3\}$ and 2 edit distance neighborhoods.

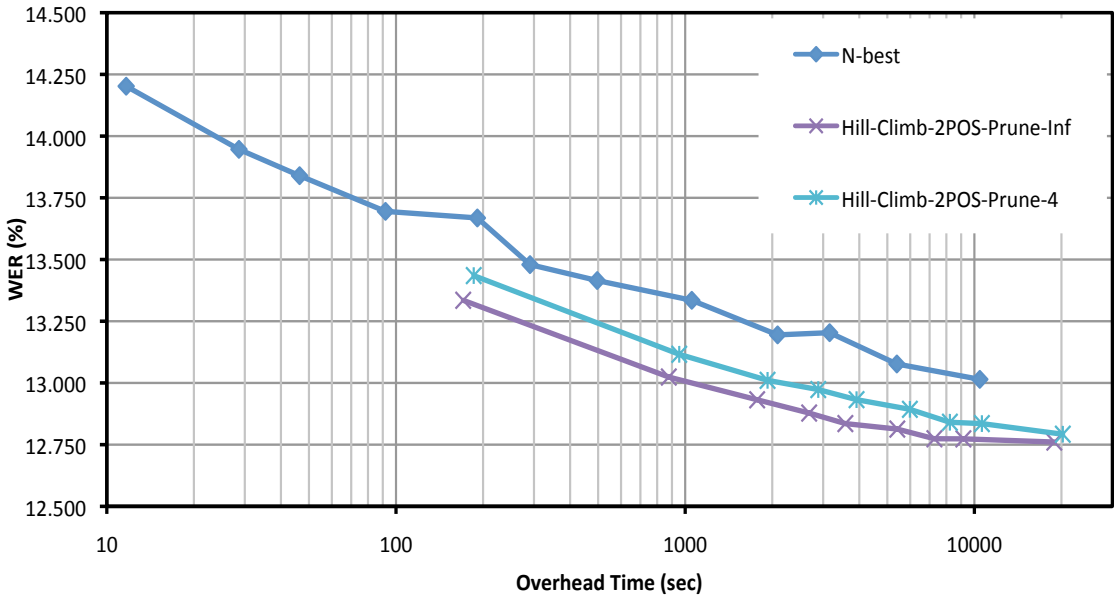


Figure 4.9: Overhead of Hill-Climbing algorithm vs. N-best rescoring on rt04 using Model M with different pruning thresholds $\theta = \{\infty, 4\}$ and 2 edit distance neighborhoods.

tures, it is a long-span model, which excludes the ability for direct lattice rescoring. Therefore, we use the SLM as the long-span model in our proposed hill climbing algorithm, which is demonstrably better than N-best rescoring (as shown in the experiments of Section 4.6). Since the hill climbing algorithm is a greedy search with no guarantees of global optimality, we use 1, 10, 20 and 50 random-restarts, repeating hill climbing with different initial word sequences sampled from the speech lattice, and output the best scoring final output. The ASR system used in these experiments is the one described in Section 4.5. The acoustic model is also the same as the one described in Section 4.5.

4.7.1 Treebank Tokenization

The original lattices in our ASR system use a different tokenization than the one needed by our SLM, which is essentially the Treebank-style tokenization[8]. The treebank-style tokenization is required for POS-tagging, dependency parsing and hence, SLM scoring. Therefore, we convert each hypothesis during rescoring to the Treebank-style tokenization where words are down cased and contractions and possessives are tokenized as shown in Table 4.1. The final output is converted back to standard tokenization (by recombining tokens that were split, namely, possessives and contractions) for WER computation as the reference transcriptions are in ASR tokenization.

[8]We also refer to the ASR tokenization as untokenized and use these two terms interchangeably throughout this work.

ASR	Treebank
DON'T	do n't
I'M	i 'm
I'LL	i 'll
I'D	i 'd
WE'VE	we 've
YOU'RE	you 're
GORE'S	gore 's
CANDIDATES'	candidates '

Table 4.1: Treebank vs. ASR tokenization

4.7.2 Sentence Boundaries

One of the challenges for incorporating a SLM into an ASR system is the lack of clear sentence boundaries. In fact, it has been shown in [79] that the quality of the automatically detected sentence boundaries can substantially affect the quality of the parser in a conversational speech recognition task. Throughout, this dissertation we use the"end of sentence" marker `</s>` hypothesized in the ASR lattices to decide the sentence boundaries. In other words, we chop a hypothesized utterance during rescoring at places where `</s>` marker is present and treat each segment as a separate sentence to apply the SLM. Let us clarify this convention via the following broadcast news example which is a hypothesis from the ASR first pass decoding (after Treebank tokenization):

```
<s> a distributor sent to the people of taiwan </s> so i think
this is a big mistake </s>
```

The example above is segmented into two sentences, namely:

(i) `<s> a distributor sent to the people of taiwan </s>`

(ii) `<s> so i think this is a big mistake </s>`

Then, to get the SLM score for the hypothesis, each sentence is POS-tagged, dependency parsed and scored by the SLM, i.e. the SLM score for a hypothesis is the sum of the SLM scores of all the comprising sentences.

Remark. We note that the sentence segmentation described above is by no means the best way to detect sentence boundaries[9] and can most definitely be improved by applying state-of-the-art statistical methods for detecting the boundaries [80]. However, employing such methods to improve the detected boundaries and measuring its effects on the performance of the SLM is simply beyond the scope of this dissertation and is left to future experimentations.

4.7.3 Results

In this section, we present the ASR experiments using the `BN` setup of Section 3.3.2. We use the SLM with $\texttt{HW+HT}_2$ interpolated with the baseline 4-gram LM (which resulted in the best perplexity performance). Lattices are generated using a pruned Kneser-Ney 4-gram LM (trained on untokenized training text.) The lattice generating LM has 15.4% WER and our baseline 4-gram LM (unpruned) has 15.1% on the

[9]The method is only based on "end of sentence" information captured by the lattice generating n-gram.

LM		Pruned	Unpruned	# Evaluations (averaged)
Kneser-Ney 4-gram		15.4%	15.1%	–
Interp. SLM	1 initial path	14.7%*	14.7%*	42
	10 initial paths	14.6%**	14.6%**	232
	20 initial paths	**14.5**%**	**14.5**%**	390
	50 initial paths	14.5%	14.5%	787

* statistically significant with $p = 0.02$ using matched pair sentence segment WER test.

** statistically significant with $p < 0.001$ using matched pair sentence segment WER test.

Table 4.2: Word error rates for BN speech recognition experiments using the proposed SLM with HW+HT$_2$ in the hill-climbing algorithm. The hill-climbing is used with 2 edit distance neighborhood structure and neighborhood pruning threshold of $\theta_{\text{hill-climb}} = 4$. The significant tests has been carried out w.r.t the unpruned 4-gram LM.

rt04 evaluation data. Also, to evaluate the effect of the proposed SLM pruning scheme which was described in Section 3.5—here, by pruning, we mean pruning the SLM not the neighborhood— we consider both the unpruned and pruned ($\theta_{\text{SLM}} = 5 \times 10^{-7}$) SLM[10]. The hill climbing with 2 edit distance neighborhood structure and neighborhood pruning threshold of $\theta_{\text{hill-climb}} = 4$ is used. Moreover, to make sure we fully realize the power of the SLM, the hill-climbing is applied with 1, 10, 20 and 50 random-restarts sampled from the original lattices (Section 4.4).

The recognition results in Table 4.2 show that the interpolated SLM significantly improves the baseline performance: a 0.6% absolute improvement in WER when 20 random-restart are used. We experimented with more initial paths (more than 50) and the WER did not improve further. Hence, it is safe to say that the WER of the optimal solution using the SLM is around 14.5%. In addition, it can be observed that our proposed SLM pruning method does not degrade performance, despite significantly

[10]The perplexity results were presented in Section 3.5.2

N-best	WER (%)
40	14.9
230	14.8
390	14.8

Table 4.3: Word error rates for speech recognition using the proposed $\texttt{HW+HT}_2$ SLM with N-best rescoring. The N is chosen to match the average number of evaluations in the hill climbing algorithm with $\{1, 10, 20\}$ initial paths.

reducing LM size. Finally, we observe that hill-climbing with 1 and 20 initial paths results in an average of only 42 and 390 hypotheses evaluated per utterance. To show that our proposed hill climbing results in a big computational saving and is able to fully realize the power of the SLM, the WER results using N-best rescoring with the SLM are reported in Table 4.3 where we assign N to be the average number of evaluations of the hill climbing algorithm for different initial paths (Table 4.2). It can be seen from the table that our proposed hill climbing algorithm even with 1 initial pants and average 40 evaluations has a better performance than 390-best rescoring, which is a big computational saving. In addition, to compare the effectiveness of the hill climbing algorithm as a search algorithm with the N-best rescoring, we plot the (averaged) model score (acoustic score + SLM score) of each algorithm in Figure 4.10. In this figure the y-axis corresponds to the averaged model score of the rescoring outputs according to Eqn. 4.1 where $\log P(W|\Lambda_{\text{new}})$ is the interpolated SLM score[11]. We compare the output score for different averaged number of evaluations. This figure clearly reveals the superior power of the hill climbing algorithm as a search algorithm

[11] Averaged model score is $\frac{1}{M}\sum_i g(A_i, W_i; \Gamma, \Lambda_{\text{new}})$ where M is the number of speech utterances and $g(A_i, W_i; \Gamma, \Lambda_{\text{new}})$ is the algorithm output score for each utterance i.

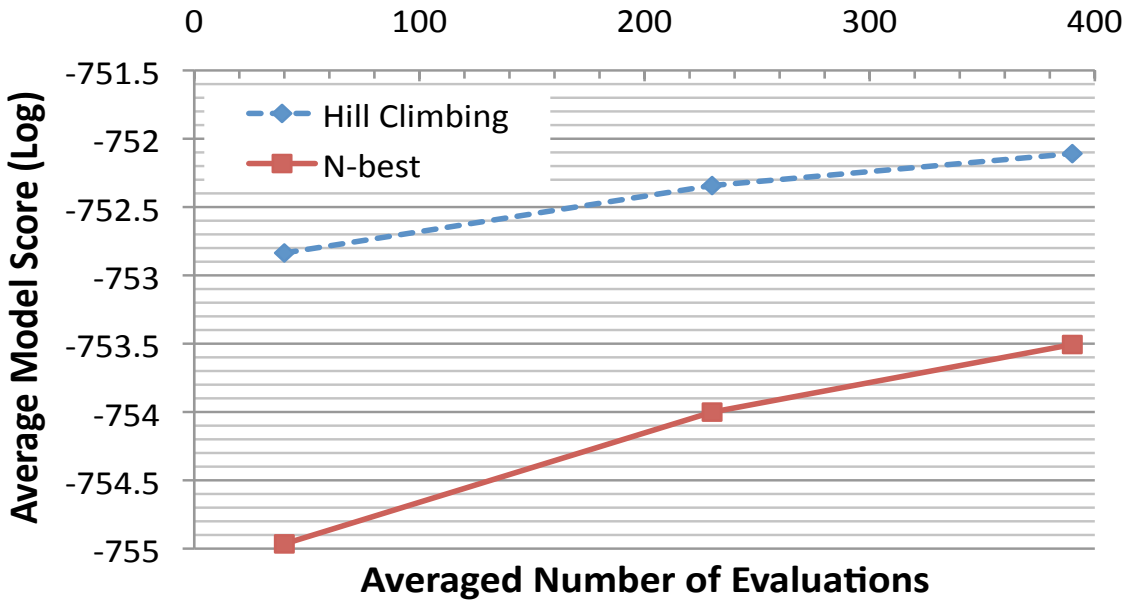


Figure 4.10: The averaged (averaged over the evaluation utterances) model score (Acoustic+SLM) of the rescoring output using N-best rescoring vs. Hill climbing algorithm.

compared to N-best rescoring. Hill climbing using only 40 evaluations per utterance (averaged) reaches the output solutions (in the lattices) with an average model score of -752.11, whereas N-best rescoring with even 400 averaged number of evaluations reaches an average model score of -753.06. In other words, solutions found by hill climbing algorithm using only 40 evaluations per utterance are $\frac{\exp(-752.1)}{\exp(-753.06)} \approx 2.6$ times more likely under the interpolated SLM than those found by 400-best rescoring.

4.8 Conclusions

In this chapter, we introduced a novel rescoring framework on speech lattices based on hill climbing via edit-distance based neighborhoods. We showed that the proposed method results in far fewer full-sentence evaluations by the complex model than conventional N-best rescoring and it results in a very efficient search algorithm. The efficiency and effectiveness of the proposed algorithm is extensively evaluated. Additionally, our experiments indicate that the SLM, when combined with the efficient hill-climbing algorithm, significantly improves recognition accuracy over regular n-gram models.

Although we have used the proposed algorithm to carry out only LM rescoring, we emphasize that the method is applicable for rescoring using a new acoustic model, with multiple knowledge sources and in a discriminative model combination scenario.

Chapter 5

Efficient Discriminative Training of Long-span Language Models

5.1 Introduction

Standard language modeling methods for automatic speech recognition (ASR) are based on building *generative* models. Regardless of the type of information (feature) used in the model, be it simple n-gram statistics or long-span syntactical features, the parameters of the model are trained using maximum likelihood (ML) estimation on text that represents the domain of interest. Therefore, the parameters of the LM are not directly optimized for the ultimate objective function, i.e. WER. In fact, the syntactic LM we proposed in Chapter 3 is a generative model where the parameters of the underlying Jelinek-Mercer LM with headword features are trained on 40M words

of text data.

Another trend, which has recently been taking hold in ASR, is towards discriminatively trained models [35] [37] [81] [36]. The idea of a discriminatively trained model is to use linguistic context to resolve acoustically confusable words in the decoded hypotheses, with the ultimate goal of reducing ASR errors on some training data. Therefore, It is natural to expect that a LM can benefit from discriminative training, given its role in selection of the correct hypothesis from the large ASR hypothesis space. In addition, discriminative models provide the flexibility to include both typical n-gram features and arbitrary features based on the global properties of word sequences (long-span features) through the use of globally normalized log-linear models. Examples of discriminative language models with long-span features such as morphological and syntactic features can be found in [2, 82, 83].

However, unlike generative LMs with long-span dependencies, such as our proposed SLM in Chapter 3, where one has to resort to N-best lists only during decoding, globally normalized discriminative models with sentence-level features force the use of N-best lists even for LM *training*. Collins et al. [2], for instance, trains a syntactic LM on the 1000-best outputs from a speech lattice. This adds a significant computational cost to training: parsing 1000s of candidate paths can practically prohibits the use of *online learning algorithms* to adjust the parameters of the discriminative model on newly received training samples.

There are now alternatives to N-best rescoring with long-span LMs [84] [66]. In

Chapter 4 we presented a *hill climbing* procedure on ASR lattices that iteratively searches for a higher-scoring hypothesis in a local neighborhood of the current-best hypothesis; the error-reduction from hill climbing matches or surpasses N-best rescoring, with up to two orders of magnitude reduction in the number of evaluated utterances. We seek a similar reduction in the computational complexity in discriminative training of globally normalized language models.

The key idea in this chapter may be described as *discriminative hill climbing*. A discriminative LM learns how to prefer a correct sequence hypothesis over all incorrect alternatives, while the main idea of hill climbing is to move incrementally towards the correct sequence-hypothesis via local changes to a current hypothesis. The new LM training procedure we present combines these two ideas: adjusting the long-span LM parameters to prefer a "locally better" hypothesis, so that when the fully trained LM is applied in conjunction with hill climbing, a tendency to move towards the correct hypothesis will result. In effect, we learn how to discriminate between good and bad hypotheses during hill climbing.

This chapter is organized as follows. We begin by introducing the discriminative language model of [2] in Section 5.2, which provides the global-linear parameterization that we use in our framework. In Section 5.3, we apply our proposed hill climbing rescoring algorithm (from Chapter 4) and apply it to rescoring speech lattices using global-linear models with long-span features. In Section 5.4, we then propose our *discriminative hill climbing* algorithm to efficiently train such global-linear models on

the speech lattices. Section 5.5 discusses some related work which is similar to our proposed algorithm. Sections 5.6 and 5.7 present our experimental setup and results followed by conclusions in Section 5.8.

5.2 Discriminatively Training with Long-Span Features

Long-span language models extend beyond the local n-gram context to incorporate long range dependencies such as features based on full-sentence parser tree [2], morphological features [83], and trigger features [83] from the word sequence. One general framework to incorporate such long-span features for the task of language modeling is through the use of *global linear* discriminatively trained models [2, 37]. We use the framework of Collins et al. [2] and begin with a review of this model.

First, let us review the statistical formulation of an ASR system using the notation developed in Section 4.2. In speech recognition, as formulated in Section 2.1, an acoustic model $P(A|W, \Gamma)$ induces a conditional distribution over acoustic sequences A given a sequence of words W. A language model (typically a n-gram model) $P(W|\Gamma)$ induces a prior distribution over word sequences. Given these models, recognition is formulated as a search task for finding the most likely word sequence $\widehat{W}$ for the input

acoustics:

$$\widehat{W} = \arg\max_{W} \gamma \log P(A|W,\Gamma) + \log P(W|\Lambda) \tag{5.1}$$

where $\gamma > 0$ is the inverse of the language model factor. $\log P(W|\Lambda)$ and $\log P(A|W,\Gamma)$ are the language model and acoustic model scores, respectively.

Collins et al. extend this model to include generic features; a d-dimensional feature vector $\Phi(A,W) \in \mathbb{R}^d$ that encodes arbitrary features of A and W. A weighted linear combination of features and model parameters $\vec{\alpha} \in \mathbb{R}^d$ is added to the model (Eqn. 5.1):

$$\gamma \log P(A|W,\Gamma) + \log P(W|\Lambda) + \langle \vec{\alpha}, \Phi(A,W) \rangle \tag{5.2}$$

where $\langle \vec{\alpha}, \Phi(A,W) \rangle$ is the dot product between the feature vector and the corresponding weights. While $\Phi(A,W)$ can in general include features from the input acoustic sequence, we use only word sequence dependent features alone throughout the chapter. To achieve a compact notation, the first feature of $\Phi(A,W)$ is defined to be the baseline (first-pass) recognizer score,[1]

$$\Phi_0(A,W) = \gamma \log P(A|W,\Gamma) + \log P(W|\Lambda), \tag{5.3}$$

so that the optimal solution under the above global linear model using the baseline feature is:

$$\widehat{W} = \arg\max_{W} \langle \vec{\alpha}, \Phi(A,W) \rangle \tag{5.4}$$

[1]We denote this feature as *baseline feature* throughout the chapter.

The *learning* task is to estimate the parameters of the global-linear model above using a set of training speech examples. During the *decoding* algorithm, on the other hand, we search for word sequences that maximize Eqn. 5.4, for the given A.

5.2.1 Features

The features $\Phi(A, W)$ used in a global linear discriminative language model are often based on the *count* of extracted statistics from the word sequences in the training data. As an example, the language modeling features might take into account n-gram statistics such as

$$\Phi_j(A, W) = \text{count of } \texttt{"the united states"} \text{ in } W$$

Collins et al. [2] uses syntactic features which are based on the count of extracted statistics from the constituent parse tree for $\mathbf{w}$, e.g.

$$\Phi_j(A, W) = \texttt{VP} \rightarrow \texttt{VB NP} \text{ in } T(W)$$

where $T(W)$ is the parse tree for W and count of $\texttt{VP} \rightarrow \texttt{VB NP}$ is an example of a context-free rule production in $T(W)$. The advantage of the global linear parameterization described in the previous section is that it allows a framework for incorporating arbitrary features for language modeling.

5.2.2 Parameter Estimation

The *global-linear* model parameters $\vec{\alpha}$ can be estimated using a discriminative objective such as global conditional log likelihood [37] or via the Perceptron algorithm [85]. Throughout this work, we rely on the Perceptron algorithm, a popular online learning algorithm with wide applications to structured learning problems in NLP [85], and in particular, discriminative language modeling for speech recognition [37]. As an online algorithm, the Perceptron processes training instances one at a time, updating $\vec{\alpha}$ after each example.

Figure 5.1 illustrates the perceptron algorithm for estimating the $\vec{\alpha}$ parameters of the global model. The input to the algorithm is the pair of acoustic inputs (A) and the corresponding reference transcriptions (W_i) over the set of training examples. For each training example in each algorithm iteration t, the best scoring hypothesis $\widehat{W_i}$ according to current model parameters $\vec{\alpha}$ is selected from the set of possible word sequences in $\mathbf{GEN}(A_i)$, which is a set of possible word sequence candidates under the first-pass (baseline) recognizer for A_i (Figure 5.1). $\widehat{W_i}$ is compared to W_i^*, the word-sequence in $\mathbf{GEN}(A_i)$ that is closest to the reference W_i in terms of WER (edit distance). If the current $\vec{\alpha}$ does not yield W_i^*, the weights in the model are updated, and the algorithm moves to the next training example (A_{i+1}, W_{i+1}). The applied update is as follows: the weight of each feature is increased by the count of that feature in W_i^* and decreased by the count of that feature in $\widehat{W_i}$. We note that the weight corresponding to the baseline feature α_0 is not updated and instead, will be

Input:
Training examples $\{(A_i, W_i)\}|_{i=1}^{M}$.
A feature-vector representation $\Phi(A, W) \in \mathbb{R}^d$.
$\mathbf{GEN}(A)$: the candidate list for example A.
Algorithm:
for $t : 1..T$ [Iterations]
  for $i : 1..M$ [Examples]
    $\widehat{W}_i = \arg\max_{W \in \mathbf{GEN}(A_i)} \langle \vec{\alpha}, \Phi(A_i, W) \rangle$
    $W_i^* = \arg\min_{W \in \mathbf{GEN}(A_i)} d(W, W_i)$
    if $\widehat{W}_i \neq W_i^*$ [If Mistake]
      for $j : 1..d$ [Update]
        $\alpha_j = \alpha_j + \Phi_j(A_i, \widehat{W}_i) - \Phi_j(A_i, W_i^*)$
Output: $\vec{\alpha}$

Figure 5.1: The Perceptron training algorithm for discriminative language modeling [2]. α_1 is the weight given to the first-pass (baseline) recognizer score and is tuned on development data.

tuned using a development set.

5.2.3 N-Best Lists as $\mathbf{GEN}(A)$

The creation of the $\mathbf{GEN}(A)$, candidate word sequences, is required for discriminative training and for decoding, when the trained model is used to rescore the baseline recognizer's output. A global linear model which only includes n-gram features in $\Phi(A, W)$ can be efficiently represented using a deterministic weighted finite-state automata (WFSA)[2]. Therefore, for such models one can directly use the first pass recognized *word-lattices* as the set of possible outputs $\mathbf{GEN}(\mathbf{a})$. Assuming the WFSA representation of first pass lattice $\mathcal{L}$ corresponding to speech signal A, the best word sequence according to the perceptron model $\widehat{W}$ can be efficiently found by

[2]We refer the readers to [37] for the details of such representation.

the following WFSA operation:

$$\widehat{W} = \arg\max_{W} \langle \vec{\alpha}, \Phi(A, W) \rangle = \text{BestPath}(\alpha_0 \mathcal{L} \circ \mathcal{D}), \tag{5.5}$$

where $\alpha_0 \mathcal{L}$ denotes scaling the lattice $\mathcal{L}$ with the baseline score α_0 and $\mathcal{D}$ denotes the WFSA representation of the n-gram discriminative model. Thus finding the best solution under the new model (which is used during both decoding and training of discriminative model) involves first, scaling $\mathcal{L}$ with α_0, intersecting it with the discriminative language model $\mathcal{D}$, and finally, finding the best scoring path in the resulting WFSA.

However, including general long-span features, such as features based on syntactic parse trees, prevents a global linear model from being represented efficiently (like $\mathcal{D}$) as a WFSA. Therefore using the word lattices $\mathcal{L}$ as $\mathbf{GEN}(A)$ is not feasible, for it is computationally intractable to enumerate all paths in the lattice and score them under the new discriminative model. The conventional solution is to instead approximate the search space $\mathbf{GEN}(A)$ for model training (Algorithm 5.1) as well as decoding with an N-best list, the N highest scoring paths from the baseline recognizer.

However, we already know from the discussions in Chapter 4 that N-best list approximation of the search space is **inefficient** and can pose significant computational costs as each enumerated path in the list must be processed and scored under the discriminative model which, in case of syntactic features, involves POS tagging and

parsing each candidate in the list. This imposes a tradeoff in the selection of N; smaller N will suffer from search errors while larger N are computationally expensive. A similar difficulty arises during model training, where the perceptron algorithm must score all the paths in the N-best list according to the current status of the discriminative model with long-span features to obtain $\widehat{W}$. Although, it can be argued that during training one only needs to **pre-process** the N-best list of each training example and extract the needed long-speech features for each hypothesis in the list once, the computation burden of the pre-processing step can make training on large scale data prohibitive. In an *online learning* scenario, it also prohibits the ability of efficiently adjusting the parameters of the trained model on newly received training samples.

Our goal in this chapter is to address the limitation of using N-best lists during training global-linear models with long-span features, and to provide an efficient framework to train such models directly using the lattices $\mathcal{L}$ in the role of $\mathbf{GEN}(A)$.

5.3 Hill Climbing Rescoring using Global-linear Models

Given a speech utterance A and a lattice $\mathcal{L}$ of candidate transcripts, lattice rescoring under a *global-linear* model (with $\vec{\alpha}$) tries to solve the search problem of Eqn. 5.4. As discussed in Section 5.2.3, including long-span features in the global-linear model

prevents the direct lattice rescoring. Using the notation developed in Chapter 4, we adapt our proposed hill climbing algorithm in Section 4.3.3, to efficiently rescore speech lattices using global-linear models. Figure 5.2 shows the adapted algorithm for a global-linear model with features $\Phi(A, W)$ and corresponding weights $\vec{\alpha}$.

It may be noted that the only modification to the original hill climbing algorithm in 4.3.3 is the way the word sequences in each generated neighborhood are rescored through the use of the new global-linear model. Also, $\mathcal{L}$ denotes the WFSA representation of the first pass lattice, whose weights are the total baseline score. These weights are used as a baseline feature for rescoring the neighborhoods, and also for pruning the neighborhoods.

It may also be noted that we do not need to use LexicographicWeights, as was the case in Section 4.3.3. During rescoring of the neighborhoods under our global-linear model, we only need the total first pass score as the baseline feature, as opposed to LM rescoring in Chapter 4 where we needed to add the new LM score to the baseline acoustic scores.

5.4 Discriminative Hill Climbing Training

We are now ready to present our discriminative hill climbing algorithm. Just as hill climbing can improve over N-best lists for rescoring, it can be used to improve discriminative LM training. The key insight is that discriminative learning, to select the

Initialization:
  $\mathcal{L} \leftarrow$ **WFSA** of the initial (first-pass) lattice
  $\theta \leftarrow$pruning threshold
  $\widehat{W} \leftarrow$Viterbi path of the initial lattice [**Initilization**]
  $N \leftarrow$ **length**(W)
repeat
  $i \leftarrow 1$ [Start at position 1]
  while $i \leq N + 1$ **do**
    [**Neighborhood generation with pruning**]
    create **FSA** $LC(\widehat{W}, i)$
    $LN(\mathbf{w}, i) \leftarrow LC(\widehat{W}, i) \circ \mathcal{L}$
    $LN(\mathbf{w}, i) \leftarrow \text{prune}(LN(\widehat{W}, i), \theta)$
    [**Neighborhood rescoring per Eqn. 5.4**]
    $\widehat{W} \leftarrow \arg\max_{W' \in LN(\widehat{W}, i)} \langle \vec{\alpha}, \Phi(A, \widehat{W}) \rangle$
    [Adjust length and position]
    $N \leftarrow$**length**$(\widehat{W})$
    if DELETION **then**
      $i \leftarrow i$-1
    end if
    $i \leftarrow i$+1
  end while
until $\widehat{W}$ does not change [Stopping criterion]

Figure 5.2: Hill climbing algorithm using *global-linear* models for a given utterance with acoustic sequence A and first pass (baseline) lattice $\mathcal{L}$.

overall best hypothesis over all incorrect hypotheses, and hill climbing, which moves towards the best sequence, can be integrated by learning how to take incremental steps towards the best hypothesis in each neighborhood.

Observe that during N-best training, $\widehat{W}$ is chosen after scoring N hypotheses, which means we cannot observe a mistake until all N sequences are scored. The result is that if model parameters lead to an error in one word position of the sequence, the model learns to correct this error only by updating based on the entire sequence. In contrast, hill climbing produces $\widehat{W}$ incrementally, which means that a scoring error

Baseline:	ABORTION LEGAL TEAM IS ASKING THE COURT TO ORDER THEM RECOUNTING OF BALLOTS
(1)	AL GORE'S LEGAL TEAM IS ASKING THE COURT TO ORDER THEM RECOUNTING OF BALLOTS
(2)	ABORTION LEGALS ASKING THE COURT TO ORDER THEM RECOUNTING OF BALLOTS
(3)	ABORTION LEGALS TEA ASKING THE COURT TO ORDER THEM RECOUNTING OF BALLOTS
⋮	
(10)	AL GORE'S LEGAL TEAM IS ASKING THE COURT TO ORDER THE RECOUNTING OF BALLOTS

Figure 5.3: An example utterance corrected by our discriminative hill climbing algorithm. The first line (baseline) shows the word sequence provided by the baseline model. Errors are shown in red. After the first pass through the sequence, the algorithm inserts the word AL and substitutes ABORTION the word GORE'S. The second pass replaces THEM with THE, yielding the correct transcription.

can be observed before the final $\widehat{W}$ is reached. By updating parameters $\vec{\alpha}$ as soon as an error is made, we can make model corrections faster and learn how to move in the right direction within a neighborhood. Therefore, if the model is prone to making an error in one position, we need not wait for the production of an final highest scoring path, but instead correct model parameters as soon as we observe the initial mistake.

As an illustration, consider the example utterance in Figure 5.3. The baseline model produces an incorrect word sequence (Figure 5.3 (baseline)), which mistakes the word "ABORTION" for "AL GORE'S" and "THEM" for "THE". If Figure 5.3(10) was the best path in an N-best list, standard N-best list training would update only after all preceding paths have been scored. Instead, the hill climbing algorithm gradually corrects the baseline sequence, traversing the word sequence left-to-right and generating the corresponding local neighborhoods. If "AL GORE'S" gets corrected in the

first iteration of hill climbing, even though "THEM" is still error, the algorithm could immediately update, encouraging hill climbing to move in the better direction of hypothesis (10) instead of hypothesis (2). In other words, we can immediately "jump" to hypothesis (10) without scoring the hypotheses in between.

Initialization:
$\mathcal{L} \leftarrow$ **WFSA** of the initial (first-pass) lattice
$\theta \leftarrow$ neighborhood pruning threshold
$\widetilde{W} \leftarrow$ Viterbi path of the initial lattice
$W \leftarrow$ reference word sequence
$N \leftarrow length(\widetilde{W})$
Algorithm:
repeat
$i \leftarrow 1$
while $i \leq N + 1$ **do**
[**Neighborhood generation with pruning**]
create **FSA** $LC(\widetilde{W}, i)$
$LN(\widetilde{W}, i) \leftarrow LC(\widetilde{W}, i) \circ \mathcal{L}$
$LN(\widetilde{W}, i) \leftarrow \text{prune}(LN(\widetilde{W}, i), \theta)$

$\widehat{W} \leftarrow \arg\max_{W' \in LN(\widetilde{W},i)} \langle \vec{\alpha}, \Phi(A, W') \rangle$ [**Best path according to model**]
$W^* \leftarrow \arg\min_{W' \in LN(\widetilde{W},i)} d'(W', W)$ [**Best path according to edit-distance to the reference transcription**]

if $W^* \neq \widehat{W}$
for $j : 1..d$
$\alpha_j = \alpha_j + \Phi_j(A, W^*) - \Phi_j(A, \widehat{W})$
end if
$\widetilde{W} \leftarrow W^*$

$N \leftarrow length(\widetilde{W})$ [Adjusting length and position]
if DELETION
$i \leftarrow i$-1
$i \leftarrow i$+1
end while
until $\widetilde{W}$ does not change [Stopping criterion]

Figure 5.4: Discriminative hill climbing training for *global-linear* models for a given training utterance with acoustic sequence $\mathbf{a}$, reference $\mathbf{s}$ and lattice $\mathcal{L}$.

The general discriminative hill climbing procedure is show in Figure 5.4. For each training utterance (given W and $\mathcal{L}(A)$, the first pass lattice), select the Viterbi path in $\mathcal{L}$ as $\widetilde{W}$. Hill climbing starts improving from $\widetilde{W}$ using current model parameters $\vec{\alpha}$. Select a position i in $\widetilde{W}$ (we start from 1 and move left to right) and use the algorithm in Section 4.3.3 to generate the pruned neighborhood $LN(\widetilde{W}, i)$. Select the best path $\widehat{W} \in LN(\widetilde{W}, i)$ by extracting features $\Phi(A, W)$, and then scoring with $\vec{\alpha}$ for each path W in the neighborhood. Before continuing to the next sequence position, we evaluate if $\widehat{W}$ was the best choice given the reference W. Since, W may not be in the neighborhood, we instead use the path $W^* \in LN(\widetilde{W}, i)$ closest to W, using edit distance.

To determine if the current hill climbing direction improves the best path with respect to the reference, we compare $\widehat{W}$ with W^*. If $\widehat{W} \neq W^*$, then the algorithm is not moving in the optimal direction. In this case, this means that either W^* correctly inserts, deletes or substitutes words at position i of $\widetilde{W}$ as compared to $\widehat{W}$ or W^* has a better baseline score (The reason for this is described below). Therefore, model parameters $\vec{\alpha}$ should be updated to prefer W^* more, and $\widehat{W}$ less, which we achieve through a Perceptron update:

$$\alpha_j = \alpha_j + \Phi_j(A, W^*) - \Phi_j(A, \widehat{W}) \ , \tag{5.6}$$

where α_j is the jth parameter. Notice that while $\Phi(A, W)$ is based on the entire

sequence, only parameters that change based on position i will be updated. After an update, the best sequence becomes the current word sequence: $\widetilde{W} \leftarrow W^*$. We iterate over all positions until hill climbing converges, i.e. no change is available in any position of the sequence. Similar to the regular perceptron algorithm, Figure 5.1, we process and iterate over training instances (A_i, W_i) one at a time.

To find the best path W^* according edit-distance, we enumerate over all the paths in the neighborhood $LN(\widetilde{W}, i)$ and find their edit distance w.r.t the reference transcription W by composing the finite-state transducer (FST) representation of the path with *levenshtein* FST, and then composing the resulting FST with the FST representation of W. To handle situations where there is no single best path in terms of edit distance w.r.t W in the neighborhood, we use the first pass baseline score to break the tie and assign the best path W^*. That is, the distance function d' in Figure 5.4 has the following natural order definition for comparing two word sequences W_1' and W_2',

$$d'(W_1', W) < d'(W_2', W) \text{ iff } \begin{cases} d(W_1', W) < d(W_2', W) \\ \text{or} \\ d(W_1', W) = d(W_2', W) \ \& \ \Phi_0(A, W_1') > \Phi_0(A, W_2') \end{cases} \tag{5.7}$$

where Φ_0, as defined before, is the first pass baseline score. In other words, for optimizing $\arg\min_{W' \in LN(\widetilde{W}, i)} d'(W', W)$ in Eqn. 5.4, we use the comparison function above.

The choice of distance function in Eqn. 5.7 and the fact that we assign W^* as the current path at each step provides us with the fixed set of encountered neighborhoods while performing discriminative hill climbing for a training utterance at each iteration of the perceptron algorithm. This enables us to prove the convergence of the algorithm, as we will do in Section 5.4.1. Moreover, fixing the set of visited neighborhoods as above, provides the ability to extract them offline for each training sample and pre-process all the word sequences included in the neighborhoods[3].

In Section 5.7, we will empirically demonstrate that discriminative training via hill-climbing requires the evaluation of far fewer word sequences, leading to a dramatic reduction in the number of calls to the parser, in case of building syntactic discriminative model. Additionally, since we parse many word sequences in a neighborhood, we could obtain further efficiencies using substructure sharing among similar word sequences.

5.4.1 Convergence of the Training Algorithm

Here, we aim to provide a proof for the convergence of the algorithm on training samples. For this, we use and follow the notation developed in [85]. The key observation for proving the convergence of our proposed discriminative hill climbing is that the sequence of extracted neighborhoods $LN(\widetilde{W}, \bullet)$ for a given training sample $\mathcal{L}(A)$ is fixed and does not change through the different iterations of the algorithm. To see

[3]This is actually one of the advantages of the N-best perceptron where one can pre-process the extracted N-best lists for each training example.

why this is so, note that initially, $\widetilde{W}$ is the Viterbi path in $\mathcal{L}(A)$, and subsequently, we set $\widetilde{W} \leftarrow W^*$, which depends only on the neighborhood of of $\widetilde{W}$, and not on the $\widehat{W}$ chosen by the current iterate $\vec{\alpha}$.

To be more concrete, let us denote $\{A_i\}|_{i=1}^{M}$ to be the M training acoustic inputs. Also, let us use N_i to represent the number of extracted neighborhoods for the input sample A_i. Therefore, the j^{th} neighborhood for the training input A_i is serving as our candidate list which we represent using $\mathbf{GEN}_j(A_i)$. Note that $\mathbf{GEN}_j(A_i)$ is indeed $LN(\widetilde{W}_{ij}, \bullet)$, the j^{th} visited neighborhood for the input sample A_i. W_{ij}^* also denotes the best path in $\mathbf{GEN}_j(A_i)$ (cf. Section 5.4). Using this notation, our proposed discriminative hill climbing algorithm can be treated as a regular perceptron algorithm with the following input training samples:

$$\left\{\{A_1, W_{1j}^*\}|_{j=1}^{N_1}, \{A_2, W_{2j}^*\}|_{j=1}^{N_2}, \cdots, \{A_M, W_{Mj}^*\}|_{j=1}^{N_M}\right\} \tag{5.8}$$

where $\{A_i, W_{ij}^*\}|_{j=1}^{N_i}$ denotes the sequence of samples considered for the acoustic input A_i. Each training sample $\{A_i, W_{ij}^*\}$ has $\mathbf{GEN}_j(A_i)$ as its candidate list and the goal of perceptron training is to estimate the parameters of the global-linear model such that W_{ij}^* is selected as best path in $\mathbf{GEN}_j(A_i)$ candidate list. In addition, the total number of training samples M_{DHC} for our algorithm is

$$M_{\text{DHC}} = N_1 + N_2 + \cdots + N_M \tag{5.9}$$

Considering the notion of training samples explained above, the proof of convergence for the proposed discriminative hill climbing training algorithm under linear separability condition can be directly implied using the proof of convergence explained in [85] for the regular perception algorithm. However, for the sake of completeness, we briefly describe the proof here. First, let us recall the definition **linear separability** from [85].

Definition Let $\overline{\mathbf{GEN}_j(A_i)} = \mathbf{GEN}_j(A_i) - \{W_{ij}^*\}$. In other words, $\overline{\mathbf{GEN}_j(\mathbf{a}_i)}$ is the set of incorrect candidates for the j^{th} neighborhood of the acoustic input A_i. A training set in (5.8) is **separable with margin** $\delta > 0$ if there exists some vector $U \in \mathbb{R}^d$ with $||U|| = 1$ such that

$$\forall i, j \ \forall W \in \overline{\mathbf{GEN}_j(A_i)}, \quad \langle U, \big(\Phi(A_i, W_{ij}^*) - \Phi(\mathbf{a}_i, W)\big)\rangle \geq \delta \tag{5.10}$$

The convergence theorem for perceptron training may be stated as,

Theorem 1 ([85]) *If the training set of (5.8) is separable with margin δ, then the discriminative hill climbing algorithm of Figure 5.4 satisfies*

$$\textit{Number of error} \leq \frac{R^2}{\delta^2} \tag{5.11}$$

where R is a constant s.t $\forall i, j, \ \forall W \in \overline{\mathbf{GEN}_j(A_i)} \quad ||\Phi(A_i, W_{ij}^*) - \Phi(A_i, W)|| \leq R$. Also, an error occurs whenever $\widehat{W}_{ij} \neq W_{ij}^*$ for some (i,j) pair in the training

iterations.

We refer the readers to [85] for the proof of the theorem above.

Implication for discriminative hill climbing: Note that (5.11) asserts that the total number of error encountered during the training iterations is finite. The algorithm of Figure 5.4, however, keeps iterating as long as it encounters errors, i.e. as long as $\exists i, j$ for which $\widehat{W}_{ij} \neq W^*_{ij}$. Therefore, the theorem above essentially states that if there exists a parameters value $\vec{\alpha} = U$ which makes zero errors on training samples, our discriminative hill climbing algorithm converges to that parameter value. As stated in [85], the proof for this theorem is independent of the size and structure of $\overline{\mathbf{GEN}_j(A_i)}$ and only depends on the separability defined above.

5.4.2 Advantages of the Algorithm

Here, we discuss and summarize some of the advantages of our proposed algorithm over the conventional perceptron training using N-best lists:

- We believe our approach is well suited to speech lattices. Normal perceptron training updates using the optimal path in the N-best list, but it may be unrealistic for the model to select the optimal sequence from the N-best and apply a one-shot update towards the path, which leads to overly aggressive learning. Instead, our method encourages gradual improvement towards the best sequence, which will be selected if available. Otherwise, the model is still trained to improve at as many word positions in the sequence as possible. This may be more compatible with

directly minimizing WER, for we correct as many word positions as possible, as opposed to conventional Perceptron training which favors sentence error rate by targeting the best sequence overall.

Figure 5.5 pictorially illustrates this point where the longer line depicts the N-best perceptron update which tries to replace the initial solution $\widetilde{W}$ (first pass solution) with the optimal solution in the list W^*. In the figure, shorter lines correspond to the local updates of our discriminative hill climbing and the gradual improvements towards the optimal solution. We demonstrate empirically that in addition to being more efficient, the gradual updates results in a better learning algorithm. It is also worth mentioning that by applying the pruning scheme (Section 4.3.2) to the neighborhoods, we make sure that we only update towards the *reachable* solutions in the space. This way, the solutions that score poorly under the baseline recognizer are avoided, even though they might be good w.r.t the reference.

- In the conventional N-best perceptron training, there is always a concern about choosing an optimal N for the training task. On one hand, we want to have a large enough N to include good solutions in the lattice and to learn enough parameters. But at the same time, increasing N requires us to extract features and score a larger number of word sequences, which could ultimately make the training/rescoring procedure slow. In the proposed algorithm, on the other hand, there is no explicit tuning for the number of word sequences extracted from the speech lattices. The algorithm **dynamically** chooses the number of word sequences for each utterance,

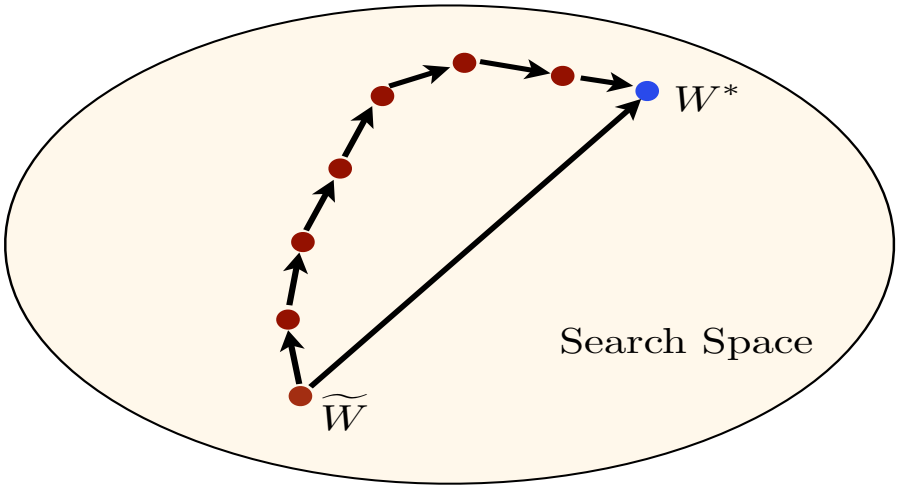


Figure 5.5: Pictorial illustration of the N-best perceptron updates vs. discriminative hill climbing updates.

taking into account utterance specific factors such as the length and the entropy of the baseline recognizer given the input acoustic signal[4].

- Finally, the proposed algorithm results in feature extraction (i.e. parsing and POS-tagging) for far fewer sentence-level feature than the N-best training. Therefore, it is more efficient to be deployed as an *online learning* algorithm in scenarios where the ASR system has access to newly corrected/transcribed training examples, e.g. voice search [86] applications.

5.4.3 Averaging Parameters

Following [85], we use the "voted perceptron" [87] by averaging the parameters of the global linear model used at each iteration of the algorithm. Define $\vec{\alpha}^{t,j,i}$ to be

[4]The higher the entropy of the baseline recognizer is, the denser the generated neighborhoods are and therefore, the number of extracted paths from the lattice is larger.

the parameter vector after the j^{th} neighborhood of the i^{th} training sample has been processed in pass t over the training data. Then, averaging outputs

$$\vec{\alpha}_{\text{avg}} = \frac{1}{M_{\text{DHC}}T} \sum_{t=1}^{T} \sum_{i=1}^{M} \sum_{j=1}^{N_i} \vec{\alpha}^{t,j,i}, \tag{5.12}$$

where M_{DHC} is defined in (5.9). The averaged perceptron is efficient to implement and is typically used in various NLP tasks [37, 85, 88].

5.4.4 Additional Efficiencies

Further speedups can be obtained by parallelization based on the distributed Perceptron of [89] (*iterative parameter mixing* with uniform weights) where the training examples $\{(A_i, W_i)\}|_{i=1}^{M}$ are divided among S parallel threads. A single epoch of discriminative hill climbing is trained on each thread in parallel, mixing the model weights $\vec{\alpha}$ after each iteration. The mixed parameter vector is then used by each machine to start another single epoch of training. The parameter values are uniformly mixed at the end of each iteration: $\vec{\alpha}_j = \frac{1}{S} \sum_{s=1}^{S} \vec{\alpha}_j^s$,where j and s indicate the iteration and machine respectively. In this dissertation, we do not attempt to apply the above parallel training.

For additional efficiency, we pre-process all the extracted neighborhoods and extract all the required features for all the paths in the neighborhoods. In case of syntactic features, this requires us to POS-tag and parse the word sequences. For each

utterance A_i, we store a record of every sequence and its features for all the unique word sequences encountered in any of the generated neighborhoods $\{LN(\widetilde{W}_{ij}, \bullet)\}$ of that utterance and use these stored features during processing the utterance at each iteration of the algorithm. Recall that the neighborhoods for each utterance A_i are fixed during the training.Some efficiency results from a word-sequence appearing in multiple neighborhoods of a given utterance. Note that caching cannot be used for N-best list training since every N-best entry is unique and must be parsed before training.

5.5 Related Work

Our approach is reminiscent of learning algorithms that improve the quality of the sequence-valued output through iterative updates, e.g a Monte Carlo technique for decoding phrase based translation lattices is used in [90], and bi-directional learning uses a Perceptron to iteratively label each position of a word sequence, updating parameters each time an incorrect label is applied [91].

One of the advantages of the proposed algorithm is that the baseline output is gradually improved towards the best solution via local updates at each step, as described in Section 5.4.2. It has been observed by Liang et al. [92] that such conservative local updating results in a better discriminative model for machine translation task. They use a similar hill climbing approach, where at each iteration they produce

a small N-best (10 best) list of translation outputs and apply updates towards the best solution in the list. For each iteration of training, they use the updated model to re-generate the N-best list[5].

Recently, there has been a trend towards using "semi-supervised" discriminative training of language models for ASR [93] [94] [95] [96]. Recall that discriminative training methods rely on manually transcribed speech as training examples. As a result the amount of training data available for building discriminative models is limited, as opposed to training generative models, which only require text. Semi-supervised techniques for training discriminative LMs address this issue by applying the discriminative training to data that consists of just text. All these semi-supervised methods are based on the idea of simulating ASR outputs that would potentially compete with the training text data, and use the pseudo outputs as a candidate list for training a discriminative model. The main difference among these methods is the technique applied to generate the simulated ASR outputs and errors. Kurata et al. [94] uses a weighted finite-state transducer based method to represent the acoustic phonetic similarities and generate confusable strings for each given sentence in a text corpus, which creates training data for the discriminative training. In the process, they produce "pseudo-lattices" by composing the machine representing the acoustic confusions with the phonetic representation of the input text (which is obtained using

[5]In our case, our model works as a re-ranking discriminative model on a static space produced by the first pass recognizer (lattice). We hill climb on this space during training using a gradual steps towards locally good solutions in the space. Liang et al. [92] instead dynamically generate the local neighborhood at each iteration of the perceptron algorithm.

a pronunciation lexicon). Xu et al. [97] simulate word-level acoustic confusions, and Sagae et al. [96] go further and use phrasal confusions (cohorts).

We want to emphasis that the proposed discriminative hill climbing can also be applied on such pseudo-lattices to efficiently train a discriminative model with long-span features on vast amount of available text data.

5.6 Experimental Setup

The ASR system used throughout the experiments of this chapter is same as the one used in Section 4.5, which is based on the 2007 IBM Speech transcription system for the GALE Distillation Go/No-go Evaluation [38]. The acoustic models are state of the art discriminatively trained models trained on Broadcast News (BN) Hub4 acoustic training data, which contains about $153K$ utterances and transcriptions, about 400 hours of speech. To train the discriminative language model we use Hub4 utterances with length < 50, giving $115K$ training utterances ($2.6M$ words). The baseline language model is a modified Kneser-Ney backoff 4-gram model trained on about 195M words from Hub4 acoustic model training transcripts, closed captions from the TDT4 , and EARS BN03 sources.

To produce word-lattices, each training utterance is decoded/processed by the baseline ASR system. The LM used for producing the first pass lattices is a Kneser-Ney 3-gram which is trained on the LM training data excluding the Hub4 acoustic

model training transcripts. The reason to use a 3-gram LM is to make sure we get enough acoustic confusions in the discriminative training data. The test set is the 4 hour rt04 Broadcast News evaluation set ($45K$ words). The dev set for tuning the *baseline feature* weight (α_0) is the dev04f Broadcast News data.

To train a discriminative syntactic LM we use the local perceptron POS-tagger of [60] and the dependency parser of [43] which is based on a maximum-entropy classifier (described in Section 3.2.1). Since the LM here is discriminative, as opposed to the generative SLM of Chapter 3, we include the full feature sets in both the parser and POS-tagger. That is, the lookahead features that were excluded from the parser and POS-tagger for experiments of Chapter 3, are now fully included in these models. The POS-tagger has accuracy of about 96.9% on broadcast news treebank test set [1]. Also, the dependency parser has LAS accuracy of about 88.5% on the BN treebank test data[6].

We parse all word sequences in the extracted neighborhoods of the training data. Our syntactic global linear model includes the following features at each word position i:

1. $(h_{-2}.\mathrm{w} \circ h_{-1}.\mathrm{w} \circ w_i)\,,\ (h_{-1}.\mathrm{w} \circ w_i)\,,\ (w_i)$
2. $(h_{-2}.\mathrm{t} \circ h_{-1}.\mathrm{t} \circ t_i)\,,\ (h_{-1}.\mathrm{t} \circ t_i)\,,\ (t_i)\,,\ (t_i w_i)$

[6]We will show in Chapter 6 how this level of accuracy for both the parser and POS-tagger is achieved thorough uptraining on broadcast news data.

where h_{-1} and h_{-2} correspond to first and second exposed headword in the history at position i, respectively. w and t denote the word identity and the POS tag of the corresponding exposed headword. The exposed headwords at each position are extracted from the dependency tree using the method described in 2.3.1.1. We again use the sentence segmentation based on end of sentence token, "</s>", described in Section 4.7.2 for POS-tagging and parsing the ASR hypotheses which include more than one sentence.

In addition, we build a discriminative LM with n-gram features only to calibrate the improvements from adding syntactic features to the LM. Specifically, at each word position i the following n-gram features are included in the n-gram discriminative model:

$$3.\ \ (\mathrm{w}_{i-2} \circ \mathrm{w}_{i-1} \circ \mathrm{w}_i)\ ,\ (\mathrm{w}_{i-1} \circ \mathrm{w}_i)\ ,\ (\mathrm{w}_i)$$

The averaged perceptron is used for training the discriminative models.

5.7 Results and Analysis

We begin with an evaluation of discriminative hill climbing (DHC) in terms of WER improvements on the evaluation set. We train the discriminative syntactic LM and n-gram LM using (i) the proposed discriminative hill climbing algorithm and (ii) the conventional $N = 100$-best lists. The first pass (baseline) lattices of the rt04 test

	WER(%)	
Baseline ASR	15.1	
	Training Algorithm	
	DHC	**100-best**
3-gram features	14.9	14.9
Syntactic features	**14.6**	14.7

Table 5.1: rt04 test set results: rescoring a baseline recognizer lattices using a discriminative LM trained with either discriminative hill climbing or conventional $N = 100$-best training.

data are then rescored using the hill climbing rescoring algorithm (Figure 5.2) with 1 initial path (Section 5.3). During training we use $\alpha_0 = 1$ as the baseline feature weight. For rescoring, however, we tune this weight on the development set which results in $\alpha_0 = 0.7$ for both cases, the discriminative model with 3-gram features and the model with syntactic features. Table 5.1 presents the WER results for each of the models and the training algorithms. As seen in the table, the discriminative syntactic model trained using the proposed hill climbing algorithm improves the WER by 0.5% (a statistically significant result with $p < 0.005$ using matched pair (MP) sentence segment test). In addition, the WER reduction due to using syntactic features is about 0.3% absolute (14.9% versus 14.6%). Finally, the proposed discriminative training procedure results in slightly better performance than 100-best training for the syntactic LM.

The proposed discriminative training algorithm processes on average about 30 hypotheses per speech utterance during training (in case of including syntactic features). When compared to average of 100 hypotheses for the 100-best training, the

	Process Time (hours)	
	100-best	DHC
POS tagger	8.2	**3.2**
Parser	160.8	**64.3**

Table 5.2: Time spent by the POS-tagger and dependency parser for processing hypotheses extracted from candidate lists in discriminative hill climbing and $N = 100$-best training.

new algorithm results in a significant speedup. Table 5.2, shows the time spent by the POS-tagger and the dependency parser for processing (POS-tagging and parsing) word sequences extracted from the candidate lists of both training algorithms. Observe that the reduction in the average number of hypotheses examined almost proportionally reduces the processing time spent by the POS-tagger and the parser: Discriminative hill climbing results in a speedup of about 2.4 compared to 100-best list training. This enables the application of discriminative training in an online learning scenario.

We can also measure learning effectiveness by considering the WER of the training data after each iteration of discriminative hill climbing. But let us first, measure the WER of the locally optimal solution in the training data that each algorithm steps towards. The WER of the optimal solutions on the training data can be found in Table 5.3. The proposed discriminative hill climbing algorithm reaches hypotheses in the training lattices with a WER of about 6.0%, and it does so with only 30 hypotheses evaluations on average. To reach a comparable hypotheses via N-best rescoring, one needs to extract and process about 1000-best hypotheses on average.

Next, we measure the WER of the training data after each iteration of discrimi-

	# Hypotheses (averaged)	Optimal WER (%)
100-best	100	7.0
1000-best	1000	5.9
DHC	**30**	**6.0**

Table 5.3: The WER of the optimal solutions in the candidate list of $N = \{100, 1000\}$-best training vs. discriminative hill climbing training.

	WER (%) 3-gram features:$(\mathrm{w}_{i-2} \circ \mathrm{w}_{i-1} \circ \mathrm{w}_i), (\mathrm{w}_{i-1} \circ \mathrm{w}_i), (\mathrm{w}_i)$		
	100-best	1000-best	DHC
Iter. 1	12.4	13.7	**11.5**
Iter. 2	10.5	11.1	**9.5**
Iter. 3	9.6	9.9	**8.7**

Table 5.4: The WER of the training data using different discriminative training methods 3-gram features.

native training. This investigation has been carried out only for the model that uses n-gram features only. The training WERs are shown in Table 5.4. It can be observed that the discriminative hill climbing algorithm results in a more effective learning algorithm at least on the training data. An interesting point in the table is that 1000-best training converges more slowly than 100-best on the training data. This is in contrast with the fact that the optimal solution using 1000-best training has a much better WER, 5.9% versus 7.0% (Table 5.3). A possible explanation for this is the perceptron update rule is not directly minimizing the expected loss, which is WER, but sentence error rate, and a large N permits stepping towards solution for the latter that is worse for the former than would be the base if N was smaller [98][7].

[7]In cases where the training sequence is δ-separable (Section 5.4.1) the perception update rule will eventually lead to parameters with *zero loss*. This is not the case where zero loss is unachievable [98].

5.8 Conclusions

We have presented discriminative hill climbing, a new algorithm for discriminative training of long-span language models, which normally requires the use of N-best lists for training and rescoring. For a syntactic language model that relies on a parser, we demonstrate improvements in WER along with significant reduction in training complexity. We also observe WER improvements over N-best training given comparable computational effort. Our algorithm can be used to train complex language models on large corpora with state of the art parsers.

There are a number of promising avenues for future work, including considering more aggressive online algorithms, or scaling training to much larger corpora by utilizing more efficient parsers, including lattice parsers. Additionally, improved training time opens the door for exploring new LM features, and more complex models.

Chapter 6

Fast Dependency Parsing via Substructure Sharing and Uptraining

6.1 Introduction

In Chapter 3 we presented an efficient SLM framework for incorporating long-distance dependencies and syntactic structure to complement the predictive power of n-grams in a language model. Chapter 5, subsequently, introduced an efficient discriminative training algorithm, with which a global linear with long-span features may be trained to directly minimize recognition errors.

However, both generative and discriminative LMs with long-span dependencies

can be too slow for practical applications, for they often cannot work directly with lattices and require a second pass *rescoring* of large N-best lists [2, 6, 9]. For discriminative models, this limitation applies to training as well. Moreover, the non-local features used by these models are usually extracted via auxiliary tools – which in the case of syntactic features include part of speech taggers and parsers – from a set of ASR system hypotheses. Separately applying auxiliary tools to each hypothesis in the N-best list leads to further inefficiencies.

In Chapter 4, we presented a hill climbing rescoring algorithm for ASR lattices which uses an iterative search for a higher-scoring hypothesis in a local neighborhood of the current-best hypothesis, leading to a more efficient algorithm in terms of the number, N, of hypotheses evaluated; the idea also led to the discriminative hill climbing training algorithm of Chapter 5 for efficiently training long-span discriminative language models.

Even so, the reliance on auxiliary tools slow LM application down to the point of being impractical for real time systems. While faster auxiliary tools are an option, speed is often obtained by trading off accuracy of analysis.

In this chapter, we propose a general modification to the decoders used in such auxiliary tools without sacrificing accuracy by exploiting the commonalities among the hypotheses. The key idea is to share substructures in transition based structured prediction algorithms, i.e. algorithms where the final structure for an input sentence is composed of a sequence of multiple individual decisions. If two hypotheses in the

N-best list or in the local neighborhood during hill climbing share prefix or suffix, for instance, a significant portion of the syntactic analysis of the first could be reused for the second.

We demonstrate the utility of this idea on a local Perceptron based part of speech tagger [60] and a shift reduce dependency parser [43], yielding significantly faster tagging and parsing of ASR hypotheses.

In an orthogonal direction, we compensate for the model's simplicity of the utilized fast structured prediction tools by training the tools on order of magnitude more data via a procedure that has been called up-training [99], yielding auxiliary tools that are both fast and accurate. The result is significant speed improvements and a reduction in word error rate (WER) for both N-best list and the already fast hill climbing rescoring. The net result is arguably the first syntactic LM fast enough to be used in a *real time ASR system.*

6.2 Incorporating Syntactic Structures

Long-span language models – generative or discriminative, N-best or hill climbing – rely on auxiliary tools to implement associated models, such as a POS tagger or a parser, for extracting features for each hypothesis during rescoring, and during discriminative training. The top-m candidates for the auxiliary structure associated with the i^{th} hypothesis, which we denote as $\mathcal{S}_i^1, \ldots, \mathcal{S}_i^m$, are generated by these tools

and used to score the hypothesis: $F(W_i, \mathcal{S}_i^1, \ldots, \mathcal{S}_i^m)$. For example, $\mathcal{S}_i^j$ can be a POS tag or a syntactic dependency. We formally define this sequential processing as:

$$
\begin{aligned}
W_1 &\xrightarrow{\text{tool(s)}} \mathcal{S}_1^1, \ldots, \mathcal{S}_1^m \xrightarrow{\text{LM}} F(W_1, \mathcal{S}_1^1, \ldots, \mathcal{S}_1^m) \\
W_2 &\xrightarrow{\text{tool(s)}} \mathcal{S}_2^1, \ldots, \mathcal{S}_2^m \xrightarrow{\text{LM}} F(W_2, \mathcal{S}_2^1, \ldots, \mathcal{S}_2^m) \\
&\vdots \\
W_N &\xrightarrow{\text{tool(s)}} \mathcal{S}_N^1, \ldots, \mathcal{S}_N^m \xrightarrow{\text{LM}} F(W_k, \mathcal{S}_N^1, \ldots, \mathcal{S}_N^m)
\end{aligned}
$$

Here, $\{W_1, \ldots, W_N\}$ represents a set of ASR output hypotheses that need to be rescored. For each hypothesis, we apply an external tool (e.g. a parser) to generate associated structures $\mathcal{S}_i^1, \ldots, \mathcal{S}_i^m$ (e.g. dependencies.) These are then passed to the language model along with the word sequence for scoring. For the SLM model developed in Chapter 3, these structures are revealed as the partial trees (history prefix trees) for any word w_i in hypothesis $W = w_1 w_2 \cdots w_n$[1]. These partial trees at each position i along with the word are then used by the SLM sequentially to score a hypothesis, i.e. $F(W, \mathcal{S}) = \sum_i F(w_i, \mathcal{S}_i)$. The discriminative global linear model of Chapter 5, on the other hand, uses the full tree structure $T(W)$ to extract exposed headwords at each word position and uses them to score each hypothesis W. In either case, the parser must analyze all the hypotheses evaluated for the LM.

[1]Using the notation of Chapter 3, these structures were referred to as *history-states* and were represented by π_{-i}^j which shows the j^{th} partial tree at position i of W.

N-best list

(1)	AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE
(2)	TO AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE
(3)	AL GORE HAS PROMISE THAT HE WOULD ENDORSE A CANDIDATE
(4)	SO AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE
(5)	IT'S AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE
(6)	AL GORE HAS PROMISED HE WOULD ENDORSE A CANDIDATE
(7)	AL GORE HAS PROMISED THAT HE WOULD ENDORSE THE CANDIDATE
(8)	SAID AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE
(9)	AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE FOR
(10)	AL GORE HIS PROMISE THAT HE WOULD ENDORSE A CANDIDATE

Hill climbing neighborhood

(1)	YEAH FIFTY CENT GALLON NOMINATION WHICH WAS GREAT
(2)	YEAH FIFTY CENT A GALLON NOMINATION WHICH WAS GREAT
(3)	YEAH FIFTY CENT GOT A NOMINATION WHICH WAS GREAT

Figure 6.1: Example of repeated substructures in candidate hypotheses.

6.3 Substructure Sharing

While long-span LMs have been empirically shown to improve WER over n-gram LMs, their computational burden limits the widespread use of such LMs in practice, particularly in real-time systems. A major complexity factor is due to the need to process 100s or 1000s of hypotheses for each speech utterance, each of which must be POS tagged and parsed. However, the candidate hypotheses of an utterance share large substructures, especially due to the locality of the neighborhood generation in the hill climbing methods. Recall from Chapter 4, for instance, that by design, $N(W_j, \bullet)$ is a set of hypotheses that all differ from each other in only 1 or 2 words out of a typical length of 20 words.

Figure 6.1 demonstrates such repetition in a 10-best list and a hill climbing neigh-

borhood $N(W, 4)$ for utterances from the broadcast news corpus. For example, the word "ENDORSE" occurs within the same local context in all hypotheses in the 10-best list and should receive the same part of speech tag in each case. Processing each hypothesis separately is computationally inefficient.

We propose a general algorithmic approach to reduce the complexity of auxilliary processing by sharing common substructures among the hypotheses.

Unlike many lattice parsing algorithms, our approach is general and produces exact output. Lattice parsing has considered processing multiple hypotheses (including lattices and N-best lists) with the goal of producing the single-best parse trees over all hypotheses. In such schemes, some sentences are only parsed partially, and some possibly not at all. The goal here is to produce the m-best parses of each hypothesis in the ASR N-best list or lattice (neighborhood).

We first present our approach and then demonstrate its generality by applying it to a dependency parser and part of speech tagger. With the goal of developing a method that is widely applicable, we assume a structured prediction model that produces output from a series of local decisions: i.e a transition model. The model begins in initial state π_0 and terminate in a possible final state π_f. All states along the way are chosen from the set of possible states Π. A transition (or action) $\omega \in \Omega$ advances the decoder from state to state, where the transition ω_i changes the state from π_i to π_{i+1}. The sequence of states $\{\pi_0 \dots \pi_i, \pi_{i+1} \dots \pi_f\}$ corresponds to an output (the model's prediction.)

The choice of action ω is given by a statistical model, such as a maximum-entropy classifier, support vector machine or Perceptron, which is trained on labeled data. Given the previous k actions up to π_i, the classifier $g : \Pi \times \Omega^k \rightarrow \mathbb{R}^{|\Omega|}$ assigns a score to each possible action, which we can interpret as a probability: $p_g(\omega_i|\pi_i, \omega_{i-1}\omega_{i-2}\ldots\omega_{i-k})$. These actions are applied to transition to new states π_{i+1}. We note that state definitions may encode the k previous actions, which simplifies the probability to $p_g(\omega_i|\pi_i)$. The score of the new state is then

$$p(\pi_{i+1}) = p_g(\omega_i|\pi_i) \cdot p(\pi_i). \tag{6.1}$$

Classification decisions require a feature representation of π_i, which is provided by feature functions $\mathbf{f} : \Pi \rightarrow \mathcal{Y}$, that map states to features. Features are treated as a sufficient statistics for multi-class classification, so $p_g(\omega_i|\pi_i) = p_g(\omega_i|\mathbf{f}(\pi))$. In this sense, states can be summarized by features.

Equivalent states are defined as two states π and π' with an identical feature representation:

$$\pi \equiv \pi' \quad \text{iff} \quad \mathbf{f}(\pi) = \mathbf{f}(\pi').$$

If two states are equivalent, then g imposes the same distribution over actions. We can benefit from this substructure redundancy, both within and between hypotheses, by sharing the distribution computed just once for the entire set of equivalent states. A similar idea of equivalent states is used by Huang and Sagae [51], except they use

equivalence to facilitate dynamic programming for shift-reduce parsing (i.e. within-hypothesis sharing) whereas we generalize it for improving the processing of similar hypotheses in general models (i.e. across-hypotheses sharing). Following the notation of Huang and Sagae, we define *kernel features* as the smallest set of atomic features $\tilde{\mathbf{f}}(\pi)$ such that,

$$\tilde{\mathbf{f}}(\pi) = \tilde{\mathbf{f}}(\pi') \quad \Rightarrow \quad \pi \equiv \pi'. \tag{6.2}$$

Equivalent distributions are stored in a *hash table* $H : \Pi \rightarrow \Omega \times \mathbb{R}$; the hash keys are the states and the values are distributions[2] over actions: $\{\omega, p_g(\omega|\pi)\}$. H caches equivalent states in a hypothesis set and resets for each new speech utterance. When the structure $\mathcal{S}_i^j$ is being developed, for each state we encounter, we first check H for equivalent states before computing the action distribution; each cache hit reduces decoding time.

Distributing hypotheses W_i across different CPU threads is another way to obtain speedups, and we can still benefit from substructure sharing by storing H in shared memory. We use

$$h(\pi) = \sum_{i=1}^{|\tilde{\mathbf{f}}(\pi)|} \text{int}(\tilde{f}_i(\pi)) \tag{6.3}$$

as the hash function, where $\text{int}(\tilde{f}_i(\pi))$ is an integer mapping of the i^{th} kernel feature. For integer typed features the mapping is trivial, for string typed features (e.g. a POS tag identity) we use a mapping of the corresponding vocabulary to integers.

[2]For pure greedy search (deterministic search) we need only retain the best action, while the entire distribution is needed in probabilistic search, such as beam search or best-first algorithms.

We empirically found that this hash function is very effective and yielded very few collisions.

To apply substructure sharing to a transition based model, we need only define the set of states Π (including π_0 and π_f), actions Ω and kernel feature functions $\tilde{\mathbf{f}}$. The resulting speedup depends on the amount of substructure duplication among the hypotheses, which we will show is significant for ASR lattice rescoring. Note that our algorithm is not an approximation; we obtain the same output $\{\mathcal{S}_i^j\}$ as we would without any sharing.

We now illustrate how to apply this algorithm to dependency parsing and POS tagging.

6.3.1 Dependency Parsing

We use the best-first probabilistic shift-reduce dependency parser of Sagae and Tsujii [43], a transition-based parser [55] with a MaxEnt classifier[3]. Dependency trees are built by processing the words left-to-right and the classifier assigns a distribution over the actions at each step. States are defined as $\pi = \{S, Q\}$: S is a stack of subtrees $s_0, s_1, \ldots$ (s_0 is the top tree) and Q are words in the input word sequence. The initial state is $\pi_0 = \{\emptyset, \{w_0, w_1, \ldots\}\}$, and final states occur when Q is empty and S contains a single tree (the output).

The set of actions, Ω, is determined by the set of dependency labels $r \in \mathbb{R}$ and

[3]Part of the description for the shift-reduce dependency parser is repeated from Section 3.2.1 to have a complete overview of how the substructure sharing can be integrated into the parser decoder.

one of three transition types:

- `Shift`: remove the head of Q and place it on the top of S as a singleton tree.
- `Reduce-Left`(r): replace the top two trees in S (s_0 and s_1) with a tree formed by making the root of s_1 a dependent of the root of s_0 with label r.
- `Reduce-Right`(r): same as *Reduce-Left*(r) except making the root of s_0 a dependent of the root of s_1.

Finally, the feature functions bring to the model's attention attributes of the current state S and queue Q that have the most bearing on syntactic structure. e.g. a feature found useful is

$$f = \begin{cases} 1 & \text{if } s_0.\text{w}{=}w \ \&\ q_0.\text{t}{=}\text{P} \\ 0 & \text{otherwise} \end{cases}$$

where s_0.w denotes the head word of the top tree in the current stack and q_0.t represent the POS tag of the first word in the current queue.

Table 6.1 shows the kernel features. See Sagae and Tsujii [43] for a complete list of features. Essentially the extracted features from each state are either one of the kernel features or are obtained by conjoining a set of kernel features. As one example the following is a feature based on conjoining the POS tags of s_0, s_1, and s_2:

$$f = (s_0.\text{t} = NN) \wedge (s_1.\text{t} = VP) \wedge (s_3.\text{t} = IN)$$

Kernel features $\tilde{\mathbf{f}}(\pi)$ for state $\pi = \{S, Q\}$ $S = s_0, s_1, \ldots$ & $Q = q_0, q_1, \ldots$					
(1)	s_0.w	s_0.t	$s_0.r$	(5)	t_{s_0-1}
		s_0.lch.t	s_0.lch.r		t_{s_1+1}
		s_0.rch.t	s_0.rch.r		
(2)	s_1.w	s_1.t	$s_1.r$	(6)	dist(s_0, s_1)
		s_1.lch.t	s_1.lch.r		dist(q_0, s_0)
		s_1.rch.t	s_1.rch.r		
(3)	s_2.w	s_2.t	$s_2.r$		
(4)	q_0.w	q_0.t		(7)	s_0.nch
	q_1.w	q_1.t			s_1.nch
	q_2.w				

Table 6.1: Kernel features for defining parser states. s_i.w denotes the head-word in a subtree and t its POS tag. s_i.lch and s_i.rch are the leftmost and rightmost children of a subtree. $s_i.r$ is the dependency label that relates a subtree head-word to its dependent. s_i.nch is the number of children of a subtree. q_i.w and q_i.t are the word and its POS tag in the queue. dist(s_0,s_1) is the linear distance between the head-words of s_0 and s_1 in the word sequence and dist(q_0,s_0) is the linear distance between the head-word of s_0 and the first word in the queue q_0.

As many features are constructed by conjoining the kernel feature, the number of actual features is larger than kernel features. Therefore, the caching based on the kernel features is more efficient than using the actual features.

Goldberg and Elhadad [100] observed that parsing time is dominated by feature extraction and score calculation. Substructure sharing brings efficiencies to these very steps for equivalent states, which are persistent throughout a candidate set. Note that there are far fewer kernel features than total features, hence the hash function calculation (6.3) is very fast.

We summarize substructure sharing for dependency parsing in Algorithm 5. We extend the definition of states to be $\{S, Q, p\}$ where p denotes the score of the state:

Algorithm 5 Best-first shift-reduce dependency parsing

$\mathbf{w} \leftarrow$ input hypothesis
$S_0 = \emptyset, Q_0 = \mathbf{w}, p_0 = 1$
$\pi_0 \leftarrow \{S_0, Q_0, p_0\}$ [initial state]
$H \leftarrow$Hash table $(\Pi \rightarrow \Omega \times \mathbb{R})$
Heap$\leftarrow$ Heap for prioritizing states and performing best-first search
Heap.push(π_0) [initialize the heap]

while Heap $\neq \emptyset$ **do**
  $\pi_{\text{current}} \leftarrow$Heap.pop() [the best state so far]
  if $\pi_{\text{current}} = \pi_f$ [if final state]
    return π_{current} [terminate if final state]
  else if H.find(π_{current})
    ActList $\leftarrow H[\pi_{\text{current}}]$ [retrieve action list from the hash table]
  else [need to construct action list]
    for all $\omega \in \Omega$ [for all actions]
      $p_\omega \leftarrow p_g(\omega|\pi_{\text{current}})$ [action score]
      ActList.insert($\{\omega, p_\omega\}$)
    H.insert(π_{current}, ActList) [Store the action list into hash table]
  end if
  for all $\{\omega, p_\omega\} \in$ ActList [compute new states]
    $\pi_{\text{new}} \leftarrow \pi_{\text{current}} \times \omega$
    Heap.push(π_{new}) [push to the heap]
end while

the probability of the action sequence that resulted in the current state. Also, following Sagae and Tsujii [43] a heap is used to maintain states prioritized by their scores, for applying the best-first strategy. For each step, a state from the top of the heap is considered and all actions (and scores) are either retrieved from H or computed using g. Sagae and Tsujii [43] use a beam strategy to increase speed. Search space pruning is achieved by filtering heap states for probability greater than $\frac{1}{b}$ the probability of the most likely state in the heap with the same number of actions. We use $b = 100$ for our experiments. We use $\pi_{\text{new}} \leftarrow \pi_{\text{current}} \times \omega$ to denote the operation of extending a state by an action $\omega \in \Omega$.

Finally, we note that while we have demonstrated substructure sharing for **dependency** parsing, the same improvements can also be made to a shift-reduce constituent parser [54].

6.3.2 Part of Speech Tagging

We use the part of speech (POS) tagger of Tsuruoka et al. [60], a transition based model with a Perceptron and a lookahead heuristic process. The tagger processes W left to right. States are defined as $\pi_i = \{c_i, W\}$: a sequence of assigned tags up to w_i ($c_i = t_1 t_2 \dots t_{i-1}$) and the word sequence W. Ω is defined simply as the set of possible POS tags that can be applied. The final state is reached once all the positions are tagged. For $\mathbf{f}$ we use the features of [60]. The kernel features are $\tilde{\mathbf{f}}(\pi_i) = \{t_{i-2}, t_{i-1}, w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}\}$. While the tagger extracts prefix and suffix features, it suffices to look at w_i for determining state equivalence. The tagger is deterministic (greedy) in that it only considers the best tag at each step, so we do not store scores. However, this tagger uses a depth-first search lookahead procedure to select the best action at each step, which considers future decisions[4] up to depth d. An example for $d = 1$ is shown in Figure 6.2. Using $d = 1$ for the lookahead search strategy, we modify the kernel features since the decision for w_i is affected by the

[4]Tsuruoka et al. [60] shows that the lookahead search improves the performance of the local "history-based" models for different NLP task.

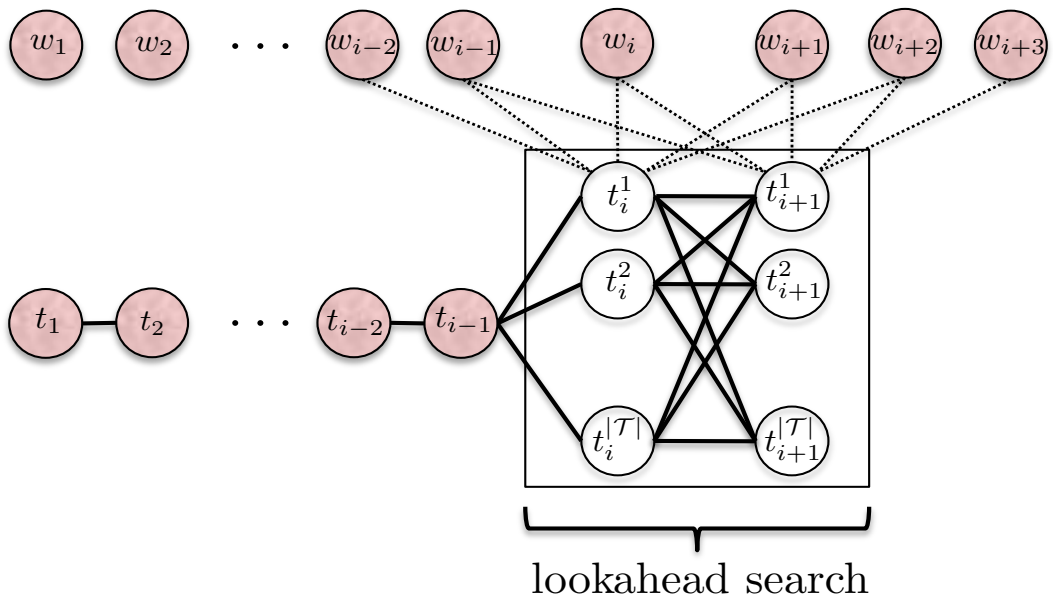


Figure 6.2: POS tagger with lookahead search of d=1. At w_i the search considers the current state and next state.

state π_{i+1}. The kernel features in position i should be $\tilde{\mathbf{f}}(\pi_i) \cup \tilde{\mathbf{f}}(\pi_{i+1})$[5]:

$$\tilde{\mathbf{f}}(\pi_i) = \{t_{i-2}, t_{i-1}, w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, w_{i+3}\}.$$

6.3.3 SLM Substructure Sharing

As described in Section 3.3, the SLM — as is the case with any generative left-to-right LM — can only use features from the history for predicting a word position w_i in a word sequence W, i.e. $p(w_i|W_{-i})$. Therefore, the associated structures of the history realized by the auxiliary tools — the POS tagger and the dependency

[5]Also the lookahead search strategy can not be utilized in the SLM scenario.

parser— should also be only a function of the history W_{-i}. In other words, to build a proper generative probabilistic model, we *can not* incorporate the **lookahead** features (features that are based on word positions to the right of w_i in W) of the POS tagger and the dependency parser. We refer to this scenario for both the parser and the POS tagger as the **no-lookahead** mode. For the POS tagger (described in Section 6.3.2) precluding the lookahead features results in the following kernel features[6]:

$$\tilde{\mathbf{f}}(\pi_i) = \{t_{i-2}, t_{i-1}, w_{i-2}, w_{i-1}\}.$$

The kernel features of the dependency parser in the no-lookahead mode are shown in Table 6.2. As seen from this table – when compared to Table 6.1 – all the queue dependent kernel features (q_0,q_1, and q_2) are excluded from the table. With the kernel features for the POS tagger and the dependency parser, we demonstrate how to utilize the substructure sharing to efficiently apply the SLM to set of hypotheses of an ASR output.

Remark It is worth mentioning that for rescoring ASR hypotheses using the SLM the *best-first* search strategy of Section 6.3.1 can not be utilized by the dependency parser; This is due to the fact that the use of best-first search algorithm makes the parser history-states for a position i be dependent on the part of the sentence that is happening after word position i. For example, if the rest of the sentence is easy

[6]Since the POS tagger in the no-lookahead mode uses the features only based on bi-gram tags and two previous words, there is no need to apply the substructure sharing in this mode;In fact, the no-lookahead POS tagger can be directly applied to the lattices to extract POS tags on each arc.

Kernel features $\tilde{\mathbf{f}}(\pi)$ for state $\pi = \{S, Q\}$ $S = s_0, s_1, \ldots$ & $Q = q_0, q_1, \ldots$					
(1)	s_0.w	s_0.t	$s_0.r$	(5)	t_{s_0-1}
		s_0.lch.t	s_0.lch.r		t_{s_1+1}
		s_0.rch.t	s_0.rch.r		
(2)	s_1.w	s_1.t	$s_1.r$	(6)	dist(s_0, s_1)
		s_1.lch.t	s_1.lch.r		
		s_1.rch.t	s_1.rch.r		
(3)	s_2.w	s_2.t	$s_2.r$	(7)	s_0.nch
					s_1.nch

Table 6.2: Kernel features for defining parser states in **no-lookahead** mode, which is used for utilizing the SLM.

to parse (the dependency parser has high confidence), the best-first strategy quickly finishes the sentence without spending too much time and hence, without generating too many parser history-states for the position. In contrast, if the parser has high entropy on the rest the sentence, the parser spends more time and generates more history-states for the position; Therefore, the set of history-states is affected by the rest of the sentence in the best-first strategy which is not desired for the SLM. To avoid this, we use parser with the regular beam-search strategy where the parser states are extended step by step, and at each step the states with same number of actions are processed.

6.4 Up-Training

While the use of simple transition based models results in fast decoding algorithms for parsing and POS tagging, it can lead to worse performance as opposed to more

complex models such as a constituent parser. Specially, when applied in the no-lookahead mode, the accuracy of the POS tagger and dependency parser significantly degrade[7]. Using more complex models with higher accuracy however is impractical because they are slow. Instead, we seek to improve the accuracy of our fast tools.

To achieve this goal we use up-training, in which a more complex model is used to improve the accuracy of a simpler model. We are given two models, M_1 and M_2, as well as a large collection of unlabeled text. Model M_1 is slow but more accurate while M_2 is fast but obtains lower accuracy. Up-training applies M_1 to tag a large amount of unlabeled data, which is then used as training data for M_2. Like self-training, a model is retrained on automatic output, but here the output comes form a more accurate model. Petrov et al. [99] used up-training as a domain adaptation technique: a constituent parser – which is more robust to domain changes – was used to label text from a new domain, and a fast dependency parser was trained on the automatically labeled data. We use a similar idea, where our goal is to recover the accuracy lost from using simpler models. Note that while up-training uses two models, it differs from co-training [101] since we care about improving only one of the models (M_2). Additionally, the models may vary in different ways. For example, they could be the same algorithm with different parameters settings or pruning methods, which can lead to faster but less accurate models. Figure 6.3 depicts the diagram for up-training framework where a model M_1 (a dependency parser) is used to label a large amount

[7]Our experiments in Section 6.7 confirms this degradation in performance for the no-lookahead models.

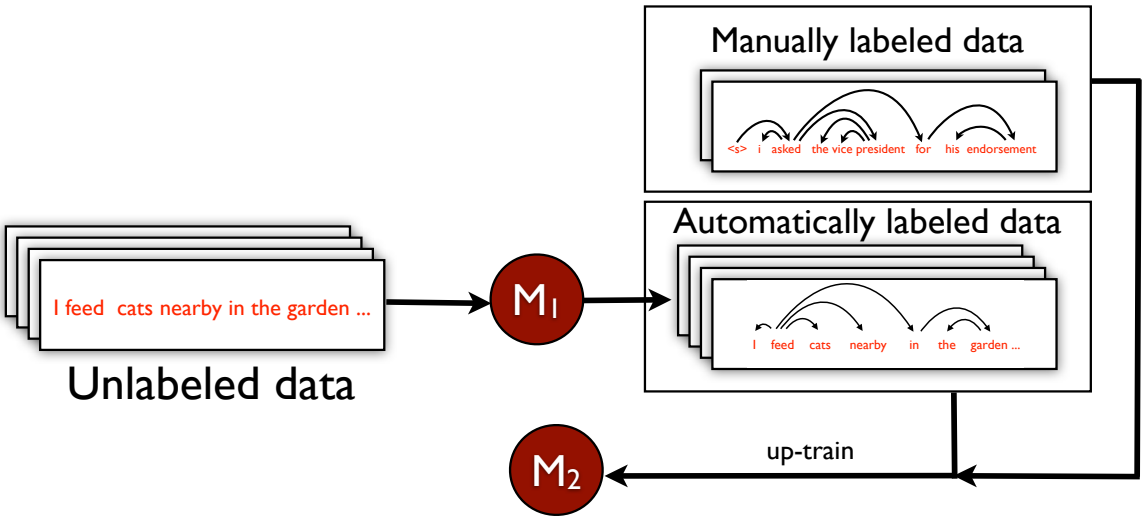


Figure 6.3: Up-training diagram for dependency parsing. Model M_1 is the accurate but impractical and M_2 is the model that we want to deploy for the task. The goal of up-training is to improve the performance of the deployed model M_2, using automatically labeled data labeled from M_1.

of text. The automatically parsed text along with the available manually labeled data is then passed to the training stage of M_2 to up-train it.

We apply up-training to improve the accuracy of both our fast part of speech tagger and dependency parser. For applying these tools in discriminative language model framework, where lookahead features are permitted, we parse a large corpus of text with a very accurate but very slow constituent parser and use the resulting data to up-train our tools in **lookahead** mode. That is, we up-train them with access to the full set of features. For the SLM framework the model M_2 is up-trained without any lookahead features, i.e. in the no-lookahead mode. Model M_1 is the same transition based model. However, the model M_1 still uses lookahead features to label the unannotated text on which M_2 is (up) trained. The idea is that while M_1 can not

be used in the SLM framework (due to left-to-right factorization of the LM), we still want M_1 to improve the accuracy of M_2 and ultimately, by providing as accurate as annotation as possible. We will demonstrate empirically that up-training improves these fast transition based models to yield better WER results in both scenarios: discriminative LM and the left-to-right SLM.

6.4.1 Up-training Dependency Parser with MaxEnt Classifier

The MaxEnt conditional model $p_g(\omega|\pi)$ used by the classifier for the dependency parser is trained to maximize the entropy subject to certain equality constraints on a training data,

$$\arg\max_{p_g} H(p_g) \;\text{ subject to } E_{\tilde{p}}[f_i] = E_{p_g}[f_i], \qquad 1 \leq i \leq |\mathbf{f}|, \tag{6.4}$$

where $E_{\tilde{p}}$ is the empirical expectation of the features on the training data and E_{p_g} corresponds to model expectation based on the trained conditional distribution. It can be shown that MaxEnt estimation above results in a $p_g(\omega|\pi)$with the following parametric form:

$$p_g(\omega|\pi) = \frac{1}{Z(\pi)} \exp\left(\sum_{i=1}^{|\mathbf{f}|} \lambda_i f_i(\omega, \pi)\right), \tag{6.5}$$

where λ_i is the weight corresponding to feature f_i. It is also well known that the parameters (λ_is) of the log-linear model above may equivalently be estimated to maximize the likelihood of the training data: $LL(\Lambda) = \sum_{\omega,\pi \in \mathcal{T}} \log(p_g(\omega|\pi))$.

To be able to apply up-training to a huge amount of automatically labeled data (using model M_1), we use the framework of Kazama and Tsujii [102] where the equality constraints in Eqn. 6.4 are replaced with box-type inequality constraints, i.e. the equality can be violated to reflect unreliability in the expectations of the features on a training data. The new MaxEnt objective function – which is referred to as inequality MaxEnt in [102] – is

$$\arg\max_{p_g} H(p_g) \;\text{ subject to }\; -U \leq E_{\tilde{p}}[f_i] - E_{p_g}[f_i] \leq U, \quad 1 \leq i \leq |\mathbf{f}|, \tag{6.6}$$

where U is a parameter that controls the amount of uncertainly allowed in the estimated expectations. Kazama and Tsujii [102] shows that the objective function above is equivalent to penalized maximum likelihood estimation of the model of (6.5), with L1 regularization, which results in sparse solutions $\{\lambda_i\}$, i.e. the inequality MaxEnt embeds feature selection in its estimation. More specifically, the dual log-likelihood objective function becomes:

$$\arg\max_{\lambda_i \geq 0} LL(\Lambda) - U \sum_i \lambda_i \tag{6.7}$$

where $LL(\Lambda)$ is the log-likelihood of the training data. Also, since $\lambda_i \geq 0$, the second

term in the equation above is the L1-norm of the parameters and hence, the solution is forced to be sparse (U controls the degree of sparsity). The sparse solution is specifically an important advantage as it utilizes up-training of the parser on a huge amount of automatically labeled data while still keeping the model size reasonable. Additionally, it is discussed and shown experimentally in [102] that the sparse solution also improves the robustness (and generalizability) of the MaxEnt model.

Following Kazama and Tsujii [102], we use the BLMVM algorithm [103] – which is a variant of the limited-memory variable metric (LMVM) algorithm with support for bound constraints – to optimize the objective function in (6.7).

6.5 Related Work

The idea of efficiently processing a set of similar sentences is similar to "lattice-parsing", in which a parser considers an entire lattice at once [104, 105]. These methods typically constrain the parsing space using heuristics, which are often model specific.Typically, they search for the best sentence-parse pair in the joint space of word sequences present in the lattice and their syntactic analyses; they are therefore not guaranteed to produce a syntactic analysis for each hypothesis. In contrast, substructure sharing is a general purpose method that we have applied to two different tasks: POS tagging and parsing. The output of the proposed algorithm is identical to processing each sentence separately and output is generated for each sentence.

Hall [104] uses a lattice parsing strategy which aims to compute the marginal probabilities of all word sequences in the lattice by summing over syntactic analyses of each word sequence. The parser sums over multiple parses of a word sequence implicitly. The lattice parser therefore, is itself a language model. In contrast, our tools are independent of the underlying syntactic modeling framework, which allows the LM to utilize whatever features, from the analysis, that are most beneficial. This independence means our tools are useful across different tasks such as machine translation. In fact, we showed that the substructure sharing can be used in both discriminative training of a syntactic LM and in the SLM framework. These differences make substructure sharing a more attractive option for efficient algorithms.

While Huang and Sagae [51] use the notion of "equivalent states", they do so for dynamic programming in a shift-reduce parser to broaden the search space for a given sentence. In contrast, we use the idea to identify substructures across multiple sentences, where our goal is to obtain efficiencies in the cumulative parsing task. Additionally, we extend the definition of equivalent states beyond parsing to general transition based structured prediction models, and demonstrate applications beyond parsing.

Finally, while there is significant work on syntactic feature engineering for language modeling, our goals of speed are orthogonal. We believe that our improved algorithms may be useful to others by others exploring new syntactic LM features.

Usage	Data	Size
Supervised training: dependency parser, POS tagger	Ontonotes BN treebank+ WSJ Penn treebank	1.3M words, 59K sent.
Up-training: dependency parser, POS tagger	TDT4 closed captions+EARS BN03 closed caption	195M words available
Evaluation: up-training	BN treebank test (following Huang et al. [106])	20k words, 1.1k sent.
Evaluation: ASR transcription	rt04 BN evaluation	4 hrs, 45k words

Table 6.3: A summary of the data for up-training the POS tagger and the dependency parser in discriminative language model experiments. The Ontonotes corpus is from [1].

6.6 Discriminative Language Model Experiments

The setup used for discriminative language model experiments is same as the setup described in Section 5.6. We use the same syntactic features for training the discriminative global linear language model. As in Section 5.6, we use a modified Kneser-Ney (KN) backoff 4-gram baseline LM. Word-lattices for discriminative training and rescoring come from this baseline ASR system.[8] The long-span discriminative LM's baseline feature weight (α_0) is tuned on dev data and hill climbing is used for training and rescoring. The parser and POS tagger are trained on BN treebank data from Ontonotes [1] and the WSJ Penn Treebank, a total of $1.3M$ words and $59K$ sentences. Phrase structures are converted to labeled dependencies using the Stanford converter. To create labeled data for up-training we use the CKY-style bottom-up constituent parser of Huang et al. [106], a state of the art BN parser, trained on the

[8]For training a 3-gram LM is used to increase confusions.

same training data as the parser and POS tagger, followed by self-training on the Hub4 CSR 1996 language model text. While accurate, the parser has a large grammar (32GB) due to the use of products-of-latent-grammars and requires $O(l^3)$ time to parse a sentence of length l. Therefore, this parser is not practical for ASR rescoring since utterances can be very long, although the (shorter) up-training text data was not a problem. Table 6.3 summarizes the data used for performing up-training experiments. To utilize up-training for the dependency parser on a huge amount of automatically labeled data, we use the inequality MaxEnt estimation described in Section 6.4.1. This essentially enables us to fully take advantage of up-training while keeping the underlying model size reasonably small. To measure the effect of the up-training on the performance of the POS tagger and dependency parser, we use part of BN treebank data, about $20k$ words and $1.1k$ sentences (the same setup as Huang et al. [106].). We evaluate both unlabeled (UAS) and labeled dependency accuracy (LAS) for the dependency parser.

6.6.1 Results

Before we demonstrate the speed of our models, we show that up-training can produce accurate and fast models. We first, apply up-training to the POS tagger where we use the POS tags produced by the constituent parser as the labeled data for up-training. The POS tags produced by the constituent parser have a tagging accuracy of 97.2% on up-training evaluation data of BN treebank (Table 6.3). Up-

training improves the POS tagger's accuracy from 95.9% to 97%, when about 2.3M words of the unannotated data are used (we experimented with more data but could not see more improvements in the accuracy of the POS tagger) . The up-trained POS tagger is used for up-training the dependency parser as it has a better performance. In other words, we first up-train the POS tagger and then use it to tag the up-training data of the dependency parser. Figure 6.4 shows improvements to parser accuracy

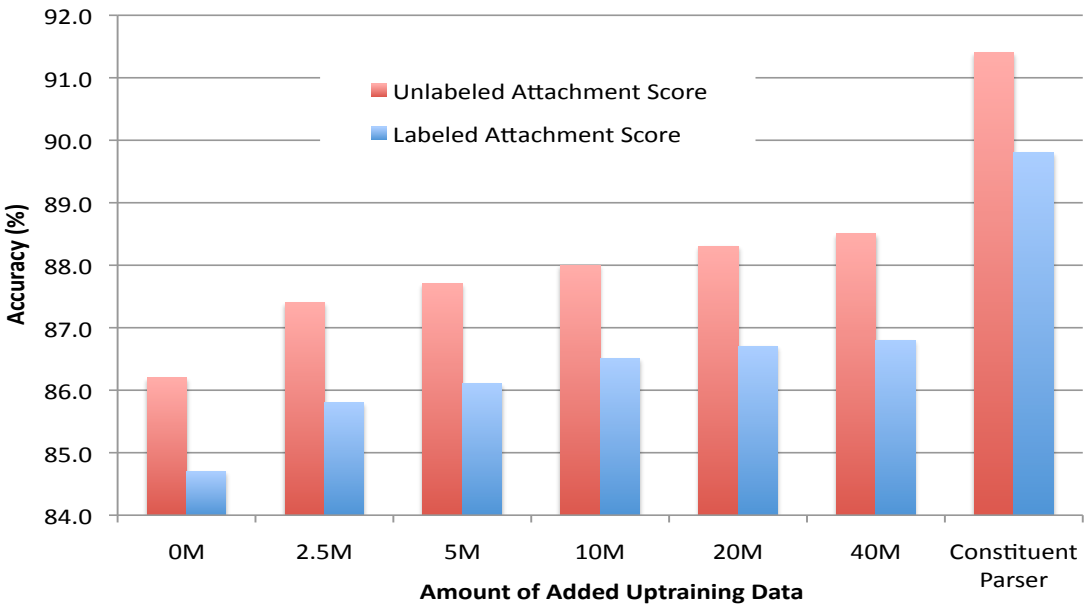


Figure 6.4: Up-training results for dependency parsing for varying amounts of automatically labeled data (number of words.) The first column is the dependency parser with supervised training only and the last column is the constituent parser (after converting to dependency trees.)

through up-training for different amount of (randomly selected) data, where each column corresponds to the amount of automatically labeled data that is added to manually labeled Treebank data (Table 6.3). The last column indicates the score

(91.4% UAS) of the constituent parser that was used as the model M_1 for up-training. We use the (up-trained) POS tagger to generate tags for dependency training to match the test setting. While there is a large difference[9] between the constituent and dependency parser without up-training (91.4% vs. 86.2% UAS), up-training can cut the difference by 44% to 88.5%, and improvements saturate around 40M words (about 2M sentences). Figure 6.5 shows the number of features included in each of the up-

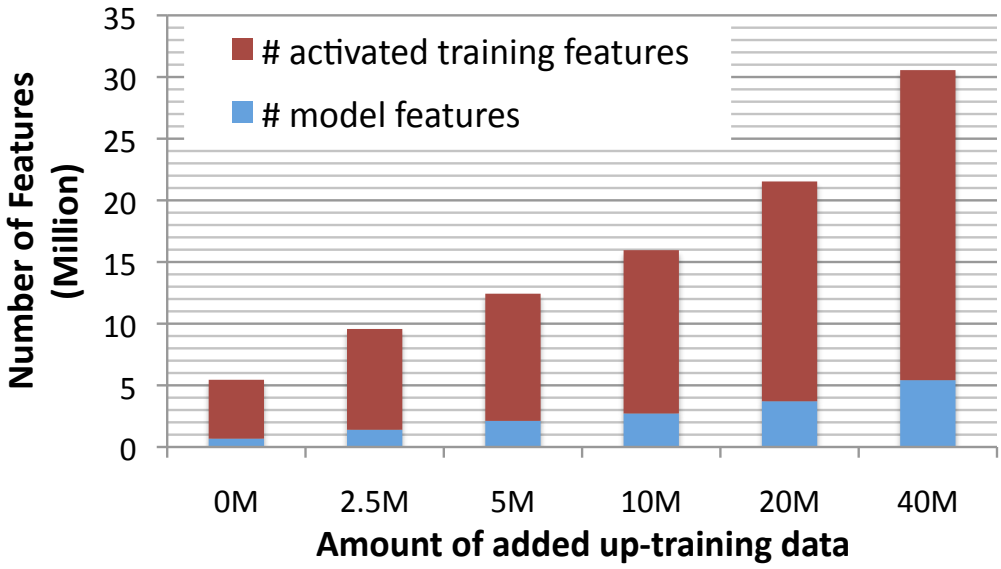


Figure 6.5: Number of features included in the MaxEnt classifier of the dependency parser trained with inequality constraints compared to the maximum number of features activated on the up-training data.

trained models. The inequality constraint parameter U is tuned on the development set of BN Treebank data to achieve the best parser accuracy, for each up-training data size. In this figure the red bars corresponds to the total number of features activated on the whole up-training data (the left bar corresponds to the number

[9]Better performance is due to the exact CKY-style – compared with best-first and beam– search and that the constituent parser uses the product of huge self-trained grammars.

of features for the supervised training only) and the blue bars show the portion of those features that are included in the final trained model. It can be seen that the inequality constraints applied to the MaxEnt classifier of the parser drastically reduce the number of features, and results in sparse models. As an example up-training with 40M words of automatically labeled data results in a model that has about 5.4M features, compared to 30.5M the full feature set size on that data, which is a 82% reduction in the number of features. As a result of inequality MaxEnt training, the dependency parser remains much smaller than the constituent parser— in addition to being much faster; the up-trained dependency model only uses 700MB of memory and has about 6M features compared with 32GB for constituency model.

6.6.2 Discriminative LM with Fast Parser

Here, we present the speedup gains using substructure sharing for the discriminative LM of Chapter 5, Section 5.7. We train the syntactic discriminative LM, with head-word and POS tag features (Section 5.7), using the faster parser and tagger and then rescore the lattices of the rt04 evaluation set. Table 6.4 shows the decoding speedups as well as the WER reductions compared to the baseline LM. Note that up-training improvements lead to WER reduction using the discriminative language model. In fact, the experiments carried out in Section 5.7 were using the already up-trained tools (POS tagger and dependency parser) and the WERs reported in Table 5.1 were with the improved models. Detailed speedups on substructure shar-

ing are shown in Table 6.5; the POS tagger achieves a 5.3 times speedup, and the parser a 5.7 speedup without changing the output. We also observed speedups during training, where the processing time spent by the POS tagger reduces from 3.2 hours to 0.54 and the dependency parsing time reduces from 64.3 to 12.9 (the processing times without substructure sharing were reported in Table 5.2).

The results above are for the already fast hill climbing decoding, but substructure sharing can also be used for N-best list rescoring. Figure 6.6 (logarithmic scale) illustrates the time for the parser and tagger to process N-best lists of varying size, with more substantial speedups for larger lists. For example, for N=100 parsing time reduces from about 20,000 seconds to 2,700 seconds, about 7.4 times as fast.

LM	WER	Substr. Share (sec) No	Yes
Baseline 4-gram	15.1	-	-
Syntactic LM	14.8	8,658	**1,648**
+ up-train	**14.6**		

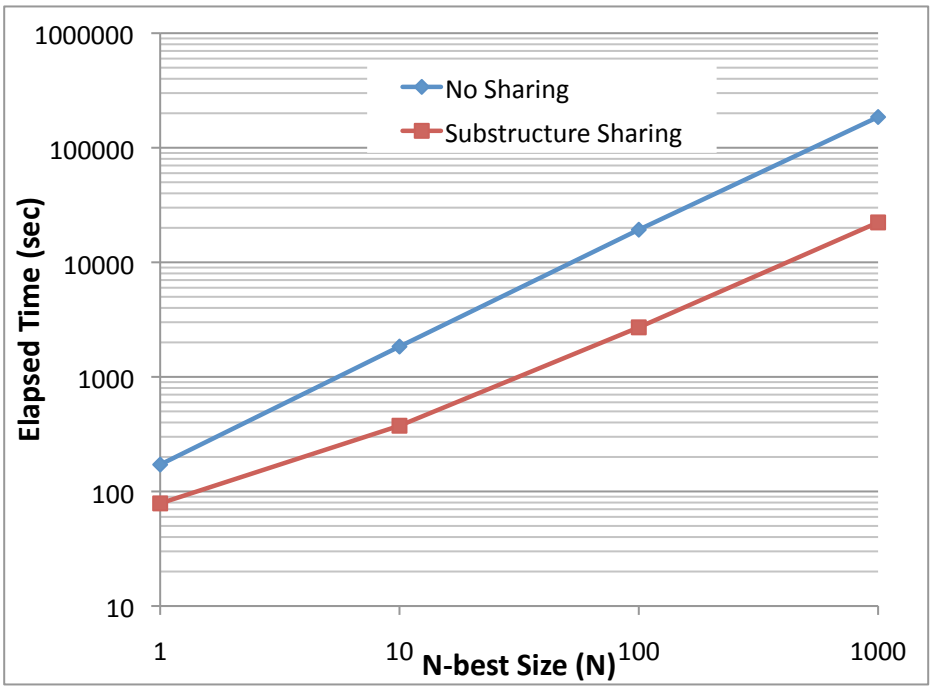
Table 6.4: Speedups and WER for hill climbing rescoring with syntactic discriminative language model. Substructure sharing yields a 5.3 times speedup. The times for with and without up-training are nearly identical, so we include only one set for clarity. Time spent is dominated by the parser, so the faster parser accounts for much of the overall speedup. Timing information includes neighborhood generation and LM rescoring, so it is more than the sum of the times in Table 6.5.

	Substr. Share No	Yes	Speedup
Parser	8,237.2	**1,439.5**	5.7
POS tagger	213.3	**40.1**	5.3

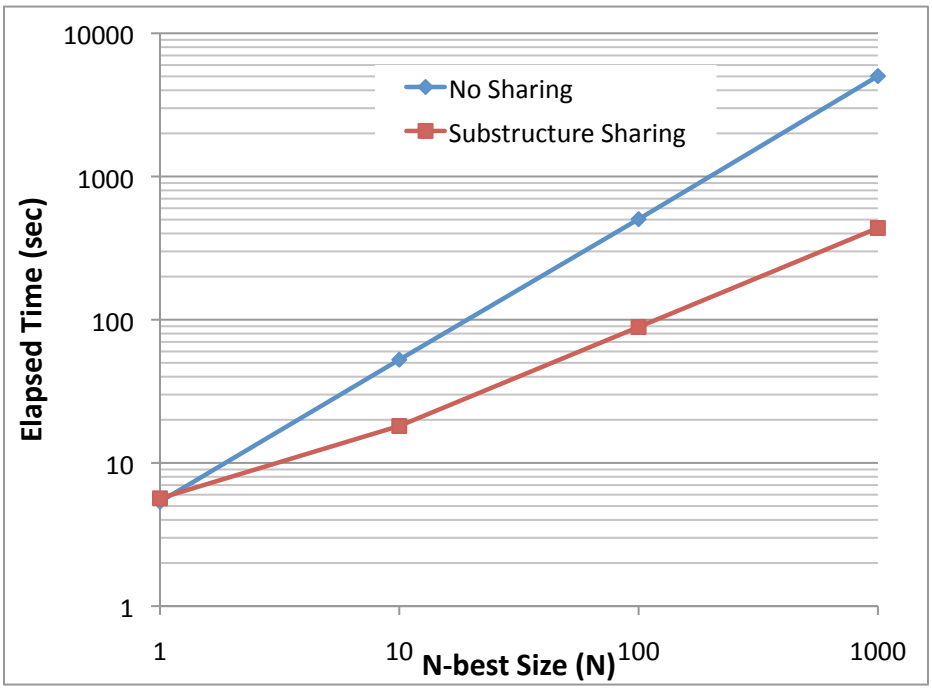
Table 6.5: Time in seconds for the parser and POS tagger in lookahead mode to process hypotheses during hill climbing rescoring with syntactic discriminative language model.

6.7 SLM Experiments

In this section, we demonstrate the impact of the proposed substructure sharing and up-training algorithm on the application of the SLM. To this end, we use the $\texttt{HW+HT}_2$ SLM developed in Chapter 3 and apply it to the speech recognition experiments of Section 4.7 for `BN` setup. We first measure the speedup gains using the substructure sharing. Recall that both the POS tagger and the dependency parser are used in the no-lookahead mode for the SLM (Section 6.3.3). Also, as discussed in Section 6.3.3, the POS tagger in the no-lookahead mode only uses features based on the two previous words in the history and hence, it can be directly applied to the lattice to POS tag each arc by traversing the arcs of the lattice, without the need



(a)



(b)

Figure 6.6: Elapsed time for (a) parsing and (b) POS tagging the N-best lists in lookahead mode with and without substructure sharing.

for state expansion. Therefore, we do not measure the speedup due to substructure sharing utilized for the POS tagger and only report substructure sharing results for the dependency parser.

Table 6.6 presents the speedup gains for hill climbing rescoring with different number of initial paths using the SLM[10], where we apply the hill climbing rescoring with and without substructure sharing. As observed, the substructure sharing again substantially reduces the processing time spent by the dependency parser. Additionally, as the number of examined hypotheses increases – by increasing the number of initial paths– the amount of duplications present in the hypotheses set increases. Subsequently, the substructure sharing becomes more useful.[11]. To confirm this, we measure the percentage of the parser states which are cached using the proposed substructure sharing during the hill climbing rescoring with different number of initial paths . Table 6.7 shows the average (per utterance) number of encountered parser states during the hill climbing. The number of cached parser states indicates the number of times we process a parser state (out of total number of encountered states) for which there is an already cached equivalent state. As seen, increasing the number of initial paths increases the percentage of cached states, which confirms the increase in the amount of substructure duplications.

Next, we perform up-training and demonstrate the impact of up-training on the

[10]The WER results using the SLM can be found in Table 4.2. Here, we only report the speedup gains of substructure sharing.

[11]The same phenomenon can be observed in Figure 6.6 where the POS tagger and the dependency parser are used in the lookahead mode to process N-best list with substructure sharing. As N increases (number of processed hypotheses), the speedup gain becomes more substantial.

	Substr. Share (sec)		Speedup
	No	Yes	
1 initial path	14642	**3184**	4.6
10 initial path	89720	**18300**	4.9
20 initial path	153083	**28697**	5.3

Table 6.6: Time in seconds for the parser (in no-lookahead mode) to process hypotheses during hill climbing rescoring using the $\texttt{HW+HT}_2$ SLM with and without substructure sharing.

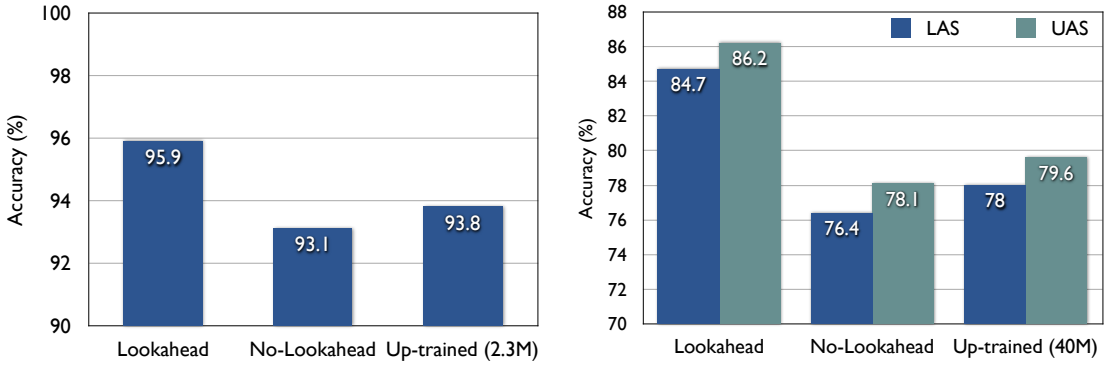
	# Parser States	# Cached Parser States	Percentage Cached (%)
1 initial path	56505.4	**54764.9**	96.92
10 initial path	347827.0	**340119.3**	97.78
20 initial path	610822.1	**604312.0**	98.93

Table 6.7: Averaged (per utterance) number of cached parser states v/s averaged number of processed parser states during hill climbing rescoring using the $\texttt{HW+HT}_2$ SLM with substructure sharing.

performance of the POS tagger and the dependency parser in the no-lookahead mode. To apply up-training in no-lookahead mode for SLM experiments, we use the model (the POS tagger or the dependency parser) with lookahead features as M_1 and label a huge amount of unlabeled data to up-train the model with no-lookahead features, i.e. M_2. In other words, although M_1 can not be used in the SLM experiments, we use it to (up) train M_2 and improve its performance. The data sets used for performing the up-training are shown in Table 6.3.

Figure 6.7 illustrates the accuracy of the lookahead models, no-lookahead models, and the up-trained no-lookahead models for the POS tagger and the dependency parser. As seen in Figures 6.7(a) and 6.7(b), excluding the lookahead features significantly degrades the accuracy of the POS tagger and the dependency parser, respectively. Additionally, it can be seen that the up-training significantly improves the

performance of the models. Specifically, applying up-training on $40M$ automatically labeled data (labeled by the lookahead parser) improves the UAS score from 78.1% to 79.6% and cut the difference between the accuracy of the lookahead parser and the no-lookahead parser by about 19%. Using the improved up-trained POS tagger and dependency parser, we re-train the $\texttt{HW+HT}_2$ SLM for the `BN` setup described in Section 3.3.2. The only difference here w.r.t the experiments of Section 3.4 is that we use the up-trained tools (POS tagger and dependency parser) to process the SLM training text data. Table 6.8 summarizes the perplexity results using the up-trained models for `BN` experiments. It can be observed that improving the accuracy of the POS tagger and the dependency parser by up-training has a direct impact on the performance of the trained SLM. Despite being modest, we want to emphasize that the improvement due to deploying up-trained models is obtained without increasing the complexity of the models and therefore, is practical.



(a) (b)

Figure 6.7: Effect of no-lookahead mode with and without up-training on the accuracy of the (a) POS tagger and (b) dependency parser. The up-training is performed using the models with full set of features (including lookahead features).

LM	Eval	Dev
SLM HW+HT$_2$	159	168
Up-trained SLM HW+HT$_2$	152	162
Interp. HW+HT$_2$	142	147
Up-trained Interp. SLM HW+HT$_2$	**139**	**143**

Table 6.8: The perplexity performance of the HW+HT$_2$ SLM using the up-trained POS tagger and dependency parser for BN setup.

6.8 Conclusion

The computational complexity of accurate syntactic processing can make structured language models impractical for applications such as ASR that require scoring hundreds of hypotheses per input. We have presented substructure sharing, a general framework that greatly improves the speed of syntactic tools that process candidate hypotheses. Furthermore, we achieve improved performance through up-training. The result is a large speedup in rescoring time, even on top of the already fast hill climbing framework, and reductions in WER from up-training. Our results make long-span syntactic LMs practical for real-time ASR, and can potentially impact machine translation decoding as well. Additionally, while not a goal of this dissertation, improved training (in case of discriminative training) and testing time means we can effectively explore new LM features.

Chapter 7

Discussion and Summary of Contributions

This dissertation has focused on practicality concerns and efficiency of syntactic language models for automatic speech recognition (ASR). Approaches for efficient training of language models that incorporate long-span features, e.g. syntactic features, are presented (Chapters 3 and 5). A new computationally efficient rescoring framework based on the well-known hill climbing algorithm is proposed to integrate complex long-span models into the ASR task (Chapter 4). The proposed hill climbing algorithm effectively mitigates search errors when these models are utilized to rescore the prohibitively huge hypothesis space of ASR systems. The proposed scheme enables the realization of the superior power of the applied syntactic language models—and could similarly benefit other linguistically motivated complex language

models—over simple n-gram models, and alleviates the dependency of such models on inefficient N-best rescoring techniques.

Even with our fast hill climbing algorithm, the reliance of syntactic models on auxiliary tools such as part of speech taggers and dependency parsers to *separately* process each hypothesis during rescoring can slow the application of these models. The key to avoiding this inefficiency is the fact that many of these hypotheses, in N-best or hill climbing rescoring, differ only slightly and there exists significant commonalities among these hypotheses[1]. We propose a general modification to the decoders used in the auxiliary tools to utilize these commonalities among the processed hypotheses (Chapter 6).

The followings are the key technical contributions of this dissertation:

- In Chapter 2, An empirical study is performed to show why n-gram models are weak word-predictors for positions where syntactic relations spans beyond the recent words in the history and are not, therefore, captured by n-gram models. In particular, It is shown that there is a substantial difference in the predictive power of n-grams LMs at positions within a sentence where syntactic dependencies reach further back than n-gram context versus positions where syntactic dependencies are local. Furthermore, it is demonstrated that the mentioned difference in the performance is not diminished by simply increasing the training data or even n-

[1]After all, they all represent likely hypotheses for a given acoustic input and given the current accuracy of state-of-the-art ASR systems on many domains, it is not hard to observe that the variations among likely hypotheses is small.

gram order, suggesting the need for explicit syntactic information in language modeling (Section 2.3).

- Chapter 3 presents a structured language modeling scheme based on the dependency structures using a state-of-the-art shift-reduce probabilistic dependency parser (Section 3.3). Although closely related to the original SLM of Chelba and Jelinek [5], two key improvements are proposed. First, A simple hierarchical interpolation of syntactic parameters is proposed, which achieves better performance without significant model complexity (Section 3.4). Second, a general information-theoretic pruning scheme to reduce the size of the underlying Jelinek-Mercer LM used by the SLM framework is demonstrated, enabling efficient estimation on a large training corpus while keeping the number of parameters small (Section 3.5). The proposed Jelinek-Mercer LM pruning algorithm is general and can be used for efficiently reducing the size of a Jelinek-Mercer LM in other situations. We also analyze the influence of long-distance syntactic dependencies in the SLM (Section 3.6) and empirically demonstrate that the captured long-distance dependencies are mostly beneficial at positions where local n-gram context is least effective.

- Our structured language model, like other long-span language models which do not make the Markov assumption, can not be used in first pass decoding due to its computational complexity. Additionally, these complex models often can not be utilize to directly rescore first pass speech lattices, as they require an

extensive expansion of the equivalent history states present in lattices. Instead, these models are exploited in an N-best rescoring framework, which suffers from several known deficiencies and inefficiencies as a search algorithm. Chapter 4 presents our proposed hill climbing rescoring algorithm, where we use the finite-state transducer machinery to efficiently search for the optimal solution in space of ASR hypotheses using these long-span models. Unlike previously investigated hill climbing algorithms for ASR, which work on confusion networks, our proposed hill climbing algorithm (Section 4.3.3) works directly on word lattices and evaluates explicit paths in the original lattice, permitting greater flexibility while avoiding the need to construct the confusion in a post processing step.

Our extensive experiments using different models (Section 4.5) show that the proposed hill climbing rescoring algorithm is an efficient and effective search strategy which quickly reaches good solutions in the space with significantly fewer hypotheses evaluation than the conventional N-best rescoring. Our proposed SLM when deployed using the efficient hill climbing framework in (Section 4.7) significantly improves the baseline state-of-the-art 4-gram model.

- The proposed finite-state transducer based hill climbing algorithm on speech lattices opens a new avenue for efficient training of discriminative language models. Discriminative language models are trained on acoustic sequences with their transcriptions, in an attempt to directly optimize the desired objective function, i.e. word error-rate (WER) and therefore, have the potential to resolve task-specific

confusions. The perceptron training of global-linear discriminative language models—which provides the flexibility to include both typical n-gram features and arbitrary features based on the global properties of word sequence (long-span features)— is considered in Chapter 5. Unlike generative language models with long-span dependencies where one has to resort to N-best lists only during decoding, discriminative models with sentence-level features force the use of N-best lists even for *training*. The idea of hill climbing is combined with discriminative training to propose a discriminative hill climbing algorithm for efficient training of such long-span language models (Section 5.4). Our experiments (Section 5.6) show that the new discriminative training algorithm is more efficient and alleviates the shortcoming of N-best training by directly using acoustic confusions captured in word lattices.

- Extracting syntactic dependencies and features in both SLM and discriminative language modeling frameworks relies on the use of auxiliary tools such as POS taggers and dependency parsers. Applying these tools separately to each hypothesis in the search space—during rescoring or discriminative training, N-best or hill climbing— is inefficient as these hypotheses often differ only slightly. In the process of exploiting the commonalities among the hypotheses of the search space, we have found a general principle to share substructure states in a transition based structured prediction algorithm deployed by the auxiliary tool, i.e. algorithms where final structures are composed of a sequence of multiple individual deci-

sions. Section 6.3 demonstrates our proposed algorithm to reduce the complexity of processing a hypothesis set by sharing common substructures among the hypotheses. It is shown that the proposed algorithm substantially improves the speed of the dependency parser when utilized in the hill climbing algorithm. Unlike some superficially similar lattice parsing algorithms, our approach is general and produces exact output for all the hypotheses. Additionally, it is separated from the ASR system and the deployed syntactic modeling framework, which means that our algorithm maybe used in other NLP applications that use transition based models.

The cumulative effect of the contributions is expected to make the use of long span language models practical and more widespread, leading to improved ASR systems in deployed application. The techniques are also extensible to other tasks that require language models, such as SMT, and other scenarios where multiple similar inputs must be processed by transition based tool for natural language processing.

Bibliography

[1] R. Weischedel, S. Pradhan, L. Ramshaw, M. Palmer, N. Xue, M. Marcus, A. Taylor, C. Greenberg, E. Hovy, R. Belvin, and A. Houston, *OntoNotes Release 2.0*, Linguistic Data Consortium, Philadelphia, 2008.

[2] M. Collins, B. Roark, and M. Saraclar, "Discriminative syntactic language modeling for speech recognition," in *ACL*, 2005.

[3] N. Chomsky, "Three models for the description of language," *IRE Transactions on Information Theory*, pp. 113–124, 1956.

[4] E. Brill, R. Florian, J. Henderson, and L. Mangu, "Beyond N-grams: Can linguistic sophistication improve language modeling?" in *Proceedings of COLING/ACL*, 1998, pp. 186–190.

[5] C. Chelba and F. Jelinek, "Structured language modeling," *Computer Speech and Language*, vol. 14, no. 4, pp. 283–332, 2000.

[6] S. Khudanpur and J. Wu, "Maximum entropy techniques for exploiting syntac-

tic, semantic and collocational dependencies in language modeling," *Computer Speech and Language*, pp. 355–372, 2000.

[7] E. Charniak, "Immediate-head parsing for language models," in *Proceedings of ACL*, 2001.

[8] A. Emami and F. Jelinek, "A Neural Syntactic Language Model," *Machine learning*, vol. 60, pp. 195–227, 2005.

[9] H. K. J. Kuo, L. Mangu, A. Emami, I. Zitouni, and L. Young-Suk, "Syntactic features for Arabic speech recognition," in *Proc. ASRU*, 2009.

[10] D. Filimonov and M. Harper, "A joint language model with fine-grain syntactic tags," in *EMNLP*, 2009.

[11] F. Jelinek, *Statistical Methods for Speech Recognition*. The MIT Press, 1998.

[12] S. Young, "Acoustic modelling for large vocabulary continuous speech recognition," *Computational Models of Speech Pattern Processing*, pp. 18–39, 1999.

[13] M. Gales and S. Young, "The Application of Hidden Markov Models in Speech Recognition," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, 2007.

[14] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *Annals of Mathematical Statistics*, vol. 37, pp. 1559–1563, 1966.

[15] R. Rosenfeld, S. F. Chen, and X. Zhu, "Whole-Sentence Exponential Language models : A Vehicle for Linguistic-Statistical Integration," *Computer Speech and Language*, vol. 15, no. 1, 2001.

[16] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker, "Perplexity—a measure of the difficulty of speech recognition tasks," *Journal of the Acoustic Society of America*, 1977.

[17] S. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," Computer Science Group, Harvard University, Tech. Rep., 1998.

[18] R. Iyer, M. Ostendorf, and M. Meteer, "Analyzing and predicting language model improvements," *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 254–261, 1997.

[19] S. F. Chen, D. Beeferman, and R. Rosenfeld, "Evaluation metrics for language models," in *Proc. of the DARPA Broadcast News Transcription and Understanding Workshop*, 1998, pp. 275–280.

[20] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, 1987.

[21] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling,"

in *Proceedings of the IEEE International Conference on Acoustic, Speech and Signal Processing*, 1995.

[22] F. Jelinek and R. L. Mercer, "Interpolated estimation of Markov source parameters from sparse data," in *Proceedings of the Workshop on Pattern Recognition in Practice*, 1980, pp. 381–397.

[23] I. J. Good, "The population frequencies of species and the estimation of population parameters," *Biometrika*, vol. 40, pp. 237–264, 1953.

[24] H. Ney, U. Essen, and R. Kneser, "On structuring probabilistic dependences in stochastic language modeling," *Computer Speech and Language*, vol. 8, pp. 1–38, 1994.

[25] R. Rosenfeld, "Two decades of statistical language modeling: Where do we go from here?" *Proceedings of the IEEE*, 2000.

[26] J. Goodman, "A bit of progress in language modeling - Extended Version," Microsoft Research Technical Report, Tech. Rep. MSR-TR-2001-72, 2001.

[27] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modelling," *Computer Speech and Language*, vol. 10, no. 3, pp. 187–228, 1996.

[28] T. Alumäe and M. Kurimo, "Efficient Estimation of Maximum Entropy Language Models with N -gram features : an SRILM extension," in *Interspeech*, 2010.

[29] J. Wu, "Maximum Entropy Language Modeling with Non-Local Dependencies," Ph.D. dissertation, Johns Hopkins University, 2002.

[30] C. Chelba and A. Acero, "Adaptation of maximum entropy capitalizer: Little data can help a lot," *Computer Speech and Language*, 2006.

[31] A. Rastrow, M. Dredze, and S. Khudanpur, "Adapting n-gram Maximum Entropy Language Models with Conditional Entropy Regularization," in *ASRU*, 2011.

[32] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech and Language*, vol. 13, no. 4, pp. 359–393, 1999.

[33] S. Della Pietra, V. Della Pietra, and J. Lafferty, "Inducing features of random fields," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 380–393, 1997.

[34] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization methods," *Mathematical Programming*, vol. 45, pp. 503–528, 1989.

[35] H.-K. J. Kuo, E. Fosler-Lussier, H. Jiang, and C.-H. Lee, "Discriminative training of language models for speech recognition," in *ICASSP*, 2002.

[36] A. Rastrow, A. Sethy, and B. Ramabhadran, "Constrained Discriminative Training of N-gram Language Models," in *ASRU*, 2009.

[37] B. Roark, M. Saraclar, and M. Collins, "Discriminative n-gram language modeling," *Computer Speech & Language*, vol. 21(2), 2007.

[38] S. Chen, B. Kingsbury, L. Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at IBM under the DARPA EARS program," *IEEE Transactions on Audio, Speech and Language Processing*, pp. 1596–1608, 2006.

[39] Z. Huang and M. Harper, "Self-Training PCFG grammars with latent annotations across languages," in *EMNLP*, 2009.

[40] M. Marcus, M. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, p. 330, 1993.

[41] J. Garofolo, J. Fiscus, W. Fisher, and D. Pallett, *CSR-IV HUB4*, Linguistic Data Consortium, Philadelphia, 1996.

[42] J. Wu and S. Khudanpur, "Combining Nonlocal, Syntactic and N-Gram Dependencies in Language Modeling," in *Proceedings of Eurospeech*, 1999, pp. 2179–2182.

[43] K. Sagae and J. Tsujii, "Dependency parsing and domain adaptation with LR

models and parser ensembles," in *Proc. EMNLP-CoNLL*, vol. 7, 2007, pp. 1044–1050.

[44] P. Xu, C. Chelba, and F. Jelinek, "A study on richer syntactic dependencies for structured language modeling," in *Proceedings of ACL*, 2002.

[45] A. Stolcke, "Entropy-based pruning of backoff language models," in *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[46] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society*, vol. 39, no. B, pp. 1–38, 1977.

[47] D. G. Hays, "Dependency theory: A formalism and some observations," *Language*, vol. 40, no. 4, pp. 511–525, 1964.

[48] M.-C. de Marneffe, B. Maccartney, and C. D. Manning, "Generating typed dependency parses from phrase structure parses," in *Proceedings of LREC*, 2006.

[49] H. Yamada and Y. Matsumoto, "Statistical dependency analysis with support vector machines," in *Proceedings of Internationl Workshop on Parsing Technologies (IWPT)*, 2003, pp. 195–206.

[50] J. Nivre, J. Hall, J. Nilsson, G. Eryigit, and S. Marinov, "Labeled pseudo-projective dependency parsing with support vector machines," in *Proceedings of CoNLL*, 2006, pp. 221–225.

[51] L. Huang and K. Sagae, "Dynamic Programming for Linear-Time Incremental Parsing," in *Proceedings of ACL*, 2010.

[52] C. Cortes and V. Vapnik, "Support vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[53] S. Buchholz and E. Marsi, "CoNLL-X shared task on multilingual dependency parsing," in *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, 2006.

[54] K. Sagae and A. Lavie, "A best-first probabilistic shift-reduce parser," in *Proc. ACL*. Association for Computational Linguistics, 2006, pp. 691–698.

[55] S. Kübler, R. McDonald, and J. Nivre, "Dependency parsing," *Synthesis Lectures on Human Language Technologies*, vol. 2, no. 1, pp. 1–127, 2009.

[56] L. E. Baum, "An equality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes," *Inequalities*, vol. 3, pp. 1–8, 1972.

[57] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text compression*. Englewood CLiffs, N.J.: Prentice Hall, 1990.

[58] L. Bahl, "A maximum likelihood approach to continuous speech recognition," *IEEE Transactions on Pattern Analysis and Machine Inteligence (PAMI)*, vol. 5, no. 2, pp. 179–190, 1983.

[59] D. Filimonov and M. Harper, "A joint language model with fine-grain syntactic tags," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

[60] Y. Tsuruoka, Y. Miyao, and J. Kazama, "Learning with Lookahead : Can History-Based Models Rival Globally Optimized Models ?" in *Proc. CoNLL*, no. June, 2011, pp. 238–246.

[61] I. Zitouni, "Backoff hierarchical class n-gram language models: effectiveness to model unseen events in speech recognition," *Computer Speech & Language*, vol. 21, no. 1, pp. 88–104, 2007.

[62] T. M. Cover, J. A. Thomas, and J. Wiley, *Elements of information theory*. New York: John Wiley and Sons, Inc, 1991.

[63] R. Rosenfeld, "A whole sentence maximum entropy language model," in *Proceedings of IEEE workshop on Automatic Speech Recognition and Understanding (ASRU)*, 1997.

[64] T. Mikolov, M. Karafiát, L. Burget, J. H. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of of International Speech Communication Association (INTERSPEECH)*, 2010.

[65] W. Wang and M. Harper, "The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources," in *Proceed-*

ings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2002.

[66] A. Deoras and F. Jelinek, "Iterative decoding: A novel re-scoring framework for confusion networks," in *ASRU*, 2009.

[67] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus among words: Lattice-based word error minimization," in *Sixth European Conference on Speech Communication and Technology*, 1999.

[68] P. Beyerlein, "Discriminative Model Combination," *Proc. ICASSP*, 1998.

[69] L. Burget and et al., "Combination of strongly and weakly constrained recognizers for reliable detection of OOVs," in *Proc. ICASSP*, 2008.

[70] S. Furui, M. Nakamura, T. Ichiba, and K. Iwano, "Why is the recognition of spontaneous speech so hard?" in *International Conference on Text, Speech, and Dialogue*, 2005, pp. 9–22.

[71] A. Rastrow, A. Sethy, and B. Ramabhadran, "A new method for OOV detection using hybrid word/fragment system," in *Proceedings of ICASSP*, 2009.

[72] C. Parada, M. Dredze, D. Filimonov, and F. Jelinek, "Contextual information improves oov detection in speech," in *Proceedings NAACL-HLT*, 2010.

[73] B. Roark, R. Sproat, and S. Izhak, "Lexicographic semirings for exact automata encoding of sequence models," in *Proceedings of ACL-HLT*, 2011, pp. 1–5.

[74] M. Mohri, "Semiring frameworks and algorithms for shortest-distance problems," *Journal of Automata, Languages and Combinatorics*, vol. 7, no. 3, pp. 321–350, 2002.

[75] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," *Proceedings of the International Conference on Implementation and Application of Automata*, vol. 4783, pp. 11–23, 2007.

[76] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, 1983.

[77] S. F. Chen, "Shrinking exponential language models," in *Proc. NAACL-HLT*, 2009.

[78] S. F. Chen, L. Mangu, B. Ramabhadran, and e. al, "Scaling shrinkage-based language models," *Proc. ASRU*, 2009.

[79] J. G. Kahn, M. Ostendorf, and C. Chelba, "Parsing conversational speech using enhanced segmentation," in *Proceeding of Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics annual meeting*, 2004.

[80] Y. Liu, "Structural event detection for rich transcription of speech," Ph.D. dissertation, Purdue University, 2004.

[81] H.-K. J. Kuo, B. Kingsbury, and G. Zweig, "Discriminative training of decoding graphs for large vocabulary continuous speech recognition," in *Proc. ICASSP*, vol. 4, 2007, pp. 45–48.

[82] I. Shafran and K. Hall, "Corrective models for speech recognition of inflected languages," in *Proceedings of EMNLP*, 2006.

[83] E. Arsoy, M. Saraçlar, B. Roark, and I. Shafran, "Discriminative language modeling with linguistic and statistically derived features," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 20, no. 2, pp. 540–550, 2012.

[84] A. Rastrow, M. Dreyer, A. Sethy, S. Khudanpur, B. Ramabhadran, and M. Dredze, "Hill climbing on speech lattices : A new rescoring framework," in *Proceeding of ICASSP*, 2011.

[85] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *EMNLP*, 2002, pp. 1–8.

[86] Y.-Y. Wang, D. Yu, T.-C. Ju, and A. Acero, "An introduction to voice search," *IEEE Signal Processing Magazine*, vol. 25, no. 3, pp. 29–38, 2008.

[87] Y. Freund and R. Schapire, "Large margin classification using the perceptron algorithm," *Machine Learning*, vol. 3, no. 37, pp. 277–296, 1999.

[88] M. Collins and B. Roark, "Incremental parsing with the perceptron algorithm," in *Proceedings of ACL*, 2004.

[89] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *NAACL-HLT*, 2010.

[90] A. Arun, B. Haddow, P. Koehn, A. Lopez, P. Blunsom, and C. Dyer, "Monte carlo techniques for phrase-based translation," *Machine Translation*, vol. 24, no. 2, August 2010.

[91] L. Shen, G. Satta, and A. Joshi, "Guided learning for bidirectional sequence classification," in *Annual Meeting-Association for Computational Linguistics*, vol. 45, no. 1, 2007, p. 760.

[92] P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar, "An end-to-end discriminative approach to machine translation," in *Proceedings COLING-ACL*, 2006.

[93] G. Kurata, N. Itoh, and M. Nishimura, "Acoustically discriminative training for language models," in *Proceedings of ICASSP*, 2009, pp. 4717–4720.

[94] G. Kurata, A. Sethy, B. Ramabhadran, A. Rastrow, N. Itoh, and M. Nishimura, "Acoustically Discriminative Language Model Training with Pseudo-Hypothesis," *Speech Communication*, vol. 54, no. 2, pp. 219–228, 2010.

[95] P. Jyothi and E. Fosler-Lussier, "Discriminative language modeling using simulated ASR errors," in *Proceedings of INTERSPEECH*, 2010.

[96] K. Sagae, M. Lehr, E. Prud'hommeaux, P. Xu, and E. Al, "HALLUCINATED

N-BEST LISTS FOR DISCRIMINATIVE LANGUAGE MODELING," in *Proceedings of ICASSP*, 2012.

[97] P. Xu, D. Karakos, and S. Khudanpur, "Self-supervised discriminative training of statistical language models," in *Proceedings of IEEE Workshop on Automatic Speech Recognition Understanding (ASRU)*, 2009, pp. 317–322.

[98] D. McAllester, T. Hazan, and J. Keshet, "Direct loss minimization for structured prediction," in *Advances in Neural Information Processing Systems*, vol. 23, 2010, pp. 1594–1602.

[99] S. Petrov, P.-C. Chang, M. Ringgaard, and H. Alshawi, "Uptraining for accurate deterministic question parsing," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, October 2010, pp. 705–713.

[100] Y. Goldberg and M. Elhadad, "An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing," in *Proc. HLT-NAACL*, no. June, 2010, pp. 742–750.

[101] X. Zhu, "Semi-Supervised learning literature survey," University of Wisconsin-Madison, Tech. Rep. Computer Sciencces Technical Report 1530, 2005.

[102] J. Kazama and J. Tsujii, "Evaluation and extension of maximum entropy mod-

els with inequality constraints," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2003.

[103] S. J. Benson and J. J. More, "A limited-memory variable-metric algorithm for bound-constrained minimization," Argonne National Laboratory, Tech. Rep. ANL/MCS-P909-0901, 2001.

[104] K. B. Hall, "Best-first word-lattice parsing: techniques for integrated syntactic language modeling," Ph.D. dissertation, Brown University, 2005.

[105] J. Cheppalier, M. Rajman, R. Aragues, and A. Rozenknop, "Lattice parsing for speech recognition," in *Sixth Conference sur le Traitement Automatique du Langage Naturel (TANL'99)*, 1999.

[106] Z. Huang, M. Harper, and S. Petrov, "Self-training with Products of Latent Variable Grammars," in *Proc. EMNLP*, no. October, 2010, pp. 12–22.

Vita

Ariya Rastrow received the B.S. degree in Electrical Engineering at Sharif University of Technology (SUT)—one of the best engineering schools of Iran—in 2006. He joined the Electrical and Computer Engineering PhD program at Johns Hopkins University in September 2006 where he has been a member of the Center for Language and Speech Processing (CLSP) and the Human Language Technology Center of Excellence (HLT-COE). During the course of his doctoral studies at Johns Hopkins University had the opportunity to work as an intern at the IBM T.J Watson research lab from May 2008 to January 2009 on Automatic Speech Recognition. He has been the recipient of various fellowships, including the inaugural "Frederick Jelinek" Fellowship supported by Google and Dean′s fellowship at Johns Hopkins University.

In June 2012, Ariya will start as a Research Scientist at Amazon in Seattle, Washington working on developing speech and language technology.