ABSTRACT

Title of dissertation: MODELING DEPENDENCIES IN NATURAL LANGUAGES WITH LATENT VARIABLES

Zhongqiang Huang, Doctor of Philosophy, 2011

Dissertation directed by: Professor Mary Harper
Department Computer Science

In this thesis, we investigate the use of latent variables to model complex dependencies in natural languages. Traditional models, which have a fixed parameterization, often make strong independence assumptions that lead to poor performance. This problem is often addressed by incorporating additional dependencies into the model (e.g., using higher order N-grams for language modeling). These added dependencies can increase data sparsity and/or require expert knowledge, together with trial and error, in order to identify and incorporate the most important dependencies (as in lexicalized parsing models). Traditional models, when developed for a particular genre, domain, or language, are also often difficult to adapt to another.

In contrast, previous work has shown that latent variable models, which automatically learn dependencies in a data-driven way, are able to flexibly adjust the number of parameters based on the type and the amount of training data available. We have created several different types of latent variable models for a diverse set of natural language processing applications, including novel models for part-of-speech tagging, language modeling, and machine translation, and an improved model for parsing. These models perform significantly better than traditional models. We

have also created and evaluated three different methods for improving the performance of latent variable models. While these methods can be applied to any of our applications, we focus our experiments on parsing.

The first method involves self-training, i.e., we train models using a combination of gold standard training data and a large amount of automatically labeled training data. We conclude from a series of experiments that the latent variable models benefit much more from self-training than conventional models, apparently due to their flexibility to adjust their model parameterization to learn more accurate models from the additional automatically labeled training data.

The second method takes advantage of the variability among latent variable models to combine multiple models for enhanced performance. We investigate several different training protocols to combine self-training with model combination. We conclude that these two techniques are complementary to each other and can be effectively combined to train very high quality parsing models.

The third method replaces the generative multinomial lexical model of latent variable grammars with a feature-rich log-linear lexical model to provide a principled solution to address data sparsity, handle out-of-vocabulary words, and exploit overlapping features during model induction. We conclude from experiments that the resulting grammars are able to effectively parse three different languages.

This work contributes to natural language processing by creating flexible and effective latent variable models for several different languages. Our investigation of self-training, model combination, and log-linear models also provides insights into the effective application of these machine learning techniques to other disciplines.

MODELING DEPENDENCIES IN NATURAL LANGUAGES WITH LATENT VARIABLES

by

Zhongqiang Huang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:
Dr. Philip Resnik, Chair
Dr. Mary Harper, Co-Chair/Advisor
Dr. Carol Espy-Wilson, Dean's Representative
Dr. Hal Daumé III
Dr. Amol Deshpande

Dedication

To my parents, wife, and son.

Acknowledgments

First and foremost I owe my deepest gratitude to my advisor Mary Harper. This thesis would not have been possible without her invaluable support over the last few years. I have benefited tremendously from her insightful guidance and her persistent commitment to help me grow as a researcher. She keeps encouraging me and has always made herself available for help and advice. The experience of working with and learning from such an extraordinary individual is one that I will cherish forever.

I would like to thank Philip Resnik, Carol Espy-Wilson, Hal Daumé III, and Amol Deshpande for serving on my thesis committee. I sincerely appreciate their suggestions and support of my research.

Thanks are due to Bowen Zhou, Martin Čmejrek, Evelyne Tzoukermann, and Tony Davis for their guidance and support during my internships at IBM T. J. Watson Research Center and StreamSage/Comcast. I thank my officemate Songfang Huang at Watson Research Center for interesting conversations on various topics.

I thank all of the professors, postdocs, and students in the Laboratory for Computational Linguistics and Information Processing for broadening my knowledge about natural language processing and making my graduate life enjoyable. I would like to especially thank Jimmy Lin for teaching me about cloud computing, Doug Oard, Kristy Hollingshead Seitz, Denis Filimonov, Vladimir Eidelman, Ke Wu, and Ferhan Ture for insightful discussions at the MT gang meeting, and Tamer Elsayed, Hendra Setiawan, Yuval Martin, Asad Sayeed, Amit Goyal, Lidan Wang, Jagadeesh

Jagarlamudi, Ke Zhai, Yuening Hu, and others for many useful conversations and help on all sorts of things.

Thanks are also due to Slav Petrov for providing the source code of Berkeley parser and for his collaboration on the product model, and to Lei Chen, Wen Wang, Liu Yang, Spence Green, Izhak Shafran, Dustin Hillard, Heng Ji, and many others for their help and collaboration on various papers.

Last but not least, I would like to thank my loving parents and my wife for their faith in me and my son for bringing joy to my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Overview

This thesis investigates the use of latent variable models to better model complex dependencies among units of natural languages, such as words, parts-of-speech (POS) of words, syntactic constituents, and word-aligned phrase pairs. Tractable models for natural language processing (NLP) often make independence assumptions, such as Markov assumptions for language modeling [2], hidden Markov assumptions for POS tagging [34], context-free assumptions for parsing [12], and synchronous context-free assumptions for machine translation [40]. For traditional models that have a fixed parameterization, overly strong independence assumptions often lead to poor performance levels. A common approach to mitigate this problem is to incorporate additional dependencies and use higher-order models. For N-gram language models, it is a simple matter to replace bigrams with trigrams or even higher-order N-grams; however, these models then suffer from greater data sparsity issues and require more training data and/or more effective smoothing methods to obtain reliable parameter estimates [38]. For models like lexicalized parsing grammars [31, 48], expert knowledge is often required, together with trial and error, in order to determine and incorporate the most important dependencies. Moreover, traditional models, when developed for a particular genre, domain, or language, are

also often difficult to adapt to another, as a tremendous amount of effort would be required to adjust the model parameterization to account for the change.

In contrast, latent variable models are able to capture complex dependencies through latent variables in a data-driven way. Take factor analysis [5, 6, 11] for example. A vector of mutually dependent continuous variables $\mathbf{O} = O_1, \cdots, O_d$ distributed according to a multivariate Gaussian distribution $\mathcal{N}(\mu, \mathbf{\Sigma})$ can be approximated by a latent variable model:

$$\mathbf{O}' = \mathbf{W}\mathbf{X} + \mu + \mathbf{u}$$

where $\mathbf{X} = X_1, \cdots, X_q$ $(q < d)$ is a vector of mutually independent latent variables distributed according to a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{u}$ is a noise model $\mathcal{N}(\mathbf{0}, \mathbf{\Phi})$ with diagonal covariance $\mathbf{\Phi}$, and $\mathbf{\Phi}$ and $\mathbf{W}$ are adaptive parameters. In this approximation, $O'_1, \cdots, O'_d$ are independent of each other given $\mathbf{X}$, i.e.,

$$\mathrm{P}(\mathbf{O}'|\mathbf{X}) = \prod_{i=1}^{d} \mathrm{P}(O'_i|\mathbf{X})$$

however, these variables are actually interdependent on each other and their joint distribution $\mathrm{P}(\mathbf{O}')$ calculated by:

$$\mathrm{P}(\mathbf{O}') \;=\; \int \mathrm{P}(\mathbf{O}'|\mathbf{X})\mathrm{P}(\mathbf{X})d\mathbf{X}$$

is a Gaussian distribution with mean μ and covariance matrix $\mathbf{\Phi} + \mathbf{W}\mathbf{W}^T$. By learning proper values of $\mathbf{\Phi}$ and $\mathbf{W}$ on the training data, the latent variable model is

able to capture the dependencies among $\mathbf{O}$ by approximating the covariance matrix $\mathbf{\Sigma}$ of the true distribution with $\mathbf{\Phi} + \mathbf{W}\mathbf{W}^T$.

Latent variable models are also able to flexibly adjust the number of parameters based on the type and the amount of training data available to learn the most important dependencies, as demonstrated by Petrov et al. [126] and Petrov [121] for probabilistic context-free grammars with latent annotations (PCFG-LA) [106]. Building upon this previous work, we have created several different types of latent variable models for a diverse set of natural language processing applications, including novel models for part-of-speech tagging, language modeling, and machine translation, and an improved model for parsing. The latent variable models are able to capture dependencies which otherwise could not be captured using conventional Markov, hidden Markov, context-free, and synchronous context-free assumptions, and perform significantly better than traditional models. We have also created and evaluated three different methods for improving the performance of latent variable models. While these methods can be applied to any of our applications, we focus our experiments on parsing with PCFG-LA grammars.

The first method involves self-training, a semi-supervised learning method to utilize unlabeled training data in model training. In self-training, we first train a model on some gold standard training data, then use it to automatically label a large amount of unlabeled training data, and finally re-train a new model on the combination of the gold standard training data and the automatically labeled training data. In contrast to conventional models that only benefit from self-training when the initial training data is small, our experiments on tagging and parsing show

that latent variable models consistently benefit from self-training regardless of the size of the initial training data, apparently due to the flexibility to adjust their model parameterization to learn more accurate models from the additional automatically labeled training data. Our self-trained latent variable parsers achieve state-of-the-art parsing accuracies for a single parser on the English Penn treebank (91.5% F) and the Chinese Penn treebank (85.2% F).

The second method builds upon the work of Petrov [122] that takes advantage of the variability among latent variable models to combine multiple models using a product model for enhanced performance. Since the product model is more accurate than the individual component models, it is able to generate more accurate automatically labeled data for self-training. The models trained with this automatically labeled data are also more accurate, and in turn, can be combined into a product model to achieve even greater parsing accuracies. We investigate several different training protocols to exploit the complementary effects of self-training and model combination for parsing with latent variable models. We conclude that self-training and product models can be effectively combined to train very high quality parsing models with accuracies of 92.5% F on the English Penn treebank and 89.6% F on the English Broadcast News treebank.

The third method replaces the generative multinomial lexical model of PCFG-LA grammars with a feature-rich log-linear lexical model to provide a principled solution to address data sparsity, handle out-of-vocabulary (OOV) words, and exploit overlapping features during model induction. We conclude from experiments that the resulting grammars are able to effectively parse three different languages,

with absolute improvements of 1% F, 1.7% F, and 2.7% F on English, Chinese, and Arabic, respectively.

In summary, this thesis contributes to natural language processing by creating high quality latent variable models for a diverse set of applications over several different languages. Our investigations using self-training, model combination, and log-linear models to improve latent variable models also provides insight into the effective application of these machine learning techniques to other disciplines.

1.2 Structure of the Thesis

The rest of the thesis is organized as follows:

- In Chapter 2, we describe the mathematical foundations of the latent variable models studied in this thesis. We describe the development of Markov models, hidden Markov models, and finally latent hidden Markov models, and then discuss how latent variable models are able to capture complex dependencies beyond the order of independence assumptions made by the other models. Training and inference algorithms are also presented.

- In Chapter 3, we first review prior work on POS tagging, discuss issues related to their strong independence assumptions, and then present our latent bigram tagger to automatically learn dependencies from the training data. Our experiments show that, compared to conventional bigram and trigram HMM taggers, the latent variable tagger is significantly more accurate and is also able to benefit much more from automatically labeled training data through

self-training.

- In Chapter 4, we first review prior work on N-gram language models and class-based language models and then present a latent language model based on the latent bigram tagger in order to circumvent the strong independence assumptions of the standard N-gram models. Our experiments show that the latent variable approach effectively learns dependencies among words in the training data, achieves significantly lower perplexity than a conventional word bigram language model, and outperforms a strong word trigram language model.

- In Chapter 5, we review prior work on two parsing models, a lexicalized parser with a fixed parameterization and a PCFG-LA parser, and compare how they perform across languages with varying amounts of training data. Our experiments show that the PCFG-LA parser achieves greater accuracy and also benefits much more from automatically labeled training data through self-training. Our analyses show that the success of the latent variable models comes from their ability to adjust their model parameterization to learn more accurate models from the additional automatically labeled training data.

- In Chapter 6, we review the prior work that exploits the variability among latent variable models through the use of model combination and then investigate several different training protocols to exploit the complementary effects of self-training and model combination for creating effective PCFG-LA grammars. Our experiments show that these two approaches can be effectively

combined to train very high quality parsing models.

- In Chapter 7, we present a principled feature-rich log-linear lexical model to address data sparsity, handle out-of-vocabulary (OOV) words, and exploit overlapping features during model induction. Our experiments on three languages show that the resulting grammars are more flexible to train and achieve greater accuracy than standard PCFG-LA grammars.

- In Chapter 8, we present a novel approach to induce latent syntactic categories and use them as soft syntactic constraints for machine translation. Our experiments show that this approach improves the baseline hierarchical phrase-based translation system on both English-to-German and English-to-Chinese tasks.

- In Chapter 9, we summarize the contributions of this thesis and discuss directions for future work.

Chapter 2

Model Foundations

2.1 Overview

In this chapter, we describe the mathematical foundations of the latent variable models studied in this thesis. We describe the development of Markov models, hidden Markov models, and finally latent hidden Markov models, and discuss how latent variable models are able to capture complex dependencies beyond the order of independence assumptions made by the other models. We also describe the learning and inference algorithms for latent variable models. Context-free grammars for syntactic parsing and synchronous context-free grammars for machine translation, which can also be enriched with latent variables, are briefly described in this chapter, with more specific details presented in Chapter 5 and Chapter 8, respectively. Table 2.1 lists some key notation used throughout this chapter.

2.2 A Markov Model

Many language processing models operate on a sentence, which is a grammatical unit of natural language consisting of a sequence of words. Let Σ be a finite alphabet representing the set of words of a language. A sentence[1] o_1^n can be viewed as being generated by a stochastic process $\{O_i\}$. According to the chain rule, the

[1] We assume that a sentence o_1^n always ends with a special end-of-sentence word $o_n = \text{EOS\_OBS}$.

Notation	**Meaning**
λ	The model parameter
O, X, or Z	The random variable for a single observation, state, or latent state
o, x, or z	The single observation, state, or latent state
$\{O_i\}$	The observation process, similarly for $\{X_i\}$ and $\{Z_i\}$
o_i	The i-th observation, similarly for x_i and x_i
o_i^j	The subsequence of observations $o_i, \cdots, o_j$, similarly for x_i^j and z_i^j
Σ	The inventory of observations
$\mathcal{X}(o)$	The state inventory of observation o
$\mathcal{Z}(x)$	The latent state inventory of state x
$\mathcal{Z}^{-1}(z)$	The state corresponding to the latent state z of a latent HMM
$E = \{\mathrm{P}_e(\cdot\|\cdot)\}$	The emission probabilities of an HMM or a latent HMM
$T = \{\mathrm{P}_t(\cdot\|\cdot)\}$	The transition probabilities of an HMM or a latent HMM
$\pi = \{\mathrm{P}_\pi(\cdot)\}$	The prior (latent) state probabilities of an HMM (or latent HMM)
$\alpha(\cdot,\cdot), \alpha(\cdot,\cdot,\cdot)$	The forward probability (defined differently for different tasks)
$\beta(\cdot,\cdot), \beta(\cdot,\cdot,\cdot)$	The backward probability (defined differently for different tasks)
SOS_OBS	The universal value for o_0
EOS_OBS	The universal value for o_n
SOS_STATE	The universal value for x_0
EOS_STATE	The universal value for x_n
SOS_LSTATE	The universal value for z_0
EOS_LSTATE	The universal value for z_n

Table 2.1: Notation Table

joint distribution of the words in a sentence can be computed as the product of conditional probabilities[2]:

$$\mathrm{P}(O_1^n = o_1^n) = \prod_{i=1}^{n} \mathrm{P}(O_i = o_i | O_1^{i-1} = o_1^{i-1}) \tag{2.1}$$

in which the distribution of the i-th word o_i depends on all of the preceding words. This is appropriate for natural languages because sentences exhibit long-distance dependencies [43]; however, it is not practical because a distribution conditioned on the complete history cannot be estimated reliably given any reasonably sized training data. Although tractable models that account for long-distance dependencies exist [77, 109], the most widely used models are based on Markov independence assumptions [78]. In the rest of this thesis, when clear from context, we will omit the values of random variables and write, for example, $O_1^n = o_1^n$ as O_1^n and $\mathrm{P}(O_i = o_i | O_1^{i-1} = o_1^{i-1})$ as $\mathrm{P}(O_i | O_1^{i-1})$ for brevity.

Definition 2.2.1 *A stochastic process $\{O_i\}$ is said to be a Markov chain of order m, where m is finite, if the process satisfies:*

$$P(O_i | O_1^{i-1}) = P(O_i | O_{i-m}^{i-1}) \tag{2.2}$$

A Markov chain is said to be stationary if the probabilities $\mathrm{P}(O_i | O_{i-m}^{i-1})$ do not depend on the index i. All of the stochastic processes discussed in this chapter are stationary. Figure 2.1 depicts the generation process of a Markov chain.

[2] $\mathrm{P}(O_1 = o_1 | O_1^0 = o_1^0)$ represents the probability of the first word.

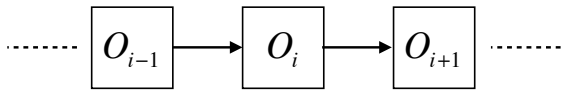


Figure 2.1: The generation process of a Markov chain

Markov chain models are widely used in N-gram language models. When modeled as a first-order Markov chain, the distribution of word o_i in a sentence is independent of everything else given the previous word o_{i-1}. This is an overly strong independence assumption that is unable to capture any dependency beyond a word bigram. A common method to add more context is to use a higher order Markov assumption; however, this greatly increases the data sparsity problem and requires strong smoothing methods to obtain reliable parameter estimates [38]. We next describe an alternative approach to capture dependencies among words using the state process of a hidden Markov model.

2.3 A Hidden Markov Model

2.3.1 Definition and Properties

Definition 2.3.1 *A first-order hidden Markov model (HMM) is a statistical Markov model of two parallel stochastic processes $\{O_i\}$ and $\{X_i\}$, where $\{X_i\}$ is a first-order Markov chain representing the hidden process and $\{O_i\}$ is an observable process in which each random variable O_i is independent of everything else when conditioned on X_i. Each observation o_i takes a value from Σ and each state x_i takes a value*

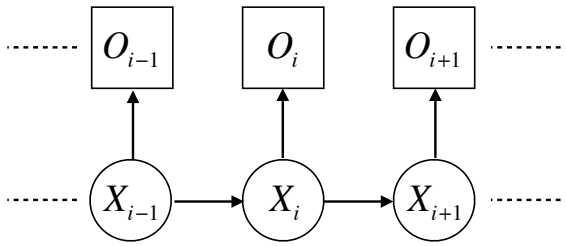


Figure 2.2: The generation process of an HMM

from $\mathcal{X}(o_i)$*, the state inventory*[3] *of* o_i*.*

Figure 2.2 depicts the generation process of an HMM. The parameter $\lambda = \{\pi, T, E\}$ of an HMM comprises three components: π represents the prior state distribution $\mathrm{P}_\pi(x_1|\lambda)$, T represents the state transition probabilities $\mathrm{P}_t(x_i|x_{i-1}, \lambda)$, and E represents the emission probabilities $\mathrm{P}_e(o_i|x_i, \lambda)$ of observations given a state. To ensure proper boundary conditions, we assume that there is always a start-of-sequence state $x_0 = \text{SOS\_STATE}$ and the sequence o_1^n (or x_1^n) always ends with an end-of-sequence observation (or state) $o_n = \text{EOS\_OBS}$ (or $x_n = \text{EOS\_STATE}$). When clear from context, we omit λ from the condition and write, for example, $\mathrm{P}_t(x_i|x_{i-1}, \lambda)$ as $\mathrm{P}_t(x_i|x_{i-1})$ and $\mathrm{P}_\pi(x_1|\lambda)$ as $\mathrm{P}_t(x_1|x_0)$ for brevity.

When modeled by an HMM, the joint probability of a word sequence o_1^n and its state sequence x_1^n can be factored as follows:

$$\mathrm{P}(O_1^n, X_1^n) = \prod_{i=1}^{n} \mathrm{P}(X_i|O_1^{i-1}, X_1^{i-1})\mathrm{P}(O_i|O_1^{i-1}, X_1^i) = \prod_{i=1}^{n} \mathrm{P}(X_i|X_{i-1})\mathrm{P}(O_i|X_i) \quad (2.3)$$

The joint probability of the word sequence o_1^n can then be computed by marginalizing

[3]It should be emphasized that each observation symbol has its own state inventory, which may overlap with the state inventory of another observation symbol.

over the states. Despite the assumption that the words are independent of each other given their states, the words themselves are dependent on each other when the states are marginalized out, as shown below:

$$\begin{aligned} \mathrm{P}(O_i|O_1^{i-1}) &= \sum_{\substack{x_{i-1}\in\mathcal{X}(o_{i-1}) \\ x_i\in\mathcal{X}(o_i)}} \mathrm{P}(O_i, X_i = x_i, X_{i-1} = x_{i-1}|O_1^{i-1}) \\ &= \sum_{\substack{x_{i-1}\in\mathcal{X}(o_{i-1}) \\ x_i\in\mathcal{X}(o_i)}} \mathrm{P}(X_{i-1} = x_{i-1}|O_1^{i-1})\mathrm{P}_t(x_i|x_{i-1})\mathrm{P}_e(o_i|x_i) \end{aligned} \quad (2.4)$$

In Equation 2.4, the history's impact on the distribution of word o_i is represented by the conditional distribution of state x_{i-1}, which generally depends on all of the previous words[4]. As a result, an HMM is able to capture longer dependencies than can be captured by a Markov chain model.

Another advantage of using HMMs over Markov chain models for modeling sequences is that the states can be viewed as clusters that abstract common phenomena from the observations (e.g., singular or mass nouns are clustered to a NN tag), thus providing smoothing [90] to alleviate the data sparsity problem that is common in many NLP applications. As we will show later in Chapter 4, an HMM-based language model is significantly better than a word bigram language model (first order Markov chain) at capturing dependencies among words, as measured by perplexity, and it can even outperform a strong word trigram language model (second order Markov chain).

The modeling capability of HMMs increases with the size of the state space

[4]If $|\mathcal{X}(o_{i-1})| = 1$, then $\mathrm{P}(X_{i-1} = x_{i-1}|O_1^{i-1})$ would be consistently 1 for the only state, and all of the history information before o_{i-1} would be ignored.

$\mathcal{X}(o)$ for each observation o; however, data sparsity could then become more severe when the training data is limited in size. A higher m-th order ($m > 1$) HMM is capable of capturing longer dependencies in general, but it can be equivalently viewed as a first-order HMM with an expanded state inventory, in which each state represents a concatenation of m states in the m-th order HMM.

2.3.2 Inference and Learning

Three natural problems [130] arise typically when using HMMs:

Likelihood Computation: given an HMM with model parameter $\lambda = \{\pi, T, E\}$, calculate the probability $\mathrm{P}(O_1^n = o_1^n|\lambda)$ that sequence o_1^n is generated by the HMM.

Inference: given an HMM with model parameter $\lambda = \{\pi, T, E\}$, find the most probable state sequence $\hat{x}_1^n$ that would have generated an observation sequence o_1^n, i.e., $\hat{x}_1^n = \arg\max_{x_1^n} \mathrm{P}(X_1^n = x_1^n|O_1^n = o_1^n, \lambda)$.

Learning: given the topological structure[5] of an HMM and a collection of observation sequences $\mathcal{T}$, find model parameter $\hat{\lambda} = \{\pi, T, E\}$ such that $\hat{\lambda} = \arg\max_{\lambda} \mathrm{P}(\mathcal{T}|\lambda)$.

These problems have been extensively studied in the literature. For example, see (Rabiner [129]) for applications tailored to speech recognition and (Cappé et al. [25]) for a mathematical treatment of HMMs. We will next briefly describe the algorithms for solving these three problems.

[5]The topological structure defines the state inventory $\mathcal{X}(o)$ for each observation o and the permissible state transitions.

2.3.2.1 Likelihood Computation

Given an HMM with parameter $\lambda = \{\pi, T, E\}$ and an observation sequence o_1^n, we define the forward probability $\alpha(i, x)$ as the probability of observing subsequence o_1^i with o_i being generated by $x_i = x$, i.e., $\alpha(i, x) = \mathrm{P}(O_1^i, X_i = x|\lambda)$. We also define the backward probability $\beta(i, x)$ as the probability of observing sequence o_{i+1}^n given that $x_i = x$, i.e., $\beta(i, x) = \mathrm{P}(O_{i+1}^n|X_i = x, \lambda)$. The forward probabilities can be computed recursively by dynamic programming [49] as follows:

$$\begin{aligned}
\alpha(i, x) &= \sum_{x' \in \mathcal{X}(o_{i-1})} \mathrm{P}(O_1^i, X_{i-1} = x', X_i = x|\lambda) \\
&= \sum_{x' \in \mathcal{X}(o_{i-1})} \mathrm{P}(O_1^{i-1}, X_{i-1} = x'|\lambda)\mathrm{P}(X_i = x|X_{i-1} = x')\mathrm{P}(O_i|X_i = x) \\
&= \sum_{x' \in \mathcal{X}(o_{i-1})} \alpha(i-1, x')\mathrm{P}_t(x|x')\mathrm{P}_e(o_i|x) \qquad (2.5)
\end{aligned}$$

with base case $\alpha(0, \mathrm{SOS\_STATE}) = 1$. The backward probabilities can be computed similarly as follows:

$$\begin{aligned}
\beta(i, x) &= \sum_{x' \in \mathcal{X}(o_{i+1})} \mathrm{P}(O_{i+1}^n, X_{i+1} = x'|X_i = x, \lambda) \\
&= \sum_{x' \in \mathcal{X}(o_{i+1})} \mathrm{P}(X_{i+1} = x'|X_i = x)\mathrm{P}(O_{i+1}|X_{i+1} = x')\mathrm{P}(O_{i+2}^n|X_{i+1} = x', \lambda) \\
&= \sum_{x' \in \mathcal{X}(o_{i+1})} \mathrm{P}_t(x'|x)\mathrm{P}_e(o_{i+1}|x')\beta(i+1, x') \qquad (2.6)
\end{aligned}$$

with base case $\beta(n, \mathrm{EOS\_STATE}) = 1$. The likelihood of observing any sequence o_1^n

can then be computed easily from the forward probabilities:

$$\mathrm{P}(O_1^n = o_1^n|\lambda) = \mathrm{P}(O_1^n = o_1^n, X_n = \text{EOS\_STATE}|\lambda) = \alpha(n, \text{EOS\_STATE}) \quad (2.7)$$

or from the backward probabilities:

$$\mathrm{P}(O_1^n = o_1^n|\lambda) = \mathrm{P}(O_1^n = o_1^n|X_0 = \text{SOS\_STATE}, \lambda) = \beta(0, \text{SOS\_STATE})$$

2.3.2.2 Inference

The decoding process searches for the most probable state sequence $\hat{x}_1^n$ given an observation sequence o_1^n:

$$\begin{aligned}
\hat{x}_1^n &= \arg\max_{x_1^n} \mathrm{P}(X_1^n = x_1^n|O_1^n = o_1^n, \lambda) \\
&= \arg\max_{x_1^n} \mathrm{P}(O_1^n = o_1^n, X_1^n = x_1^n|\lambda) \\
&= \arg\max_{x_1^n} \mathrm{P}(X_i|X_{i-1})\mathrm{P}(O_i|X_i)
\end{aligned}$$

This can be computed by the Viterbi algorithm [153]. We define the viterbi probability $\alpha'(i, x)$ as the joint probability of o_1^i and its most probable state sequence that ends with $x_i = x$, i.e.,

$$\alpha'(i, x) = \max_{\substack{1 \le k \le i-1 \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(O_1^i, X_1^{i-1}, X_i = x|\lambda) \quad (2.8)$$

Similarly to the forward probabilities, the viterbi probabilities can be com-

puted recursively:

$$
\begin{aligned}
&\alpha'(i,x) \\
&= \max_{x' \in \mathcal{X}(o_{i-1})} \max_{\substack{1 \le k \le i-2 \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(O_1^i, X_1^{i-2}, X_{i-1} = x', X_i = x|\lambda) \\
&= \max_{x' \in \mathcal{X}(o_{i-1})} \max_{\substack{1 \le k \le i-2 \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(O_1^{i-1}, X_1^{i-2}, X_{i-1} = x'|\lambda)\mathrm{P}(X_i = x|X_{i-1} = x')\mathrm{P}(O_i|X_i = x_i) \\
&= \max_{x' \in \mathcal{X}(o_{i-1})} \alpha'(i-1,x')\mathrm{P}_t(x|x')\mathrm{P}_e(o_i|x) \qquad (2.9)
\end{aligned}
$$

with base case $\alpha'(0, \text{SOS\_STATE}) = 1$. The most probable state sequence $\hat{x}_1^n$ can be retrieved by back-tracing the decisions up to the computation of $\alpha'(n, \text{SOS\_STATE})$:

$$
\hat{x}_{i-1} = \arg \max_{x \in \mathcal{X}(o_{i-1})} \alpha'(i-1,x)\mathrm{P}_t(\hat{x}_i|x)\mathrm{P}_e(o_i|\hat{x}_i)
$$

with base case $\hat{x}_n = \text{EOS\_STATE}$.

2.3.2.3 Learning

Given the topological structure of an HMM and a collection of independent and identically distributed training samples $\mathcal{T}$ that are assumed to be generated by an HMM, the goal of maximum likelihood estimation (MLE) is to find parameter $\hat{\lambda}$ such that the likelihood of the training samples is maximized, i.e.,

$$
\hat{\lambda} = \arg \max_{\lambda} \mathrm{P}(\mathcal{T}|\lambda)
$$

The likelihood function is unfortunately not concave with respect to the model

parameter and there is no approach that guarantees an optimal solution. A common solution is to use the Baum-Welch algorithm [7], a particular instance of the generalized expectation-maximization algorithm (EM) [50], to iteratively increase the lower bound of the likelihood function (see (Borman [13]) for a short tutorial). The EM algorithm iterates between the E-step and the M-step.

The E-step computes, $Q(\lambda', \lambda)$, the expected value of the complete log-likelihood of the new model parameter λ' with respect to the posterior distribution of the state sequence x_1^n given the observation sequence o_1^n under the current model parameter λ. That is:

$$
\begin{aligned}
Q(\lambda', \lambda) &= \sum_{o_1^n \in \mathcal{T}} \mathrm{E}_{X_1^n | O_1^n, \lambda} \log \mathrm{P}(O_1^n, X_1^n | \lambda') \qquad (2.10) \\
&= \sum_{o_1^n \in \mathcal{T}} \sum_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(X_1^n | O_1^n, \lambda) \log \mathrm{P}(O_1^n, X_1^n | \lambda') \\
&= \sum_{o_1^n \in \mathcal{T}} \sum_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(X_1^n | O_1^n, \lambda) \log (\prod_{i=1}^{n} \mathrm{P}(X_i | X_{i-1}, \lambda') \mathrm{P}(O_i | X_i, \lambda')) \\
&= \sum_{o_1^n \in \mathcal{T}} \sum_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \sum_{i=1}^{n} \mathrm{P}(X_1^n | O_1^n, \lambda) \log \mathrm{P}(X_i | X_{i-1}, \lambda') \\
&\quad + \sum_{o_1^n \in \mathcal{T}} \sum_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \sum_{i=1}^{n} \mathrm{P}(X_1^n | O_1^n, \lambda) \log \mathrm{P}(O_i | X_i, \lambda') \\
&= \sum_{o_1^n \in \mathcal{T}} \sum_{i=1}^{n} \sum_{\substack{x' \in \mathcal{X}(o_{i-1}) \\ x \in \mathcal{X}(o_i)}} \mathrm{P}(X_{i-1} = x', X_i = x | O_1^n, \lambda) \log \mathrm{P}_t(x | x', \lambda') \\
&\quad + \sum_{o_1^n \in \mathcal{T}} \sum_{i=1}^{n} \sum_{x \in \mathcal{X}(o_i)} \mathrm{P}(X_i = x | O_1^n, \lambda) \log \mathrm{P}_e(o_i | x, \lambda')
\end{aligned}
$$

where the posterior state probabilities $\mathrm{P}(X_i = x | O_1^n, \lambda)$ and state transition proba-

bilities $\mathrm{P}(X_{i-1} = x', X_i = x|O_1^n, \lambda)$ can be computed from the forward (Equation 2.5) and backward probabilities (Equation 2.6) as shown in Equation 2.11 and 2.12, respectively:

$$\begin{aligned}
&\mathrm{P}(X_i = x|O_1^n, \lambda) \\
&= \frac{\mathrm{P}(O_1^i, X_i = x, O_{i+1}^n|\lambda)}{\mathrm{P}(O_1^n|\lambda)} \\
&= \frac{\mathrm{P}(O_1^i, X_i = x|\lambda)\mathrm{P}(O_{i+1}^n|X_i = x, \lambda)}{\mathrm{P}(O_1^n|\lambda)} \\
&= \frac{\alpha(i,x)\beta(i,x)}{\mathrm{P}(O_1^n|\lambda)}
\end{aligned} \tag{2.11}$$

$$\begin{aligned}
&\mathrm{P}(X_{i-1} = x', X_i = x|O_1^n, \lambda) \\
&= \frac{\mathrm{P}(O_1^{i-1}, X_{i-1} = x', X_i = x, O_i, O_{i+1}^n, |\lambda)}{\mathrm{P}(O_1^n|\lambda)} \\
&= \frac{\mathrm{P}(O_1^{i-1}, X_{i-1} = x'|\lambda)\mathrm{P}(X_i = x|X_{i-1} = x')\mathrm{P}(O_i|X_i = x)\mathrm{P}(O_{i+1}^n|X_i = x, \lambda)}{\mathrm{P}(O_1^n|\lambda)} \\
&= \frac{\alpha(i-1,x')\mathrm{P}_t(x|x')\mathrm{P}_e(o_i|x)\beta(i,x)}{\mathrm{P}(O_1^n|\lambda)}
\end{aligned} \tag{2.12}$$

The M-step finds the new model parameter λ' that maximizes $Q(\lambda, \lambda')$. It can be shown that the maximum is achieved with the following parameter update formulas [130]:

$$\begin{aligned}
\mathrm{P}_t(x|x', \lambda') &= \frac{\sum_{o_1^n \in \mathcal{T}} \sum_{i=1}^n \mathrm{P}(X_{i-1} = x', X_i = x|O_1^n, \lambda)}{\sum_{o_1^n \in \mathcal{T}} \sum_{i=1}^n \mathrm{P}(X_{i-1} = x'|O_1^n, \lambda)} \\
\mathrm{P}_e(o|x, \lambda') &= \frac{\sum_{o_1^n \in \mathcal{T}} \sum_{i=1}^n \delta(o_i, o)\mathrm{P}(X_i = x|O_1^n, \lambda)}{\sum_{o_1^n \in \mathcal{T}} \sum_{i=1}^n \mathrm{P}(X_i = x|O_1^n, \lambda)}
\end{aligned}$$

where $\delta(\cdot,\cdot)$ returns 1 if the two operands are identical and 0 otherwise.

2.4 A Latent Hidden Markov Model

2.4.1 Definition and Properties

HMMs have been extensively used in sequence labeling tasks such as part-of-speech tagging, in which words are the observations and POS tags are the states. Given a sentence o_1^n and an HMM with parameter $\lambda = \{\pi, T, E\}$, the tagging process finds the most probable tag sequence $\hat{x}_1^n$, which can be solved by the Viterbi algorithm described in Section 2.3.2.2:

$$\hat{x}_1^n = \arg \max_{\substack{1 \le i \le n \\ x_i \in \mathcal{X}(o_i)}} \mathrm{P}(O_1^n, X_1^n = x_1^n | \lambda)$$

One problem of using HMMs for sequence labeling is that the joint process $\{O_i, X_i\}$ is Markovian, as shown below:

$$\begin{aligned} \mathrm{P}(O_i, X_i | O_1^{i-1}, X_1^{i-1}) &= \mathrm{P}(X_i | O_1^{i-1}, X_1^{i-1}) \mathrm{P}(O_i | O_1^{i-1}, X_1^i) \\ &= \mathrm{P}(X_i | X_{i-1}) \mathrm{P}(O_i | X_i) \\ &= \mathrm{P}(O_i, X_i | O_{i-1}, X_{i-1}) \end{aligned}$$

This means any history information about the joint process of $\{O_i, X_i\}$ is ignored in the prediction of future observation/state pairs when the value of the current observation/state pair is known, which is a problem similar to using Markov chain

models for language modeling. Just as latent variables can be introduced to Markov chain models to capture longer dependencies, latent variables can also be introduced to HMMs for the same purpose.

Definition 2.4.1 *A latent hidden Markov model is an extension of a hidden Markov model for statistical modeling of three parallel stochastic processes:* $\{O_i\}$, $\{X_i\}$, *and* $\{Z_i\}$. $\{O_i\}$ *is the observation process,* $\{X_i\}$ *is the state process. Each state* x *is split by function* $\mathcal{Z}(\cdot)$ *into a set of latent states such that* $P(X = x|Z = z) = 1$ *if* $z \in \mathcal{Z}(x)$ *and 0 otherwise*[6]. *A latent state* z *can be mapped back deterministically by function* $\mathcal{Z}^{-1}(\cdot)$ *to state* $x = \mathcal{Z}^{-1}(z)$. $\{Z_i\}$ *is a first-order Markov process in which each* $z_i \in \mathcal{Z}(x_i)$ *represents a latent state of state* $x_i \in \mathcal{X}(o_i)$. *Each observation* o_i *is independent of everything else given its latent state* z_i.

Figure 2.3 depicts the generation process of a latent HMM. The parameter $\lambda = \{\pi, T, E\}$ of a latent HMM comprises three components: π represents the prior distribution of the latent states $\mathrm{P}_\pi(z_1|\lambda)$, T represents the latent state transition probabilities $\mathrm{P}_t(z_i|z_{i-1}, \lambda)$, and E represents the emission probabilities $\mathrm{P}_e(o_i|z_i, \lambda)$ of observations given the latent states. Latent HMMs are essentially HMMs for modeling the joint process of $\{O_i, Z_i\}$ with the constraint that the latent states are clustered into states, and the relationship between the $\{O_i\}$ process and the $\{X_i\}$ process is mediated through the latent state process $\{Z_i\}$. Such models were first introduced in (Krogh [87, 88]) to describe HMMs in which states have shared labels

[6]It is possible to share latent states across different states; however, for the problems investigated in this thesis, all states (e.g., the POS tags) have clear distinctions and so we choose to assign latent states exclusively to a state.

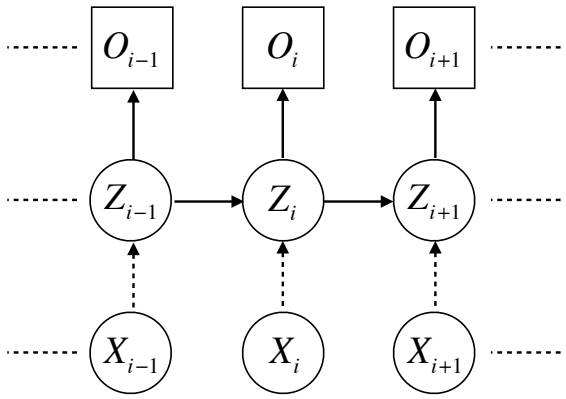


Figure 2.3: The generation process of a latent HMM

and have been applied successfully to sequence annotation tasks in bioinformatics [19, 89, 96]. Different from the prior work, the latent states in our model are automatically induced from the training data as we will describe in Section 2.4.2.3.

Given our definition, the joint distribution of the parallel process $\{O_i, X_i\}$ is computed as follows[7]:

$$\begin{aligned}
\mathrm{P}(O_1^n, X_1^n) &= \sum_{\substack{1 \le k \le n \\ z_k \in \mathcal{Z}(x_k)}} \mathrm{P}(O_1^n, X_1^n, Z_1^n = z_1^n) \\
&= \sum_{\substack{1 \le k \le n \\ z_k \in \mathcal{Z}(x_k)}} \mathrm{P}(O_1^n, Z_1^n = z_1^n) \\
&= \sum_{\substack{1 \le k \le n \\ z_k \in \mathcal{Z}(x_k)}} \prod_{i=1}^{n} \mathrm{P}(Z_i | Z_{i-1}) \mathrm{P}(O_i | Z_i)
\end{aligned}$$

Unlike HMMs, the joint process $\{O_i, X_i\}$ modeled by a latent HMM is no longer

[7]In terms of the boundary condition of latent HMMs, we do not split the SOS_STATE nor the EOS_STATE, and their only latent state is SOS_LSTATE and EOS_LSTATE, respectively.

Markovian, as shown in the following equation:

$$
\begin{aligned}
&\mathrm{P}(O_i, X_i | O_1^{i-1}, X_1^{i-1}) \\
&= \sum_{\substack{z_{i-1} \in \mathcal{Z}(x_{i-1}) \\ z_i \in \mathcal{Z}(x_i)}} \mathrm{P}(O_i, X_i, Z_i = z_i, Z_{i-1} = z_{i-1} | O_1^{i-1}, X_1^{i-1}) \\
&= \sum_{\substack{z_{i-1} \in \mathcal{Z}(x_{i-1}) \\ z_i \in \mathcal{Z}(x_i)}} \mathrm{P}(Z_{i-1} = z_{i-1} | O_1^{i-1}, X_1^{i-1}) \mathrm{P}_t(z_i | z_{i-1}) \mathrm{P}_e(o_i | z_i) \qquad (2.13)
\end{aligned}
$$

In Equation 2.13, like Equation 2.4, the history's impact on the distribution of observation/state pair (o_i, x_i) is represented by the conditional distribution of latent state z_{i-1}, which generally depends on all of the previous observation/state pairs[8]. As a result, a latent HMM is able to capture longer dependencies than can be captured by an HMM.

2.4.2 Inference and Learning

Since the relationship between $\{O_i\}$ and $\{Z_i\}$ in latent HMMs is essentially the same as the relationship between $\{O_i\}$ and $\{X_i\}$ in HMMs, the inference and learning algorithms of HMMs can also be applied to latent HMMs. However, in sequence labeling problems using latent HMMs[9], we are more interested in utilizing process $\{Z_i\}$ to better capture the dependencies in the joint process of $\{O_i, X_i\}$. Hence, we are typically interested in the following three problems:

[8] If $|\mathcal{Z}(x_{i-1})| = 1$, then $\mathrm{P}(Z_{i-1} = z_{i-1} | O_1^{i-1}, X_1^{i-1})$ would be consistently 1 for the only latent state, and all of the history information before (o_{i-1}, x_{i-1}) would be ignored.

[9] When applied to POS tagging, the states are the POS tags and the latent states are the fine-grained latent POS tags. It is the POS tags that we want to recover accurately from an input sentence at decoding time, not the latent states.

Likelihood Computation: given a latent HMM with model parameter $\lambda = \{\pi, T, E\}$, calculate the probability $\mathrm{P}(O_1^n = o_1^n, X_1^n = x_1^n|\lambda)$ of observing sequence o_1^n with state sequence x_1^n.

Inference: given a latent HMM and its model parameter $\lambda = \{\pi, T, E\}$, find the most probable state sequence $\hat{x}_1^n$ given the observation sequence o_1^n.

Learning: given a collection $\mathcal{T}$ of observation sequences $\{o_1^n\}$ accompanied by their state sequences $\{x_1^n\}$, find model parameter $\hat{\lambda} = \{\pi, T, E\}$ of a latent HMM such that $\hat{\lambda} = \arg\max_\lambda \mathrm{P}(\mathcal{T}|\lambda)$.

We will next briefly describe the algorithms for solving these problems.

2.4.2.1 Likelihood Computation

This problem is similar to the likelihood computation problem in HMMs. We define the forward probability $\alpha(i, z)$ as the probability of generating o_1^i and x_1^i, with the latent state of x_i being $z \in \mathcal{Z}(x_i)$, i.e, $\alpha(i, z) = \mathrm{P}(O_1^i, X_1^i, Z_i = z|\lambda)$. Similarly, we define backward probability $\beta(i, z)$ as the probability of generating o_{i+1}^n and x_{i+1}^n given that the latent state of x_i is $z \in \mathcal{Z}(x_i)$, i.e., $\beta(i, z) = \mathrm{P}(O_{i+1}^n, X_{i+1}^n|Z_i = z, \lambda)$. The forward probabilities can be computed recursively as follows:

$$\begin{aligned}
\alpha(i, z) &= \sum_{z' \in \mathcal{Z}(x_{i-1})} \mathrm{P}(O_1^i, X_1^i, Z_{i-1} = z', Z_i = z|\lambda) \\
&= \sum_{z' \in \mathcal{Z}(x_{i-1})} \alpha(i-1, z')\mathrm{P}_t(z|z')\mathrm{P}_e(o_i|z)
\end{aligned}$$

with base case $\alpha(0, \text{SOS\_LSTATE}) = 1$. The backward probabilities can be computed similarly as follows:

$$\begin{aligned}\beta(i, z) &= \sum_{z' \in \mathcal{Z}(x_{i+1})} \mathrm{P}(O_{i+1}^n, X_{i+1}^n, Z_{i+1} = z' | Z_i = z, \lambda) \\ &= \sum_{z' \in \mathcal{Z}(x_{i+1})} \mathrm{P}_t(z'|z) \mathrm{P}_e(o_{i+1}|z') \beta(i+1, z')\end{aligned}$$

with base case $\beta(n, \text{EOS\_LSTATE}) = 1$. The joint probability of (o_1^n, x_1^n) can be then computed easily from the forward probabilities:

$$\mathrm{P}(O_1^n, X_1^n | \lambda) = \mathrm{P}(O_1^n, X_1^n, Z_n = \text{EOS\_LSTATE} | \lambda) = \alpha(n, \text{EOS\_LSTATE})$$

or from the backward probabilities:

$$\mathrm{P}(O_1^n, X_1^n | \lambda) = \mathrm{P}(O_1^n, X_1^n | Z_0 = \text{SOS\_LSTATE} | \lambda) = \beta(n, \text{SOS\_LSTATE})$$

2.4.2.2 Inference

Decoding with latent HMMs is more complicated than HMMs (see (Matsuzaki et al. [106], Petrov and Klein [123]) for related problems and solutions for context-free grammars with latent annotations). Recall that decoding with HMMs refers to finding the most probable state sequence $\hat{x}_1^n$ given an observation sequence o_1^n, i.e.,

$$\hat{x}_1^n = \arg\max_{x_1^n} \mathrm{P}(X_1^n = x_1^n | O_1^n = o_1^n, \lambda) \tag{2.14}$$

Decoding with latent HMMs could, however, involve any of the following three different tasks:

1. Finding the most probable latent state sequence $\hat{z}_1^n$ given the observation sequence o_1^n:

$$\hat{z}_1^n = \arg\max_{z_1^n} \mathrm{P}(Z_1^n = z_1^n | O_1^n = o_1^n, \lambda) \tag{2.15}$$

2. Finding the most probable latent state sequence $\hat{z}_1^n$ given both the observation sequence o_1^n and the state sequence x_1^n:

$$\hat{z}_1^n = \arg\max_{z_1^n} \mathrm{P}(Z_1^n = z_1^n | O_1^n = o_1^n, X_1^n = x_1^n, \lambda) \tag{2.16}$$

3. Finding the most probable state sequence $\hat{x}_1^n$ given the observation sequence o_1^n:

$$\hat{x}_1^n = \arg\max_{x_1^n} \mathrm{P}(X_1^n = x_1^n | o_1^n = o_1^n, \lambda) = \arg\max_{x_1^n} \sum_{\substack{1 \le i \le n \\ z_i \in \mathcal{Z}(x_i)}} \mathrm{P}(Z_1^n = z_1^n | O_1^n = o_1^n, \lambda) \tag{2.17}$$

Since the relationship between $\{O_i\}$ and $\{Z_i\}$ in a latent HMM is essentially modeled as an HMM, the first task can be solved by an algorithm similar to the Viterbi algorithm for decoding with HMMs. The second task is similar to the first task except that the search space is constrained by the state sequence, and it can also be solved efficiently by a variant of the Viterbi algorithm. However, the third task is intractable (see (Brejová et al. [17]) for a proof of its NP-hardness) because the search space consists of combinatorially many x_1^n sequences, each requiring summation over

probabilities of combinatorially many z_1^n sequences.

Several approximate solutions exist for the third task. One method is to first extract the N-best state sequences using a HMM, and then select the one with the highest probability with respect to the latent HMM. Another method is to first extract the best latent state sequence (the task in Equation 2.15) and then simply apply the $\mathcal{Z}^{-1}(\cdot)$ function to retrieve the corresponding state sequence. Both methods have obvious limitations. The quality of the HMM directly affects the accuracy of the first method, while the second method relies on the assumption that the most probable latent state sequence dominates the probability mass of the corresponding state sequence, which is not necessarily true.

Note that the most probable state sequence minimizes the expected 0-1 loss of a hypothesized state sequence x_1^n. x_1^n has loss 0 if it is exactly the same as the unknown reference $\bar{x}_1^n$ and loss 1 otherwise, regardless of the number of errors on the individual states:

$$\begin{aligned} \hat{x}_1^n &= \arg\max_{x_1^n} \mathrm{E}_{\bar{X}_1^n = \bar{x}_1^n | O_1^n = o_1^n} \delta(x_1^n, \bar{x}_1^n) \\ &= \arg\max_{x_1^n} \mathrm{P}(X_1^n = x_1^n | O_1^n = o_1^n) \end{aligned}$$

Instead of taking 0-1 loss on the entire state sequence, we can take it on the individual states, i.e., the hypothesized state x_i has loss 0 if it is identical to the unknown reference state $\bar{x}_i$ and loss 1 otherwise. This gives the following most

probable state (MostProb-S) estimation:

$$\begin{aligned}\hat{x}_1^n &= \arg\max_{x_1^n} \sum_{i=1}^{n} \mathrm{E}_{\bar{X}_i=\bar{x}_i|O_1^n=o_1^n} \delta(x_i, \bar{x}_i) \\ &= \arg\max_{x_1^n} \sum_{i=1}^{n} \mathrm{P}(X_i = x_i | O_1^n = o_1^n)\end{aligned}$$

The optimal state sequence $\hat{x}_1^n$ under this decoding criterion can be calculated efficiently given the values of $\mathrm{P}(X_i = x_i|O_1^n = o_1^n)$, which can be computed in a way similar to HMMs as in Equation 2.11. We define the forward probability $\alpha(i, x, z) = \mathrm{P}(O_1^i, X_i = x, Z_i = z|\lambda)$ as the probability of generating observation sequence o_1^i with the state and latent state at i being x and $z \in \mathcal{Z}(x)$, respectively, and the backward probability $\beta(i, x, z) = \mathrm{P}(O_{i+1}^n|X_i = x, Z_i = z, \lambda)$ as the probability of generating observation sequence o_{i+1}^n given that the state and latent state at i are x and $z \in \mathcal{Z}(x)$, respectively. The forward probabilities can be computed recursively as follows:

$$\begin{aligned}\alpha(i, x, z) &= \sum_{x' \in \mathcal{X}(o_{i-1})} \sum_{z' \in \mathcal{Z}(x')} \mathrm{P}(O_1^i, X_{i-1} = x', Z_{i-1} = z', X_i = x, Z_i = z|\lambda) \\ &= \sum_{x' \in \mathcal{X}(o_{i-1})} \sum_{z' \in \mathcal{Z}(x')} \mathrm{P}(O_1^i, X_{i-1} = x', Z_{i-1} = z'|\lambda) \mathrm{P}_t(z|z') \mathrm{P}_e(o_i|z) \\ &= \sum_{x' \in \mathcal{X}(o_{i-1})} \sum_{z' \in \mathcal{Z}(x')} \alpha(i-1, x', z') \mathrm{P}_t(z|z') \mathrm{P}_e(o_i|z)\end{aligned}$$

with base case $\alpha(0, \mathrm{SOS\_STATE}, \mathrm{SOS\_LSTATE}) = 1$. The backward probabilities

can be computed similarly as follows:

$$\begin{aligned}
\beta(i,x,z) &= \sum_{x' \in \mathcal{X}(o_{i+1})} \sum_{z' \in \mathcal{Z}(x')} \mathrm{P}(O_{i+1}^n, X_{i+1} = x', Z_{i+1} = z' | X_i = x, Z_i = z, \lambda) \\
&= \sum_{x' \in \mathcal{X}(o_{i+1})} \sum_{z' \in \mathcal{Z}(x')} \mathrm{P}_e(z'|z) \mathrm{P}_O(o_i|z') \mathrm{P}(O_{i+2}^n | X_{i+1} = x', Z_{i+1} = z', \lambda) \\
&= \sum_{x' \in \mathcal{X}(o_{i+1})} \sum_{z' \in \mathcal{Z}(x')} \mathrm{P}_e(z'|z) \mathrm{P}_O(o_i|z') \beta(i+1, x', z')
\end{aligned}$$

with base case $\beta(n, \text{EOS\_STATE}, \text{EOS\_LSTATE}) = 1$. Once the forward and backward probabilities are computed, $\mathrm{P}(X_i = x | O_1^n, \lambda)$ can be easily computed by:

$$\mathrm{P}(X_i = x | O_1^n, \lambda) = \frac{\mathrm{P}(O_1^n, X_i = x | \lambda)}{\mathrm{P}(O_1^n | \lambda)} = \frac{\sum_{z \in \mathcal{Z}(x)} \alpha(i,x,z)\beta(i,x,z)}{\mathrm{P}(O_1^n | \lambda)}$$

where $\mathrm{P}(O_1^n | \lambda) = \alpha(n, \text{EOS\_STATE}, \text{EOS\_LSTATE})$.

We can take this one step further and take 0-1 loss on the individual transitions. A transition between a pair of states (x_{i-1}, x_i) has loss 0 if it matches the unknown reference transition $(\bar{x}_{i-1}, \bar{x}_i)$ and loss 1 otherwise. This gives the most probable transition (MostProb-T) estimation:

$$\begin{aligned}
\hat{x}_1^n &= \arg\max_{x_1^n} \sum_{i=1}^n \mathrm{E}_{\bar{X}_{i-1} = \bar{x}_{i-1}, \bar{X}_i = \bar{x}_i | O_1^n = o_1^n} \delta(x_{i-1}, \bar{x}_{i-1}) \delta(x_i, \bar{x}_i) \\
&= \arg\max_{x_1^n} \sum_{i=1}^n \mathrm{P}(X_{i-1} = x_{i-1}, X_i = x_i | O_1^n = o_1^n)
\end{aligned}$$

where $\mathrm{P}(X_{i-1} = x', X_i = x | O_1^n, \lambda)$ can be computed from the forward and backward

probabilities:

$$\begin{aligned}
\mathrm{P}(X_{i-1} = x', X_i = x | O_1^n, \lambda) &= \frac{\mathrm{P}(O_1^n, X_{i-1} = x', X_i = x | \lambda)}{\mathrm{P}(O_1^n | \lambda)} \\
&= \frac{\sum\limits_{\substack{z' \in \mathcal{Z}(x') \\ z \in \mathcal{Z}(x)}} \alpha(i-1, x', z') \mathrm{P}_t(z|z') \mathrm{P}_e(o_i|z) \beta(i, x, z)}{\mathrm{P}(O_1^n | \lambda)}
\end{aligned}$$

and the optimal state sequence $\hat{x}_1^n$ can then be computed using dynamic programming.

A problem with both of the MostProb-S and MostProb-T approaches is that the summation is dominated by high probability states and transitions, and so the method could effectively ignore the existence of low probability states and transitions, which are more likely to cause errors. When parsing using context-free grammars with latent variables, Petrov and Klein [123] showed that it is better to use product instead of summation to discourage low probability components. Applying the same idea to latent HMMs, we obtain the MostProb-S-P method based on individual states:

$$\hat{x}_1^n = \arg\max_{x_1^n} \prod_{i=1}^{n} \mathrm{P}(X_i = x_i | O_1^n = o_1^n)$$

and the MostProb-T-P method based on state transitions:

$$\hat{x}_1^n = \arg\max_{x_1^n} \prod_{i=1}^{n} \mathrm{P}(X_{i-1} = x_{i-1}, X_i = x_i | O_1^n = o_1^n) \tag{2.18}$$

Our experiments show that the MostProb-T-P method is the most effective

method of all of the tractable decoding methods described above, and thus it will be used in Chapter 3 for POS tagging.

2.4.2.3 Learning

Given a collection $\mathcal{T}$ of observation sequences $\{o_1^n\}$ accompanied by their state sequences $\{x_1^n\}$, the goal of maximum likelihood estimation (MLE) is to find parameter $\hat{\lambda}$ of a latent HMM such that the likelihood of the training samples is maximized, i.e.,

$$\hat{\lambda} = \arg\max_{\lambda} \mathrm{P}(\mathcal{T}|\lambda) \tag{2.19}$$

Similarly to HMMs, this objective function is non-concave and can be optimized indirectly by the EM algorithm. Let λ be the current parameter. At each EM iteration, we find the new parameter λ' that maximizes the auxiliary function $Q(\lambda', \lambda)$:

$$Q(\lambda', \lambda) = \sum_{(o_1^n, x_1^n) \in \mathcal{T}} \mathrm{E}_{Z_1^n | O_1^n, X_1^n, \lambda} \log \mathrm{P}(O_1^n, X_1^n, Z_1^n | \lambda')$$

which can be rewritten, in a way similar to Equation 2.10, as:

$$\begin{aligned} Q(\lambda, \lambda') \;=\; & \sum_{(o_1^n, x_1^n) \in \mathcal{T}} \sum_{i=1}^{n} \sum_{\substack{z \in \mathcal{Z}(x_i) \\ z' \in \mathcal{Z}(x_{i-1})}} \mathrm{P}(Z_{i-1} = z', Z_i = z | O_1^n, X_1^n, \lambda) \log \mathrm{P}_t(z | z', \lambda') \\ & + \sum_{(o_1^n, x_1^n) \in \mathcal{T}} \sum_{i=1}^{n} \sum_{z \in \mathcal{Z}(x_i)} \mathrm{P}(Z_i = z | O_1^n, X_1^n, \lambda) \log \mathrm{P}_e(o_i | z, \lambda') \end{aligned}$$

Maximizing the above equation with respect to the new model parameter λ'

produces the following parameter update formulas for any $z \in \mathcal{Z}(x)$ and $z' \in \mathcal{Z}(x')$:

$$\begin{aligned}
\mathrm{P}_t(z|z',\lambda') &= \frac{\sum_{(o_1^n,x_1^n)\in\mathcal{T}} \sum_{i=1}^{n} \delta(x_{i-1},x')\delta(x_i,x)\mathrm{P}(Z_{i-1}=z', Z_i=z|O_1^n,X_1^n,\lambda)}{\sum_{(o_1^n,x_1^n)\in\mathcal{T}} \sum_{i=1}^{n} \delta(x_{i-1},x')\mathrm{P}(Z_{i-1}=z'|O_1^n,X_1^n,\lambda)} \\
\mathrm{P}_e(o|z,\lambda') &= \frac{\sum_{(o_1^n,x_1^n)\in\mathcal{T}} \sum_{i=1}^{n} \delta(x_i,x)\delta(o_i,o)\mathrm{P}(Z_i=z|O_1^n,X_1^n)}{\sum_{(o_1^n,x_1^n)\in\mathcal{T}} \sum_{i=1}^{n} \delta(x_i,x)\mathrm{P}(Z_i=z|O_1^n,X_1^n,\lambda)}
\end{aligned}$$

The above EM training algorithm assumes a given set of latent states for each state, but how are these latent states induced? The goal of using latent states is to model dependencies of the joint process $\{O_i, X_i\}$ that cannot be modeled by a hidden Markov model, and so it is desirable to allocate more latent states to states that occur in more complex contexts than others. When applied to POS tagging, we split each POS tag into a set of latent tags. The fact that some POS tags such as NN (normal noun) and VB (base verb) appear in more complex contexts than others like IJ (interjection) suggests that using a fixed number of latent tags for all POS tags would be too limited (under-training) for some POS tags and too much (over-training) for others. Metrics such as the frequency of a POS tag, the number of unique words associated with a tag, or the conditional entropy of a tag all relate to the complexity of a tag's context, and might be used to decide the number of latent tags for a POS tag. However, they do not directly relate to the likelihood score of the training data, which is the training objective.

Finding the latent states that contribute most to the improvement of training likelihood is a hard task, because there is no way to accurately measure how a set of latent states contributes to an increase in the training likelihood until a sufficient number of EM iterations is carried out. The hierarchical split-merge (SM) approach

taken in (Petrov et al. [126]) (for inducing latent syntactic categories in PCFG grammars) gradually increases the number of latent categories while allocating them adaptively to places where they would produce the greatest increase in training likelihood. At each iteration, this approach first splits each current latent category into two, followed by several rounds of EM to learn parameters associated with the new latent categories. It then merges the least useful splits back (based on an approximate loss in training likelihood if they are merged), again followed by additional rounds of EM to re-adjust parameters. This approach was shown in (Petrov et al. [126]) to significantly outperform a method that assigns the same number of latent categories to each syntactic category [106]. We adopt the same approach to induce latent states for latent HMMs and will next describe the splitting and merging operations.

Splitting This operation splits each current latent state[10] into two and initializes the transition and emission probabilities associated with the new latent states. Suppose latent state z is split into z_1 and z_2, and latent state z' is split into z'_1 and z'_2. The transition probabilities from z_i to z'_j for $i, j \in \{1, 2\}$ are initialized as:

$$\mathrm{P}_t(z'_j|z_i) \;=\; \frac{1+(-1)^j\delta}{2}\mathrm{P}_t(z'|z) \;\text{ for } i,j \in \{1,2\}$$

where δ is a small random number between 0 and $\frac{\mathrm{P}_t(z'|z)}{100}$. The emission prob-

[10]Except SOS_LSTATE and EOS_LSTATE.

ability of observation o given latent state z_i for $i \in \{1, 2\}$ is initialized as:

$$\mathrm{P}_e(w|z_i) \;=\; \mathrm{P}_e(w|z)$$

We could introduce some randomness into the emission probabilities as in the transition probabilities, but it is not necessary because the randomness in transition probabilities is sufficient to break the symmetry of model parameters through iterations of EM training.

Merging This operation first calculates the approximate loss in training likelihood if two recently split latent states are merged back to the original latent state, and then merges a certain percentage of the splits with the least approximate loss. Let x_{i-1}, x_i, and x_{i+1} be three consecutive states in a training example (o_1^n, x_1^n), z_1 and z_2 be two latent states of state x_i, and o be the observation of state x_i. The likelihood of the training example can be computed based on the forward and backward probabilities:

$$\mathrm{P}_{\mathrm{old}}(O_1^n, X_1^n) = \sum_{z \in \mathcal{Z}(x_i)} \alpha_{\mathrm{old}}(i, z)\beta_{\mathrm{old}}(i, z)$$

Now consider merging latent states z_1 and z_2 into a single latent state $z_{1,2}$. Before merging, the transition and emission probabilities associated with z_k

$(k \in \{1, 2\})$ are:

$$\begin{aligned}
\mathrm{P}_t(z_k|z') &= \frac{c(z', z_k)}{c(z')} \text{ for } z' \in \mathcal{Z}(x_{i-1}) \\
\mathrm{P}_t(z'|z_k) &= \frac{c(z_k, z')}{c(z_k)} \text{ for } z' \in \mathcal{Z}(x_{i+1}) \\
\mathrm{P}_e(o|z_k) &= \frac{c(z_k, o)}{c(z_k)}
\end{aligned}$$

where $c(\cdot, \cdot)$ and $c(\cdot)$ are the expected counts accumulated in the previous E-step of the EM iteration. If we only merge z_1 and z_2 of state x_i in a particular training example into $z_{1,2}$, and maintain all of the other latent states, the transition and emission probabilities associated with $z_{1,2}$ become:

$$\begin{aligned}
\mathrm{P}_t(z_{1,2}|z') &= \frac{c(z', z_1) + c(z', z_2)}{c(z')} \text{ for } z' \in \mathcal{Z}(x_{i-1}) \\
\mathrm{P}_t(z'|z_{1,2}) &= \frac{c(z_1, z') + c(z_2, z')}{c(z_1) + c(z_2)} \text{ for } z' \in \mathcal{Z}(x_{i+1}) \\
\mathrm{P}_e(o|z_{1,2}) &= \frac{c(z_1, o) + c(z_2, o)}{c(z_1) + c(z_2)}
\end{aligned}$$

The new forward and backward probabilities for latent state $z_{1,2}$ become:

$$\begin{aligned}
\alpha_{\text{new}}(i, z_{1,2}) &= \sum_{z' \in \mathcal{Z}(x_{i-1})} \alpha_{\text{old}}(i-1, z') \mathrm{P}_t(z_{1,2}|z') \mathrm{P}_e(o|z_{1,2}) \\
\beta_{\text{new}}(i, z_{1,2}) &= \sum_{z' \in \mathcal{Z}(x_{i+1})} \mathrm{P}_t(z'|z_{1,2}) \mathrm{P}_e(o|z') \beta_{\text{old}}(i+1, z')
\end{aligned}$$

Algorithm 1 Split-Merge Latent HMM Training Algorithm

Initialize λ based on the HMM estimated on the training data
for $i = 1$ to N_{sm} **do**
  Do Splitting as described on Page 33
  Run N_s iterations of EM
  Do Merging as described on Page 34
  Run N_m iterations of EM

and the new likelihood after merging these two latent tags becomes:

$$\begin{aligned} & \mathrm{P}_{\mathrm{new}}(O_1^n, X_1^n) \\ &= \mathrm{P}_{\mathrm{old}}(O_1^n, X_1^n) - \sum_{k=1}^{2} \alpha_{\mathrm{old}}(i, z_k) * \beta_{\mathrm{old}}(i, z_k) + \alpha_{\mathrm{new}}(i, z_{1,2}) \beta_{\mathrm{new}}(i, z_{1,2}) \end{aligned}$$

The loss in log-likelihood can then be approximated by:

$$\log \frac{\mathrm{P}_{\mathrm{old}}(O_1^n, X_1^n)}{\mathrm{P}_{\mathrm{new}}(O_1^n, X_1^n)}$$

For each pair of latent tags split from the same latent state, we accumulate its approximate loss of log-likelihood across the training set and merge it back to a single latent state if the approximate loss is less than 50% of the other split pairs.

Algorithm 1 summarizes the procedure for training a latent HMM. The number of split-merge iterations (N_{sm}) and the number of EM iterations after splitting (N_s) and merging (N_m) are determined based on a development set.

2.5 Probabilistic Context-Free Grammars and Latent Annotations

A probabilistic context-free grammar (PCFG) [43] is a probabilistic model that describes the hierarchical generation process of a sentence.

Definition 2.5.1 *A PCFG grammar G is a tuple: $G = (\mathcal{V}, \Sigma, \mathcal{R}, S, P)$ where:*

- *$\mathcal{V}$ is a finite set of nonterminals representing different types of syntactic categories.*
- *Σ is a finite set of terminals, disjoint from V, representing the alphabet of a language.*
- *$\mathcal{R}$ is a set of rewriting rules of the form $X \to \gamma$, where $X \in \mathcal{V}$ and $\gamma \in (\mathcal{V} \cup \Sigma)^*$.*
- *$S \in \mathcal{V}$ is the unique root symbol that represents the whole sentence.*
- *P assigns a probability to each rule in $\mathcal{R}$ such that $\sum_{\gamma: X \to \gamma \in R} P(X \to \gamma) = 1$.*

A PCFG generates a sentence as follows. Starting from the root node with nonterminal S, any node that is on the left hand side of a rewriting rule is replaced with the right hand side according to the probabilistic distribution P, until all leaf nodes are terminals, which constitute the sentence. PCFGs make a similar independence assumption as in HMMs, i.e., the probability that a nonterminal is expanded via some rule is independent of everything else given the nonterminal itself. It is well-known that the independence assumption in PCFGs is unrealistically strong that they are unable to effectively model the dependencies in the generation process

of natural sentences. Matsuzaki et al. [106] addressed this problem by introducing latent annotations to the nonterminals, similar to the way that latent states are introduced to HMMs in Section 2.4. The resulting grammar is called a PCFG grammar with latent annotations (PCFG-LA). We will describe the details of PCFG-LA grammars in Chapter 5 and then present a series of methods in Chapters 5, 6, and 7 to improve PCFG-LA grammars and obtain high quality parsing models for a variety of languages.

2.6 Synchronous Probabilistic Context-Free Grammars and Latent Annotations

Synchronous probabilistic context-free grammars (SPCFGs) [76] are an extension of PCFG grammars to model a pair of languages synchronously. SPCFGs are formally defined as follows:

Definition 2.6.1 *A SPCFG grammar G is a tuple: $G = (\mathcal{V}, \Sigma_1, \Sigma_2, \mathcal{R}, S, P)$ where*

- *$\mathcal{V}$ is a finite set of nonterminals representing the same set of different types of syntactic categories in two languages.*
- *Σ_1 is a finite set of terminals, disjoint from V, representing the alphabet of the first language.*
- *Σ_2 is a finite set of terminals, disjoint from V, representing the alphabet of the second language.*

- $\mathcal{R}$ *is a set of rewriting rules of the form* $X \rightarrow \langle \gamma_1, \gamma_2 \rangle$*, where* $X \in \mathcal{V}$*,* $\gamma_1 \in (\Sigma_1 \cup (\mathcal{V} \times \mathbb{N}))^*$*,* $\gamma_2 \in (\Sigma_2 \cup (\mathcal{V} \times \mathbb{N}))^*$*, and every* $(X, n) \in \mathcal{V} \times \mathbb{N}$ *can occur at most once in either* γ_1 *and* γ_2 *and whenever it occurs in* γ_1*, it also occurs in* γ_2*, and vice versa.*

- $S \in \mathcal{V}$ *is the unique root symbol that represents the whole sentence.*

- P *assigns a probability to each rule in* $\mathcal{R}$ *such that* $\sum_{\gamma_1, \gamma_2 : X \rightarrow \langle \gamma_1, \gamma_2 \rangle \in R} P(X \rightarrow \langle \gamma_1, \gamma_2 \rangle) = 1$.

SPCFGs suffer from the same independence problem as in PCFGs and HMMs, and can also be enhanced by introducing latent categories to the nonterminals. However, this addition would dramatically increase decoding cost when applied to hierarchical phrase-based machine translation systems. In Chapter 8, we will describe an alternative approach to introduce latent annotations to SPCFG grammars to capture and enforce syntactic constraints for machine translation in an efficient and effective way.

2.7 Conclusions

In this chapter, we discussed the strong independence assumptions of Markov models and HMMs and the problems that arise from making such strong assumptions. We then described how to address these problems by using HMMs and latent HMMS. The inference and learning algorithms for HMMs and latent HMMs were also presented. We also briefly discussed similar issues with PCFG grammars and

SPCFG grammars and how they can be addressed by introducing latent variables. In the next chapter, we will develop and evaluate a latent bigram POS tagger based on a latent HMM.

Chapter 3

POS Tagging with Latent Variables

3.1 Overview

Generative probabilistic part-of-speech (POS) tagging models typically make an assumption that the assignment of tags to words is only dependent on local factors and is independent of everything else given the local constraints. While these models are practical and fairly effective, they are unable to utilize and benefit from longer dependency constraints that are inherent in natural languages.

We model the intrinsic dependencies among words and tags using latent variables by splitting POS tags of a bigram HMM tagger into fine-grained latent tags that are better able to capture contextual and lexical constraints. The proposed approach has the ability to learn fine-grained latent tags at different levels of granularity, and so it is able to adjust the number of parameters based on the amount of training data.

Our experiments on both Chinese and English show that the latent bigram tagger significantly outperforms conventional bigram and trigram taggers. We also observe that the latent bigram tagger is more effective than conventional models when learning from additional training data that is automatically labeled.

This chapter is organized as follows. Section 3.2 provides a short introduction to POS tagging. Section 3.3 and 3.4 briefly review previous research on generative

and discriminative POS taggers, respectively. Section 3.5 describes our approach to building a bigram POS tagger with latent variables. Section 3.6 investigates the use of self-training for POS tagging. Experimental results are presented in Section 3.7. The last section concludes this chapter.

3.2 Introduction to POS Tagging

POS tagging is the process of assigning each word in a sentence with a POS tag, e.g., NN for common nouns and JJ for adjectives. POS tagging is often a prerequisite step for many natural language processing tasks such as named entity detection [112], parsing [48], sentence boundary detection [98], and advanced systems such as machine translation [75] and information retrieval [102].

POS tags are linguistic categories that are generally defined based on the syntactic or morphological behaviors of words. The common POS tags in English include noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection, and there are many more categories and subcategories. The set of POS tags varies for different languages, as well as for different corpora of the same language. For example, nouns in some languages can be further divided into plural, possessive, and singular forms, and verbs in some languages can be marked for tense, aspect, etc.

The challenge for POS tagging arises from the fact that many words have multiple POS tags. Table 3.1 shows the breakdown of word types in the Brown Corpus for English based on their degree of ambiguity in terms of POS tags [51]. It

Degree of ambiguity	Number of word types
Unambiguous (1 tag)	35340
Ambiguous (2-7 tags)	4100
2 tags	3760
3 tags	264
4 tags	61
5 tags	12
6 tags	2
7 tags	1 ("still")

Table 3.1: Number of words by degree of ambiguity

is reported that over 10% of the (unique) English vocabulary words and over 40% of the tokens (running words) in the Brown Corpus are ambiguous. Words in other languages can be more ambiguous than in English. For example, more than 29.9% of the vocabulary words in the Chinese Penn treebank (CTB) [161, 162] have more than one POS tag [151].

Many approaches have been developed in the literature for POS tagging, ranging from rule-based methods [60, 83, 154] that rely on a dictionary to provide the possible POS tags for a word and hand crafted rules to decide what categories can co-occur, to transformation-based learning methods [18] that start with some simple initial assignment of tags and then incrementally learn rules to recover errors based on the training data, and to probabilistic methods [15, 79, 131, 147, 151].

We will next briefly describe two popular types of POS taggers, HMM-based generative POS taggers and discriminative POS taggers, discuss their limitations, and present our latent variable tagger.

3.3 HMM POS Tagger

Let Σ be the set of vocabulary words and $\mathcal{X}(o)$ be the set of possible POS tags for each word $o \in \Sigma$. POS tagging using a hidden Markov model (HMM) can be considered as an instance of Bayesian inference, wherein we observe a sequence of words $o_1^n = o_1, \cdots, o_n$, and need to assign them the most likely tag sequence $\hat{x}_1^n$, i.e.,

$$\begin{aligned} \hat{x}_1^n &= \arg \max_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(X_1^n = x_1^n | O_1^n = o_1^n) \\ &= \arg \max_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(X_1^n = x_1^n, O_1^n = o_1^n) \end{aligned}$$

In a bigram HMM tagger, the joint probability of (o_1^n, x_1^n) is factored as:

$$\begin{aligned} \mathrm{P}(O_1^n = o_1^n, X_1^n = x_1^n) &= \prod_{i=1}^{n} \mathrm{P}(X_i | X_1^{i-1}, O_1^{i-1}) \mathrm{P}(O_i | X_1^i, O_1^{i-1}) \\ &= \prod_{i=1}^{n} \mathrm{P}(X_i | X_{i-1}) \mathrm{P}(O_i | X_i) \end{aligned}$$

The tagging performance of a bigram tagger is limited by its overly strong first-order hidden Markov assumption. A common approach to address this problem is to use a higher-order hidden Markov assumption. For example, the TnT tagger [15] employs trigram transition probabilities and bigram emission probabilities[11]:

$$\mathrm{P}(O_1^n, X_1^n) = \prod_{i=1}^{n} \mathrm{P}(X_i | X_{i-1}, X_{i-2}) \mathrm{P}(O_i | X_i) \tag{3.1}$$

[11] Equation 3.1 is slightly different from the original TnT tagger, which used a bigram transition at the right boundary. We assume $X_{-1} = \text{SSOS\_STATE}$.

and the second-order HMM tagger developed by Thede and Harper [147] employs both trigram transition and emission probabilities:

$$\mathrm{P}(O_1^n, X_1^n) = \prod_{i=1}^{n} \mathrm{P}(X_i|X_{i-1}, X_{i-2})\mathrm{P}(O_i|X_{i-1}, X_i) \tag{3.2}$$

In supervised tagging tasks, a corpus $\mathcal{T}$ of POS-tagged sentences are given as the training data and the parameters of HMM taggers can be simply estimated by normalizing observed counts on the training data according to the maximum likelihood criterion. In the case of the bigram tagger in Equation 3.1, the maximum-likelihood estimates are:

$$\mathrm{P}_t(x|x') = \frac{c(x, x')}{c(x')} \tag{3.3}$$

$$\mathrm{P}_e(o|x) = \frac{c(o, x)}{c(x)} \tag{3.4}$$

where $c(\cdot, \cdot)$ and $c(\cdot)$ denote event counts in the training data. For example,

$$c(x, x') = \sum_{(x_1^n, o_1^n) \in \mathcal{T}} \sum_{i=1}^{n} \delta(x_{i-1}, x')\delta(x_i, x) \tag{3.5}$$

in which $\delta(\cdot, \cdot)$ returns 1 if the two operands are identical and 0 otherwise.

It is well know that maximum likelihood estimation suffers from the data sparsity problem. The conditional probabilities are usually over-estimated for the events observed in the training data and are under-estimated, actually set to zero, for the events that are not observed in the training data but could occur in a test

sentence. Data sparsity becomes more severe for higher-order models, and so it is essential to utilize effective smoothing techniques. For example, the trigram tagger of Thede and Harper [147] uses a log-based smoothing method to smooth transition (and similarly emission) probabilities:

$$\begin{aligned} \mathrm{P}_{\mathrm{Thede}}(X_i|X_{i-1}, X_{i-2}) &= f(c(x_{i-2}^i))\mathrm{P}(X_i|X_{i-1}, X_{i-2}) \\ &\quad +(1 - f(c(x_{i-2}^i))\mathrm{P}_{\mathrm{Thede}}(X_i|X_{i-1}) \end{aligned} \tag{3.6}$$

with

$$f(x) = \frac{\log_a(x+1) + b}{\log_a(x+1) + (b+1)} \tag{3.7}$$

where a and b are two parameters that need to be tuned.

The smoothing method in Equation 3.6 does not handle out-of-vocabulary (OOV) words and would still assign a zero probability to them. A simple approach to model OOV words is to uniformly estimate the emission probabilities of those unseen words based on the statistics of rare words in the training data; however, this totally ignores the morphological properties of words that could otherwise be informative of the POS types of OOV words, especially for inflected languages [138]. As reported in (Brants [15]), 98% of the words in the Wall Street Journal (WSJ) part of the Penn treebank ending in *able* are adjectives (e.g. fashionable, variable) while the remaining 2% are nouns (e.g. cable, variable). The English trigram tagger in (Thede and Harper [147]) combines suffix and three binary features, i.e., whether

the word is capitalized, whether the word is hyphenated, and whether the word contains numbers, into a signature of the word and uses it to estimate the emission probabilities of OOV words. Note that OOV words from different languages have different characteristics and thus should be handled differently. For example, the word formation process for Chinese words is very different from English words [120]. Indeed, the last characters in a Chinese word are, in some cases, most informative of the POS type, while for others, it is the characters at the beginning. Furthermore, it is not uncommon for a character in the middle of a word to provide some evidence for the POS type of the word. In (Huang et al. [72]), we exploited this Chinese-specific characteristic to estimate the emission probability of an OOV word based on all of the characters and achieved improved tagging accuracy on Chinese.

3.4 Discriminative POS Taggers

Discriminative POS taggers model the conditional probability $\mathrm{P}(X_1^n|O_1^n)$ instead of the joint probability $\mathrm{P}(O_1^n, X_1^n)$ as in HMM models. In maximum-entropy based models [131], the conditional probability is typically approximated as follows:

$$\begin{aligned} \mathrm{P}(X_1^n|O_1^n) &= \prod_{i=1}^{n} \mathrm{P}(X_i|X_1^{i-1}, O_1^n) \\ &\approx \prod_{i=1}^{n} \mathrm{P}(X_i|H_i) \end{aligned} \tag{3.8}$$

where H_i represents the approximate history. The conditional probability $\mathrm{P}(X_i =$

Condition	**Features**	
o_i is not rare	$o_i = o$	& $x_i = x$
o_i is rare	p is the prefix of o_i, $\|p\| \leq 4$	& $x_i = x$
	s is the suffix of o_i, $\|s\| \leq 4$	& $x_i = x$
	o_i contains number	& $x_i = x$
	o_i contains uppercase character	& $x_i = x$
	o_i contains hyphen	& $x_i = x$
$\forall o_i$	$x_{i-1} = x'$	& $x_i = x$
	$x_{i-2} = x^*$, $x_{i-1} = x'$	& $x_i = x$
	$o_{i-1} = o$	& $x_i = x$
	$o_{i-2} = o$	& $x_i = x$
	$o_{i+1} = o$	& $x_i = x$
	$o_{i+2} = o$	& $x_i = x$

Table 3.2: Feature templates

$x|H_i = h)$ is modeled by a log-linear model of k features:

$$\mathrm{P}(X_i = x|H_i = h) = p(x|h) = \frac{\exp(\sum_{j=1}^{k} \lambda_j f_j(x,h))}{\sum_{x' \in \mathcal{X}} \exp(\sum_{j=1}^{k} \lambda_j f_j(x',h))} \tag{3.9}$$

where $\mathcal{X}$ is the tag inventory, f_i is a typically a binary feature function, and λ_i is the corresponding feature weight. Table 3.2 lists the feature templates used in (Ratnaparkhi [131]). The benefit of modeling conditional probabilities with log-linear models is that one can encode any overlapping information about the pair of (h, x) as features to help predict POS tags. For example, Toutanova and Manning [148] improved the model of Ratnaparkhi [131] by incorporating features that better handle capitalization for OOV words, disambiguate the tense forms of verbs, and discriminate particles from prepositions and adverbs.

The feature weights $\lambda_1, \cdots, \lambda_k$ of the probability distributions $\hat{p}$ that best fit

the training data can be obtained by maximum likelihood estimation:

$$\begin{aligned} Q = \{p | p(x|h) &= \frac{\exp(\sum_{j=1}^{k} \lambda_j f_j(x,h))}{\sum_{x' \in \mathcal{X}} \exp(\sum_{j=1}^{k} \lambda_j f_j(x',h))}\} \\ L(p) &= \sum_{x,h} \tilde{p}(x,h) \log p(x|h) \\ \hat{p} &= \arg\max_{p \in Q} L(p) \end{aligned} \tag{3.10}$$

in which $\tilde{p}$ is the empirical distribution of (x, h) on the training data. Berger et al. [9] showed that this is equivalent to maximizing the conditional entropy:

$$H(p) = -\sum_{x,h} \tilde{p}(h) p(x|h) \log p(x|h) \tag{3.11}$$

under the following constraints:

$$\sum_{x,h} \tilde{p}(x,h) f_j(x,h) = \sum_{x,h} \tilde{p}(h) p(x|h) f_j(x,h) \tag{3.12}$$

Maximum entropy models suffer from the label bias problem [14] that results from the fact that the scores of transitions going out of a state are locally normalized, causing a bias toward states with fewer outgoing transitions. Conditional random field (CRF) models [79] avoid this bias by normalizing the scores of the entire state sequence instead of each single state:

$$\mathrm{P}(X_1^n = x_1^n | O_1^n = o_1^n) = \frac{\exp\left(\sum_{i=1}^{n} \sum_{j=1}^{k} \lambda_j f_j(x_{i-1}, x_i, o_1^n)\right)}{Z(o_1^n)} \tag{3.13}$$

where $f_j(x_{i-1}, x_i, o_1^n)$ represents a feature involving states and observations and

$Z(o_1^n)$ is the partition function to ensure that $\mathrm{P}(X_1^n = x_1^n | O_1^n = o_1^n)$ is a proper distribution over all possible x_1^n sequences given the sentence.

Despite the fact that discriminative models have the flexibility to incorporate overlapping features, including long distance features that relate non-local observations to local states, they are still subject to strong independence assumptions among the states in order to keep model inference tractable. For example, the feature function $f_j(x_{i-1}, x_i, o_1^n)$ of the CRF model in Equation 3.13 is only able to capture dependencies among neighboring states. Although approaches like discriminative reranking [72] are able to use long distance features, they are still built upon base models that suffer from using only local constraints.

3.5 Latent Bigram POS Tagger

In POS tagging, we are interested in modeling the dependencies among the word/tag sequence (o_1^n, x_1^n), not the word sequence o_1^n alone. When modeled by a first-order HMM, the generation process of the word/tag pairs is Markovian and is only able to capture dependencies among the neighboring pairs, as we have shown in Section 2.4:

$$\mathrm{P}(O_i, X_i | O_1^{i-1}, X_1^{i-1}) \;=\; \mathrm{P}(O_i, X_i | O_{i-1}, X_{i-1}) \tag{3.14}$$

The traditional approach to capture longer dependencies is to use a higher-order hidden Markov assumption; however, higher-order taggers suffer from more severe data sparsity issues and require more sophisticated smoothing methods to

make them effective, which itself is a challenging task. Using a fixed order of independence assumption is also not likely to be optimal because it may be too strong for some parts of the model where the training instances are sufficient but too weak for other parts of the model where the training instances are sparse.

In order to effectively model dependencies among words and tags, we develop a *latent bigram tagger* that splits each POS tag x into a set of fine-grained latent tags $\mathcal{Z}(x)$ and model the word generation process using a latent hidden Markov model described in Section 2.4. In this model, the joint process that generates the word/tag pairs is no longer Markovian. Instead, all word/tag pairs are now dependent on each other:

$$\begin{aligned}
&\mathrm{P}(O_i, X_i | O_1^{i-1}, X_1^{i-1}) \\
&= \sum_{\substack{z_{i-1} \in \mathcal{Z}(x_{i-1}) \\ z_i \in \mathcal{Z}(x_i)}} \mathrm{P}(O_i, X_i, Z_i = z_i, Z_{i-1} = z_{i-1} | O_1^{i-1}, X_1^{i-1}) \\
&= \sum_{\substack{z_{i-1} \in \mathcal{Z}(x_{i-1}) \\ z_i \in \mathcal{Z}(x_i)}} \mathrm{P}(Z_{i-1} = z_{i-1} | O_1^{i-1}, X_1^{i-1}) \mathrm{P}_t(z_i | z_{i-1}) \mathrm{P}_e(o_i | z_i) \qquad (3.15)
\end{aligned}$$

and the impact of the complete history (o_1^{i-1}, x_1^{i-1}) on the distribution of (o_i, x_i) can be represented by the conditional distribution $\mathrm{P}(Z_{i-1} = z_{i-1} | O_1^{i-1}, X_1^{i-1})$ of the latent tag z_{i-1}. Note that if x_{i-1} has only one latent tag, $\mathrm{P}(Z_{i-1} = z_{i-1} | O_1^{i-1}, X_1^{i-1})$ would be consistently 1 for the only latent tag, and thus it would not be able to capture any history information in (o_1^{i-1}, x_1^{i-1}). The number of latent tags split from each POS tag directly affects the latent bigram tagger's capability to capture complex dependencies; however, it is also not desirable to excessively split the POS

tags because this would then cause severe data sparsity problems (and over-fitting).

Ideally, we would like to allocate more latent tags to POS tags that have more context variation in order to better capture those contexts. Figure 3.1 illustrates the number (in logarithmic scale) of tokens and types (i.e., unique words) associated with each POS tag in sections 0-18 of the WSJ Penn treebank [104]. In general, frequent tags and tags associated with many unique words tend to appear in a variety of different contexts and thus should receive more latent tags than others. However, this is not always true. It is easy to observe that the tag frequency follows the Zipf's Law [101]. However, determining the number of latent tags based on the tag frequency is not an appropriate strategy because a frequent tag may only appear in a few simple contexts. For example, a period (.) always appears at the end of a sentence, and thus there is no need to use more than a few latent tags to capture the limited contextual variation. Solely relying on the number of unique words associated with each POS tag is also not a reliable strategy for determining the number of latent tags because a POS tag associated with only a small number of frequent words (e.g., comma (,) or determiner (DT)) can appear in very different contexts and thus would benefit from using relatively more latent tags.

The data-driven split-merge latent HMM training algorithm described in Section 2.4.2.3 is able to adaptively allocate latent tags to cases that are most beneficial at increasing the training likelihood and so is used to train our latent bigram tagger. As shown in Figure 3.1, the POS tags for nouns (NN, NNS, NNP, and NNPS)[12],

[12]NN stands for singular or mass nouns, NNS for plural nouns, NNP for proper nouns, and NNPS for proper plural nouns.

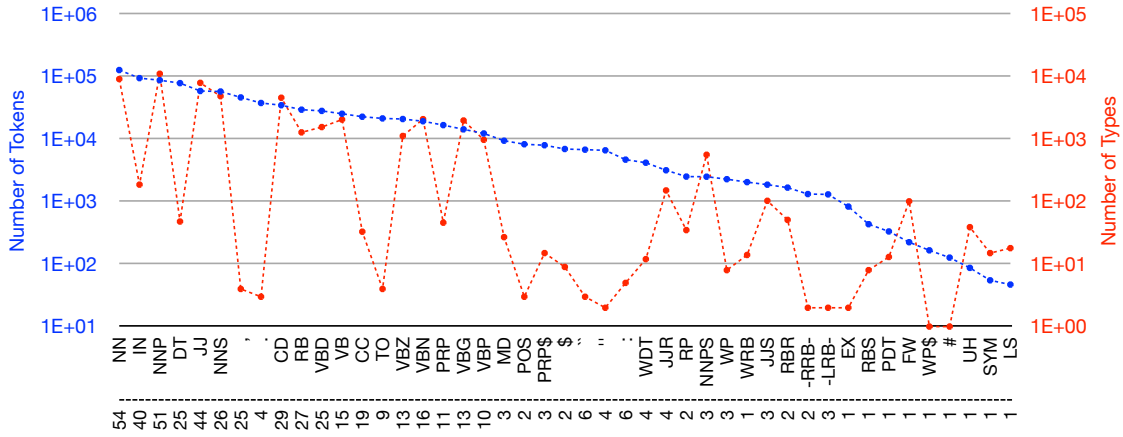


Figure 3.1: The number (in logarithmic scale) of tokens (blue) and types (red) for each POS tag in the WSJ Penn treebank training set. The number of latent tags induced by the split-merge training algorithm is also displayed below each POS tag.

verbs (VB, VBD, VBG, VBN, VBP, and VBZ)[13], adjectives (JJ), prepositions (IN), and determiners (DT) receive the most latent tags, while tags for interjections (UH), symbol (SYM), and some others only receive one latent tag. This is consistent with our discussion about which POS tags should be assigned the most latent tags and which ones should not. For example, period (.) is the 8-th most frequent POS tag in the figure but it receives only 4 latent tags, much fewer than the less frequent POS tags such as the verb tags.

In our experiments, we always run 50 iterations of EM training after each splitting operation and 20 iterations of EM training after each merging operation, followed by 10 additional iterations of EM with smoothing, as described in the next subsection.

[13]VB stands for the base form, VBD for past tense, VBG for gerund or present participle, VBN for past participle, VBP for non-3rd person singular present, and VBZ for 3rd person singular present.

3.5.1 Smoothing

As more latent tags are allocated using the split-merge training algorithm, the modeling power of a latent bigram tagger increases. However, this would eventually lead to over-fitting as EM is guaranteed to increase the training likelihood at each iteration. In order to alleviate the over-fitting problem and support more latent tags, we follow the approach introduced in (Petrov et al. [126]) to smooth the latent tags split from the same POS tag. The smoothed emission probabilities $\mathrm{P}'_e(o|z)$ and transition probabilities $\mathrm{P}'_t(z'|z)$ for all $z \in \mathcal{Z}(x)$ of POS tag x are computed as follows:

$$\begin{aligned}
\bar{P}_e &= \frac{1}{|\mathcal{Z}(x)|} \sum_{z^* \in \mathcal{Z}(x)} \mathrm{P}(o|z^*) \\
\mathrm{P}'_e(o|z) &= \epsilon_1 \bar{P}_e + (1-\epsilon_1)\mathrm{P}_e(o|z) \\
\bar{P}_t &= \frac{1}{|\mathcal{Z}(x)|} \sum_{z^* \in \mathcal{Z}(x)} \mathrm{P}(z'|z^*) \\
\mathrm{P}'_t(z'|z) &= \epsilon_2 \bar{P}_t + (1-\epsilon_2)\mathrm{P}_t(z'|z)
\end{aligned}$$

where the smoothing parameters ϵ_1 and ϵ_2 are tuned on the development set.

The above smoothing approach can alleviate the data sparsity problem that arises when fine-grained latent tags are introduced based on the maximum-likelihood criterion; however, it is not sufficient to address the sparsity issue associated with rare words because their co-occurrence with latent tags is much harder to estimate reliably. To handle this problem, we tie the emission probabilities $\mathrm{P}_e(o|z)$ of words whose frequency is less than threshold τ into a single parameter $\mathrm{P}_e(\mathit{unk}|z)$ in pro-

portion to the co-occurrence frequency of the word o and the corresponding POS tag $x = \mathcal{Z}^{-1}(z)$, i.e.,

$$\mathrm{P}_e(o|z) \;=\; \frac{c(x,o)}{\sum_{o':c(o')<\tau} c(x,o')}\mathrm{P}_e(unk|z)$$

and tune $\mathrm{P}_e(unk|z)$ as a parameter in EM training.

3.5.2 OOV Handling

While the training algorithm to induce latent tags is language independent, language dependent methods are needed to effectively handle OOV words. We experiment with two languages, English and Chinese. For English, we estimate the emission probability of an OOV word based on its suffixes, as well as several other features such as whether the word is capitalized, whether the word is hyphenated, and whether the word contains numbers [147]. However, this approach is not effective to handle Chinese OOV words for the reasons we discussed in Section 3.3. Following (Huang et al. [72]), we use the geometric average of the emission probabilities of all of the characters in an OOV Chinese word to estimate the emission probability of the OOV word:

$$\mathrm{P}(w|t_x) = \sqrt[n]{\prod\nolimits_{c_k \in w, \mathrm{P}(c_k|t)\neq 0} \mathrm{P}(c_k|t)}$$

where $n = |\{c_k \in w|\mathrm{P}(c_k|t) \neq 0\}|$. Characters not seen in the training data are ignored in the computation of the geometric average. We back off to use the rare

word statistics regardless of word identity when the above equation cannot be used to compute the emission probability.

3.5.3 Decoding

As we discussed in Section 2.4.2.2, retrieving the most-likely POS sequence given a test sentence is an intractable task for a latent bigram tagger, and thus we must use approximate solutions or optimize on a different criterion. In our experiments, we use the MostProb-T-P method of Equation 2.18 to return the POS sequence that maximizes the product of the posterior probabilities of transitions because it performs better than the other approaches.

$$\hat{x}_1^n \quad = \quad \arg\max_{x_1^n} \prod_{i=1}^{n} \mathrm{P}(X_{i-1} = x_{i-1}, X_i = x_i | O_1^n = o_1^n)$$

3.6 Self-Training

It is always desirable to have more high quality training data to train more accurate taggers; however, gold standard hand-labeled training data is often quite limited, because human annotation is both expensive and time-consuming. Recognizing that unlabeled data is often ubiquitous and can be obtained in large quantities at a low cost, semi-supervised learning methods such as self-training are naturally exploited for improving tagging performance. Self-training, as shown in Figure 3.2, proceeds as follows: we first train an initial model on some gold standard training data, then use that model to label a large amount of unlabeled data, and finally train

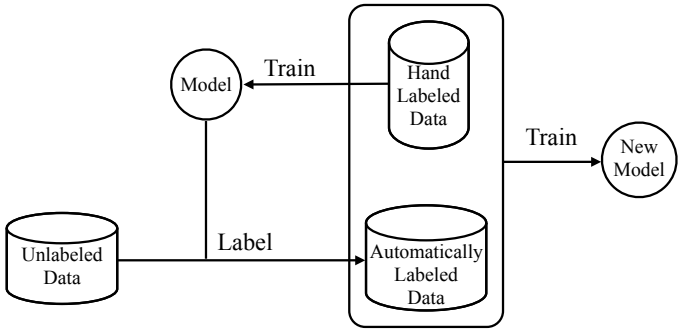


Figure 3.2: Self-Training

a new model on the combination of gold standard training data and automatically labeled training data.

Early investigations of self-training on POS tagging have mixed outcomes. Clark et al. [45] reported positive results with a small amount of gold standard training data but negative results when the amount of gold standard training data increases. Wang et al. [156] also reported improvement from self-training using a trigram tagger with a small amount of gold standard data, but as we will show in our experiments, the same tagger does not benefit from self-training when more gold standard training data is available. The value of the latent variable approach for tagging is that it can learn more fine grained tags to better model the training data. Liang and Klein [94] analyzed errors of unsupervised learning using EM and found that both estimation and optimization errors decrease as the amount of unlabeled data increases. In our case, the learning of latent annotations through EM may also benefit from a large set of automatically labeled data to improve tagging performance. We will investigate self-training of the latent bigram tagger and compare it with conventional bigram and trigram HMM taggers.

We next discuss two decisions associated with self-training. There are several

ways to automatically label the unlabeled data for self-training. A fairly standard method is to tag the unlabeled sentences with a tagger trained on gold standard training data, and then combine the automatically labeled data with the gold standard data to re-train the tagger. This is the approach we choose for self-training. An alternative approach is to use the latent bigram tagger trained on the gold standard training set as the initial model and perform unsupervised training on the unlabeled training data. The problem is that this would slow down the training process because the POS tags would need to be determined in addition to the latent POS tags at each EM iteration.

Another important decision is how to combine the gold standard and automatically labeled data when training a new tagger. Errors in the automatically labeled data could limit the accuracy of the self-trained model, especially when there is a much greater quantity of automatically labeled data than the gold standard training data. To balance the gold standard and automatically labeled data, one could duplicate the gold standard training data to match the amount of automatically labeled data, which is what is done to self-train the conventional bigram and trigram HMM taggers. In order to avoid redundant EM computations over the same data when self-training the latent bigram tagger, we weight the posterior probabilities computed for the gold and automatically labeled data, so that they contribute equally to the resulting model.

3.7 Experiments

3.7.1 Setup

We conduct experiments on two languages: Chinese and English. The Chinese Penn treebank 6.0 (CTB6) [162] is used as the gold standard data in our study on Chinese. CTB6 contains news articles, which are used as the primary source of gold standard training data in our experiments, as well as broadcast news transcriptions. Since the news articles were collected during different time periods from different sources with a diversity of topics, in order to obtain a representative split of train/development/test sets, we divide them into blocks of 10 files in sorted order and for each block use the first file for development, the second for test, and the remaining for training. The broadcast news data exhibits many of the characteristics of newswire text (it contains many nonverbal expressions, e.g., numbers and symbols, and is fully punctuated), and so it is also included in the training data set. We also utilize a greater number of unlabeled sentences in the self-training experiments. They are selected from sources similar to the newswire articles, and are normalized [165] and word segmented [150].

The English experiments are conducted on the WSJ Penn treebank [104]. Following the data splits in (Toutanova et al. [149]), we use sections 0-18 for training, 19-21 for development, and 22-24 for testing. We also utilize a large number of unlabeled sentences from the BLLIP corpus [35] for the self-training experiments. Table 3.3 summarizes the data used in our experiments.

We evaluate the performance of the latent bigram tagger (denoted by Bi-

	Train	Dev	Test	Unlabeled
Chinese	24.4k (678.8k)	1.9k (51.2k)	2.0k (52.9k)	210k (6,254.9k)
English	38.2k (912.3k)	5.5k (131.8k)	5.5k (129.7k)	210k (5,082.1k)

Table 3.3: The number of sentences (and tokens in parentheses) in our experiments

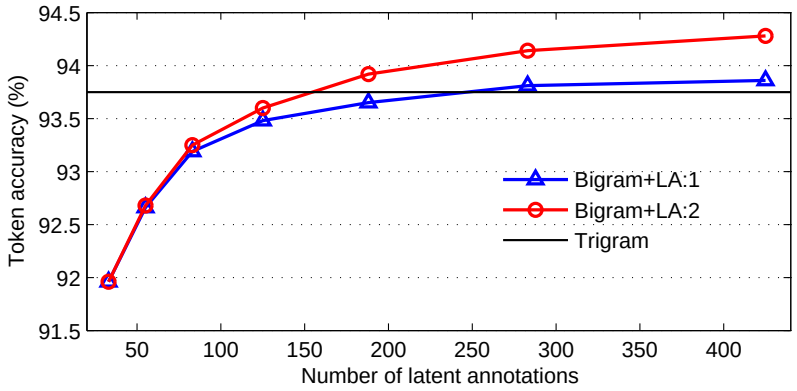


Figure 3.3: The learning curves of the latent bigram tagger on the development set

gram+LA) compare it with a conventional bigram tagger without latent variables (denoted by Bigram) and a state-of-the-art trigram HMM tagger (denoted by Trigram) [72] that uses trigram transition and emission models together with bidirectional decoding. Due to the randomness introduced in the split-merge training algorithm, different random seeds produce latent bigram taggers with slightly different tagging performance. For each setup, we train 10 latent bigram taggers with different seeds and select the tagger to evaluate based on its performance on the development set. In Chapter 6, we will investigate methods to utilize the variability among latent variable models for enhanced performance.

3.7.2 Chinese Results

Figure 3.3 plots the learning curves of two versions of the latent bigram tagger and compares them with the performance of the trigram tagger. The first version (labeled as Bigram+LA:1) does not implement the rare word smoothing method described Section 3.5.1, while the second version (labeled as Bigram+LA:2) does. Both of the latent bigram taggers initially have a much lower tagging accuracy than the trigram tagger, due to their strong but invalid independence assumption. As the number of latent annotations increases, the latent bigram taggers are able to learn more from the context based on the latent annotations and eventually outperform the trigram tagger. The performance gap between the two latent bigram taggers suggests that over-fitting occurs in the word emission model when more latent annotations are available for training and sharing the statistics among rare words alleviates the data sparsity issue. In the later experiments, we use Bigram+LA to denote the latent bigram tagger with rare word smoothing.

Figure 3.4 compares the effectiveness of self-training (ST) for three models (the latent bigram tagger and the conventional bigram and trigram taggers) when trained using different amounts of gold standard training data. The entire unlabeled data set is used for self-training. There are two interesting observations that distinguish the latent bigram tagger from the other two taggers.

First, although all of the taggers improve as more gold standard training data becomes available, the performance gap between the latent bigram tagger and the other two taggers also increases. This is because the additional training data sup-

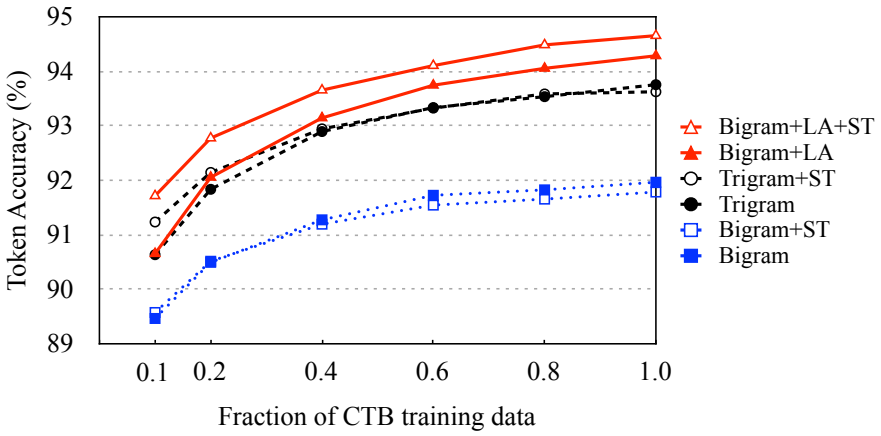


Figure 3.4: The performance of three taggers evaluated on the Chinese development set, before and after self-training with different sizes of gold standard training data

ports an increased number of latent annotations that are able to learn more complex dependencies. Second, the latent bigram tagger benefits much more from self-training. Except for a slight improvement when there is a small amount of gold standard training data, self-training hurts the performance of the conventional bigram tagger as the amount of gold standard data increases. The trigram tagger benefits from self-training initially but eventually has a similar pattern to the bigram tagger when it is trained on the entire gold standard training data. The performance of the latent bigram tagger improves consistently with self-training. Although the gain decreases for models trained on larger training sets because stronger models are harder to improve, self-training still significantly improves tagging accuracy.

Table 3.4 reports the final tagging accuracies on the CTB6 test set for the taggers trained on the entire gold standard training data. All of the improvements are statistically significant ($p < 0.005$). The latent bigram tagger is significantly more accurate than the conventional trigram tagger and it is able to further benefit

Tagger	Test Accuracy
Bigram	92.25
Trigram	93.99
Bigram+LA	94.53
Bigram+LA+ST	94.78

Table 3.4: The token accuracy (%) of the taggers on the CTB6 test set

from self-training.

It is worth mentioning that we initially added latent annotations to a trigram tagger, rather than a bigram tagger, to build from a stronger starting point; however, this did not work well. A trigram tagger requires sophisticated smoothing to handle data sparsity, and introducing latent annotations exacerbates the sparsity problem, especially for trigram word emissions. The uniform extension of a bigram tagger to a trigram tagger ignores whether the use of additional context is helpful and supported by enough data, and it is unable to capture dependencies beyond the second-order independence assumption. Furthermore, there is no mechanism to eliminate trigram contexts that are not useful. In contrast, the latent bigram tagger is able to learn different granularities for tags based on the training data.

3.7.3 English Results

Figure 3.5 compares the performance of the three English taggers using different sizes of WSJ training data before and after self-training. The general observation that the latent bigram tagger benefits more from self-training than the conventional bigram and trigram taggers is similar to that of the Chinese experiment. When us-

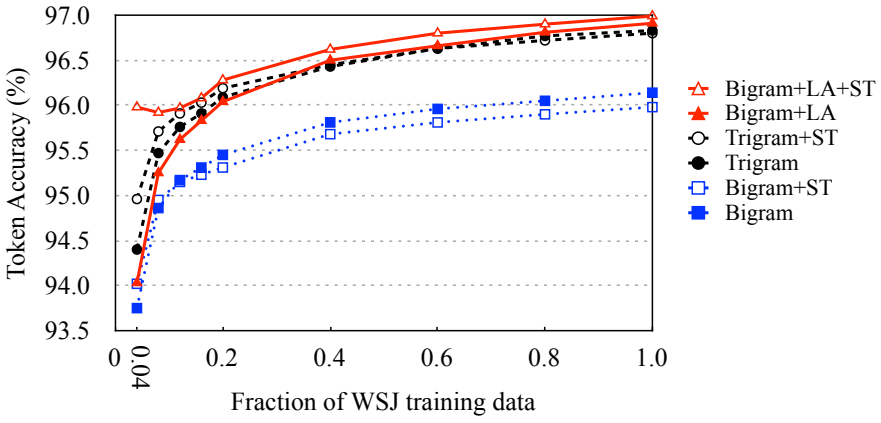


Figure 3.5: The performance of three taggers evaluated on the English development set, before and after self-training with different sizes of gold standard training data

ing only 4% of the training data, the latent bigram tagger scored at 94.03%, poorer than 94.4% obtained by the trigram tagger. This is probably because we use a simple smoothing method for the latent bigram tagger and the data is too sparse to learn fine-grained latent tags without over-fitting. However, the accuracy of the latent bigram tagger improves significantly to 95.98% after self-training, compared to only 94.96% for the self-trained trigram tagger. Note that the accuracy of the self-trained latent bigram tagger drops slightly when the size of the gold standard training data increases from 4% to 8% of the WSJ training set. This is because the performance of the latent bigram tagger is sensitive (due to over-fitting) to the number of latent tags when there is a small amount of training data and the split-merge training algorithm may not get close to the optimal number of latent tags because it increases the number of latent tags discretely[14].

Table 3.5 reports the final results of the taggers on the WSJ test set. The

[14]Recall that each split-merge iteration splits every latent tag into two and then merges half of them back.

Tagger	**4%**	**100%**
Bigram	93.83	96.13
Bigram+ST	94.09	95.94
Trigram	94.67	96.92
Trigram+ST	95.25	96.84
Bigram+LA	94.20	96.98
Bigram+LA+ST	95.99	97.05

Table 3.5: The token accuracy (%) of the taggers trained on 4% and 100% of the WSJ training set before and after self-training, evaluated on the WSJ test set

conventional bigram and trigram taggers benefit from self-training when initially trained on 4% of the gold standard training data but they are hurt by self-training when initially trained on the entire gold standard training data. In contrast, the latent bigram tagger is able to consistently benefit from self-training. It should be noted that the improvement from 96.98% to 97.05% obtained by self-training the latent bigram tagger trained on the entire gold standard training data is impressive given the fact that the inter-annotator tagging agreement rate on the WSJ Penn treebank is only around 97% [104].

3.8 Conclusions

In this chapter, we have developed and evaluated a latent bigram tagger based on the latent hidden Markov model. Our experiments on Chinese and English showed that the latent bigram tagger achieves significantly better tagging accuracies than the conventional bigram and trigram taggers and is able to consistently benefit from self-training. We will further study self-training in Chapter 5 for parsing and

analyze why it boosts the performance of latent variable models. In the next chapter, we will develop and evaluate a language model that uses the latent bigram tagger developed in this chapter.

Chapter 4

Language Modeling with Latent Variables

4.1 Overview

A statistical language model is used to determine how likely a given word sequence comes from a language. The most commonly used word N-gram language models suffer from severe data sparsity problems and require effective smoothing methods to obtain reliable parameter estimates. Class-based language models are an alternative that clusters similar words into classes to reduce data sparsity; however, clustering removes much of the lexical information and tends to results in worse performance than the word N-gram models.

We develop a latent language model that is able to capture the contextual dependencies among words using latent variables. In this model, each word is associated with a set of latent tags that are induced automatically from the training data. Different uses of a word under different contexts are represented by different latent tags and similar uses of different words are represented by their shared latent tags. The latent variable approach is able to flexibly adjust the model parameterization to learn dependencies at different levels of granularity and so is able to achieve better performance than conventional N-gram models.

The rest of the chapter is structured as follows. Section 4.2 provides some background information on language models. Sections 4.3 and 4.4 briefly review

previous work on word N-gram language models and class-based language models, respectively. Section 4.5 describes our approach to building language models with latent tags, and Section 4.6 describes a special instantiation of latent language models based on POS tags. Experimental results for the POS-based latent language model are presented in Section 4.7. The last section concludes this chapter.

4.2 Introduction to Language Modeling

A language model assigns a probability mass to any word sequence $o_1^n = o_1, \cdots, o_n$ of a language:

$$\mathrm{P}(O_1 = o_1, \cdots, O_n = o_n) \tag{4.1}$$

and it is the backbone of many applications such as speech recognition [78], machine translation [20], optical character recognition [74], spelling correction [81], handwriting recognition [142], and information retrieval [128]. A typical use of a language model in these applications is as a component in a noisy channel model. Taking speech recognition for example, the goal is to find the most likely word sequence $\hat{W}$ given acoustic input A, which can be expressed as follows:

$$\hat{W} \;=\; \arg\max_W \mathrm{P}(W|A) = \arg\max_W \frac{\mathrm{P}(W,A)}{\mathrm{P}(A)} = \arg\max_W \mathrm{P}(W)\mathrm{P}(A|W) \tag{4.2}$$

in which the conditional probability $\mathrm{P}(A|W)$ is computed by the acoustic model and the probability $\mathrm{P}(W)$ is assigned by the language model.

The joint probability of any word sequence o_1^n can be factored by the chain rule as follows:

$$\mathrm{P}(O_1^n = o_1^n) = \prod_{i=1}^{n} \mathrm{P}(O_i = o_i | O_1^{i-1} = o_1^{i-1}) \tag{4.3}$$

where each o_i in the conditional distribution is the *future* word to predict, and o_1^{i-1} is the complete *history* for estimating the distribution of o_i. The prediction of o_i is most accurate given the complete history o_1^{i-1}; however, it is impossible to reliably estimate probabilities conditioned on the complete history. This problem is usually addressed by defining some mapping function $H(\cdot)$ that maps the observed history into an equivalence class:

$$\mathrm{P}(O_1^n) \approx \prod_{i=1}^{n} \mathrm{P}(O_i | H(O_1^{i-1})) \tag{4.4}$$

Since language models are generally integrated with other models in various different applications, the best way to compare performance of different language models is to directly evaluate their performance on the application of interest. For example, recognition word error rate (WER) [114] is the performance measurement used to evaluate most modern speech recognition systems[15], and thus different language models can be compared on the WER of a speech recognition task. Alternatively, perplexity is a task-independent metric that measures a language model's ability to predict the next word, and it has been found to correlate well with in-

[15]There are alternative evaluation metrics, e.g., see (McCowan et al. [108]).

domain task performance [39]. The perplexity, $\mathrm{PPL}(o_1^n)$, of text o_1^n for a language model is calculated as:

$$\mathrm{PPL}(o_1^n) = 2^{\mathrm{Entropy}(o_1^n)} \tag{4.5}$$

$$\mathrm{Entropy}(o_1^n) = -\frac{1}{n}\log_2 \mathrm{P}(O_1^n = o_1^n) \tag{4.6}$$

where $\mathrm{Entropy}(\cdot)$ measures the average number of bits to encode a word in a text with the language model. An improved language model can better predict words in a sequence and thus can encode each word using fewer bits. It should be noted that a language model with a lower perplexity does not necessarily perform better on a specific task. As noted in (Rosenfeld [135]), a reduction of $10\% \sim 20\%$ in perplexity is noteworthy, and often (but not always) translates into some improvement in application performance.

In the next two sections, we briefly describe previous research on word N-gram language models and class-based language models, and discuss the problems associated with these two approaches to motivate our method of using latent variables. Note that there are many other language modeling approaches, and we refer readers to (Goodman [58]) for an excellent review of statistical language modeling techniques.

4.3 N-gram Language Models

In a word N-gram language model, the history equivalence classes are defined as $H(o_1^{i-1}) = o_{i-N+1}^{i-1}$, following the $(N-1)$-th order Markov assumption that the

prediction of word o_i is independent of everything else given the preceding $(N-1)$ words. Hence $\mathrm{P}(O_1^n = o_1^n)$ can be approximated as follows:

$$\mathrm{P}(O_1^n) \approx \prod_{i=1}^{n} \mathrm{P}(O_i | O_{i-N+1}^{i-1}) \tag{4.7}$$

In particular, when $N = 2$ (or $N = 3$), it corresponds to the bigram (or trigram) language model.

Let $c(\cdot)$ represent the count of an event denoted by $\cdot$ in the training data. The maximum-likelihood estimates of the conditional probability $\mathrm{P}(O_i | O_{i-N+1}^{i-1})$ of an N-gram language model can be computed based on the statistics in the training data:

$$\mathrm{P}(O_i = o_i | O_{i-N+1}^{i-1} = o_{i-N+1}^{i-1}) \quad = \quad \frac{c(o_{i-N+1}^{i})}{c(o_{i-N+1}^{i-1})} \tag{4.8}$$

Although the Markov assumption greatly reduces the amount of history to model, N-gram language models still suffer from severe data sparsity problems. Maximum likelihood estimation usually over-estimates the probabilities of N-grams observed in the training data while under-estimating the probabilities of unobserved N-grams. In particular, it assigns a zero probability to unobserved N-grams, and as a result, any word sequence that contains novel N-grams would receive a zero probability from the language model, which is undesirable in real world applications [158]. Even in bigram language models, there could be a significant number of bigrams in the test set that do not occur in the training data. It was reported

in (Rosenfeld [135]) that a third of the trigrams (i.e., consecutive word triplets) in news articles of a test set selected from the same domain as a training set with 38 million words are not observed in the training set, and furthermore that the vast majority of the observed trigrams occurred in the training set only once. The problem becomes more severe for larger N values. A common solution to this problem is to apply smoothing techniques to allocate some of the probability mass of the observed N-grams to unobserved N-grams. See (Chen and Goodman [38]) for a comprehensive review of smoothing techniques for language modeling.

In order to provide a concrete example of how smoothing works, we next describe one of the best performing smoothing methods, the modified Kneser-Ney (KN) smoothing method from (Chen and Goodman [38]). The KN-smoothed N-gram prediction probability $\mathrm{P}_{\mathrm{KN}}(O_i|O_{i-N+1}^{i-1})$ takes the following form:

$$\mathrm{P}_{\mathrm{KN}}(O_i|O_{i-N+1}^{i-1}) = \frac{c(o_{i-N+1}^{i}) - D_N(c(o_{i-N+1}^{i}))}{\sum_{o_i} c(o_{i-N+1}^{i})} + \gamma(o_{i-N+1}^{i-1})\mathrm{P}_{\mathrm{KN}}(O_i|O_{i-N+2}^{i-1}) \quad (4.9)$$

where

$$D_N(c) = \begin{cases} 0 & \text{if } c = 0 \\ D_N^1 & \text{if } c = 1 \\ D_N^2 & \text{if } c = 2 \\ D_N^{3+} & \text{if } c \geq 3 \end{cases} \quad (4.10)$$

This smoothing method subtracts a discounted value (controlled by $D_N(\cdot)$ based on N-gram counts) from the counts of the observed N-grams and recursively inter-

polates the resulting N-gram probability with lower-order smoothed probabilities. The weighting parameter $\gamma(o_{i-N+1}^{i-1})$ is chosen so that the conditional probability $\mathrm{P_{KN}}(O_i|O_{i-N+1}^{i-1})$ sums to one:

$$\gamma(o_{i-N+1}^{i-1}) = \frac{\sum_{o_i} D_n(c(o_{i-N+1}^{i}))}{\sum_{o_i} c(o_{i-N+1}^{i})} \tag{4.11}$$

Special care is also taken to estimate the smoothed unigram probability $\mathrm{P_{KN}}(O)$ based on the observation that words that occur frequently in the training data do not necessarily occur more often with a new history in the test set. For example, although the word *Francisco* occurs a lot in the training data, it usually only appears in *San Francisco* and it is less likely to appear in other contexts in the test set. Taking this into account, Kneser-Ney smoothing estimates unigram word probability based on the number of unique bigram histories in which a word occurs in the training data:

$$\mathrm{P_{KN}}(O) = \frac{c_{1+}(\cdot, o)}{\sum_{o'} c_{1+}(\cdot, o')} \tag{4.12}$$

where $c_{1+}(\cdot, o)$ denotes the number of different words that precede word o in the training data.

There are different ways to determine the discounting parameters D_N. In modified Kneser-Ney fixed smoothing, these numbers are calculated based on the

statistics from the training data as follows:

$$\begin{aligned} Y &= \frac{N_1}{N_1 + 2N_2} \\ D_1 &= 1 - 2Y\frac{N_2}{N_1} \\ D_2 &= 2 - 3Y\frac{N_3}{N_2} \\ D_{3+} &= 3 - 4Y\frac{N_4}{N_3} \end{aligned}$$

where N_i is the number of unique N-grams that occur exactly i times in the training collection. In modified Kneser-Ney held-out smoothing, these values are tuned on a held-out development set.

A problem associated with N-gram models is that given a fixed set of training data, some N-grams are abundant enough to support a higher N value to achieve more accurate prediction of words, while some other N-grams are sparse, or even unobserved, so that a smaller N value would be preferred (as in these smoothing approaches) in order to retain reliable probability estimation. A fixed N-gram model does not have the flexibility to model dependencies with different granularities. The common practice is to choose a reasonably large value of N depending on the amount of training data and rely on smoothing techniques to handle the sparse N-grams. Other approaches include variable-length N-gram models [84, 116] and class-based language models that we describe next.

4.4 Class-based Language Models

N-gram language models directly model dependencies among individual words. An alternative approach is to cluster words into classes, and then model dependencies among words and classes. For example, *Thursday* and *Friday* tend to appear in similar contexts, and thus clustering them together to one class would enable the use of observed patterns of one word in the training data to help model novel but similar contexts of the other word in a test sentence. The class-based language model defined in (Brown et al. [21]) has the following form:

$$\begin{aligned} \mathrm{P}(O_i|O_1^{i-1}) &= \mathrm{P}(O_i|\mathcal{X}(o_i))\mathrm{P}(\mathcal{X}(o_i)|O_1^{i-1}) \\ &\approx \mathrm{P}(O_i|\mathcal{X}(o_i))\mathrm{P}(\mathcal{X}(o_i)|H(O_1^{i-1})) \\ &= \mathrm{P}(O_i|X_i)\mathrm{P}(X_i|X_{i-N+1}^{i-1}) \end{aligned} \tag{4.13}$$

where $\mathcal{X}(\cdot)$ is a many-to-one mapping that maps each word o_i to a class x_i, and the word history equivalence function $H(\cdot)$ maps the complete history to the preceding $(N-1)$ classes. If a word can have multiple classes, i.e, $\mathcal{X}(\cdot)$ maps a word to a set of classes, the generation of the word sequence o_1^n can be viewed as a hidden Markov process:

$$\mathrm{P}(O_1^n) \approx \sum_{\substack{1\le k\le n \\ x_k\in\mathcal{X}(o_k)}} \prod_{i=1}^{n} \mathrm{P}(O_i|X_i)\mathrm{P}(X_i|X_{i-N+1}^{i-1}) \tag{4.14}$$

When $N = 2$, this reduces to a first-order HMM described in Section 2.2. The

conditional probability, $\mathrm{P}(O_i|O_1^{i-1})$, of word o_i given the complete history o_1^{i-1}, can then be expressed as:

$$\begin{aligned}\mathrm{P}(O_i|O_1^{i-1}) &= \sum_{\substack{x_{i-1}\in\mathcal{X}(o_{i-1})\\ x_i\in\mathcal{X}(o_i)}} \mathrm{P}(O_i, X_i = x_i, X_{i-1} = x_{i-1}|O_1^{i-1}) \\ &= \sum_{\substack{x_{i-1}\in\mathcal{X}(o_{i-1})\\ x_i\in\mathcal{X}(o_i)}} \mathrm{P}(X_{i-1} = x_{i-1}|O_1^{i-1})\mathrm{P}_t(x_i|x_{i-1})\mathrm{P}_e(o_i|x_i) \quad (4.15)\end{aligned}$$

where the impact of the complete history o_1^{i-1} on the prediction of word o_i is represented by the conditional distribution $\mathrm{P}(X_{i-1} = x_{i-1}|O_1^{i-1})$ of x_{i-1}.

Class-based language models use a smaller number of contexts and are less affected by sparsity. The classes can be induced automatically based on statistical distributional similarities as in (Brown et al. [21]) or chosen to represent linguistic categories such as POS tags [4]. However, in spite of the benefits of class-based language models, their performance has often not been on par with word N-gram language models. Compared to a word trigram language model, Brown et al. [21] reported an 11% relative increase on perplexity using a class-based trigram language model with automatically induced classes. Bangalore [4] reported a 24.5% relative increase on perplexity using a POS-based trigram language model when compared to a word trigram language model. In general, these models must be combined with word N-gram models to achieve perplexity reduction.

The number of classes is very important for class-based language models. Ney et al. [115] found in an experiment with statistically induced word classes that, as the number of classes increases, the perplexity on the test set decreases initially but

then increases. There is a trade-off between generalizability and discriminability as the number of classes increases. Similar to word N-gram models, the discriminative power of class-based language models increases by using higher-order N-grams, but they usually suffer less from data sparsity than their word N-gram counterparts.

Niesler and Woodland [116] obtained a lower perplexity with a trigram POS-based language model than a bigram POS-based language model, and an even lower perplexity when using a variable length POS-based language model with a proper smoothing method. However, their best performing POS-based model still had an 11.3% higher perplexity than a word trigram model. This suggests that word classes such as POS tags over-generalize words and mask much of the lexical information that is often helpful for predicting the next word. Recognizing this problem, Heeman and Allen [66] proposed to add word context back into the POS-based language model:

$$\begin{aligned} \mathrm{P}(O_1^n, X_1^n) &= \prod_{i=1}^{n} \mathrm{P}(O_i, X_i | O_1^{i-1}, X_1^{i-1}) \\ &= \prod_{i=1}^{n} \mathrm{P}(O_i | O_1^{i-1}, X_1^{i}) \mathrm{P}(X_i | O_1^{i-1}, X_1^{i-1}) \\ &\approx \prod_{i=1}^{n} \mathrm{P}(O_i | O_{i-N+1}^{i-1}, X_{i-N+1}^{i}) \mathrm{P}(X_i | O_{i-N+1}^{i-1}, X_{i-N+1}^{i-1}) \end{aligned} \quad (4.16)$$

where they model the joint distribution of words and their classes based on contexts involving both words and classes. The probability of the complete sentence can then

be obtained by summing over all possible classes for each word:

$$\mathrm{P}(O_1^n) = \sum_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \mathrm{P}(O_1^n, X_1^n) \tag{4.17}$$

Significant improvements have been reported using this richer representation of context. Using a trigram joint language model, Heeman and Allen [66] achieved a 44.5% relative reduction in perplexity over a trigram POS-based language model, and Heeman [65] achieved a 14.5% relative reduction in perplexity over a word trigram language model. It should be noted that the enrichment of context in a joint language model imposes an even greater data sparsity problem, and so these successful investigations all used decision tree based approaches to effectively cluster histories.

It is important to note that the use of stochastic classes in Equation 4.14 significantly differs from the use of deterministic classes in Equation 4.13. Using deterministic classes ignores the fact that a word can have multiple distinctive usages. For instance, it might be beneficial to cluster *can* together with *might* or *could* when it is used as a modal word, but with *bottle* or *cup* when it is used as a noun; however, such a difference cannot be captured by deterministic classes. In a bigram class-based language model, the transition $\mathrm{P}(X_i|X_{i-1})$ in Equation 4.13 only tells us that the current word deterministically belongs to class x_i and the model is unable to capture anything from non-adjacent history o_1^{i-2}. In contrast, the assignment of classes in Equation 4.13 is stochastic, and we show in Equation 4.15 that the distribution of x_{i-1} depends on the complete word history o_1^{i-1}. A different history

would result in a different distribution of x_{i-1}. For example, suppose o_{i-1} is *can* and it occurs in the context of a noun, the distribution of x_{i-1} would be skewed toward a noun rather than a modal verb, abstracting the history into a compact form that is helpful for predicting o_i.

Class-based language models utilizing stochastic classes often rely on POS tags, which model different syntactic uses of words but not in a fine-grained manner. A word can have different senses with the same POS. For example, *bank* can be a financial institution or the raised ground bordering a lake or a river; it is a noun in both cases. Different histories leading to different uses of a word cannot be differentiated by POS tags when these usages share the same POS tag. Moreover, even if the same word is used with the same sense, it would still be beneficial to model the contexts in a fine-grained manner to improve word prediction. The joint language model in Equation 4.16 is able to better predict words by adding lexical cues; however, it is sill limited by the Markov assumptions and the fact that higher order models have increased data sparsity.

4.5 Latent Language Model

We develop a language model with latent variables that is able to capture clustering information among words, as well as distinctive uses of words in different contexts. We call it a *latent language model* or an LLM for short. On the one hand, words with similar usages are clustered into the same class so that data sparsity can be controlled. On the other hand, different classes are introduced for words that

have multiple usages so that it reduces the need for using higher order models to capture contextual dependency. We focus on the conceptual ideas related to learning latent classes in this section and will present a practical latent language model in the next section. Our model is essentially a class-based first-order HMM model:

$$\mathrm{P}(O_1^n) \approx \sum_{\substack{1 \le k \le n \\ x_k \in \mathcal{X}(o_k)}} \prod_{i=1}^{n} \mathrm{P}(O_i|X_i)\mathrm{P}(X_i|X_{i-1}) \tag{4.18}$$

This model differs from the traditional class-based language models in that the classes are induced automatically on the training data. Rather than using higher order models to capture more complex dependencies, this model simply induces more fine-grained classes. We also call the classes in this model states in order to emphasize that this model is essentially an HMM. The set of states can be induced using the following two operations given some initial states:

Splitting: A state occurring in a variety of different contexts may be split into several sub-states. For example, a state corresponding to a set of nouns may be split into two states, one focusing on plural nouns and another on singular nouns. Similarly, a state that often follows one state whose emitted words are mostly nouns and another state whose emitted words are mostly verbs may also be split into two states, one for each type of word distribution for the preceding states.

Merging: States that share similar contexts may be combined together. For example, two states with similar distributions of emitted words $\mathrm{P}(O_i|X_i)$ and

$$
\begin{array}{l}
\cdots\ A\ X\ R\ \cdots \\
\cdots\ B\ X\ S\ \cdots \\
\cdots\ A\ Y\ R\ \cdots \\
\cdots\ A\ Y\ T\ \cdots
\end{array}
\quad\Longrightarrow\quad
\begin{array}{l}
\cdots\ A_1\ X_1\ R_1\ \cdots \\
\cdots\ B_1\ X_2\ S_1\ \cdots \\
\cdots\ A_1\ Y_1\ R_1\ \cdots \\
\cdots\ A_1\ Y_1\ T_1\ \cdots
\end{array}
\quad\Longrightarrow\quad
\begin{array}{l}
\cdots\ A_1\ XY_1\ R_1\ \cdots \\
\cdots\ B_1\ X_2\ S_1\ \cdots \\
\cdots\ A_1\ XY_1\ R_1\ \cdots \\
\cdots\ A_1\ XY_1\ T_1\ \cdots
\end{array}
$$

Figure 4.1: An example of learning contexts using latent tags

similar distributions of following states $\mathrm{P}(X_{i+1}|X_i)$ may be merged together.

Suppose each word initially has a distinct state. The splitting operation splits the state of words with complex contexts into multiple sub-states, each modeling a certain type of context that enables non-adjacent words to impact word prediction. In contrast, the merging operation merges states of words with similar patterns of contexts together so that the associated parameters can be estimated more reliably.

Figure 4.1 illustrates how a latent language model learns from contexts. On the left are samples of word sequences involving words A, B, R, S, T, X and Y. Suppose these word sequences occur equally frequently. A word bigram language model cannot capture any dependency between A and R in word sequence "$\cdots\ A\ X\ R\ \cdots$". Whether the word before X is A or B does not affect the distribution of words after X because the prediction of words after X is independent of everything else given X in a word bigram model. This problem can be addressed by introducing states to the words. In the middle of the table, we assign a unique state W_1 to each word $W \in \{A, B, R, S, T, Y\}$ and assign two states X_1 and X_2 to word X, one for each context. In this latent language model, the presence of word A (or B) before word X can influence whether the word following X is R (or S) because only X_1 (or X_2) can follow A_1 (or B_1) and only R_1 (or S_1) can follow X_1 (or X_2). Observing that

states X_1 and Y_1 occur in similar contexts, i.e., preceded by word A and followed by word R, it might be beneficial to merge these two states together as a new state XY_1, as on the right side of table. Merging states X_1 and Y_1 results in a more robust model that is able to assign a non-zero probability to a novel word sequence "$\cdots\ A\ X\ T\ \cdots$", which would otherwise receive a zero probability from the word bigram model without smoothing.

The above example is illustrative of what latent tags can do, but in order to achieve these benefits we need to solve the following two problems:

1. How can we induce the states by splitting and merging?

2. How can we learn emission and transition parameters associated with the states?

If all of the words are initialized to share a single state, then the splitting and merging operations described in Section 2.4 for hidden HMMs can be employed to induce the states for latent language models. If each word is initialized with a unique state, the same splitting operation can still be employed to split states. However, in order to merge states of different words, we need to employ a generalized version of the merging operation to merge states of different words together. Both approaches are totally data driven but very computationally expensive. The initial state of the first approach is very coarse and requires extensive splitting, while the initial states of the second approach are very specific and require extensive merging. In the next section, we exploit POS tags to provide some linguistic guidance for inducing the states.

4.6 POS-based Latent Language Model

In this section, we describe a POS-based latent language model, or POS-LLM for short, that takes POS tags as the initial classes and learns fine-gained latent POS tags based on a POS-annotated training corpus. Each training sentence o_1^n is accompanied with a POS tag sequence x_1^n. The joint probability of (o_1^n, x_1^n) according to a POS-based bigram language model is:

$$\mathrm{P}(O_1^n, X_1^n) = \prod_{i=1}^{n} \mathrm{P}(O_i|X_i)p(X_i|X_{i-1}) \tag{4.19}$$

and accordingly, the probability of the word sequence o_1^n is:

$$\mathrm{P}(O_1^n) = \sum_{\substack{1\leq k\leq n \\ x_k\in\mathcal{X}(o_k)}} \prod_{i=1}^{n} p(O_i|X_i)p(X_i|X_{i-1}) \tag{4.20}$$

where $\mathcal{X}(o)$ denotes the set of POS tags associated with word o.

Our goal is to learn a set of latent tags $\mathcal{Z}(x)$ for each POS tag x such that each latent tag $z \in \mathcal{Z}(x)$ is a refinement of POS tag x and the probability of the POS-annotated training corpus is maximized by a POS-based latent bigram language model. Let z_1^n represent the sequence of latent tags. The joint probability of (o_1^n, z_1^n) according to a POS-based latent language model is computed as follows:

$$\mathrm{P}(O_1^n, Z_1^n) = \prod_{i=1}^{n} \mathrm{P}(O_i|Z_i)\mathrm{P}(Z_i|Z_{i-1}) \tag{4.21}$$

Accordingly, the joint probability of (o_1^n, x_1^n) is computed as follows:

$$\mathrm{P}(O_1^n, X_1^n) = \sum_{\substack{1 \le k \le n \\ z_k \in \mathcal{Z}(x_k)}} \mathrm{P}(O_1^n, Z_1^n = z_1^n) = \sum_{\substack{1 \le k \le n \\ z_k \in \mathcal{Z}(x_k)}} \prod_{i=1}^{n} \mathrm{P}(O_i|Z_i)\mathrm{P}(Z_i|Z_{i-1}) \quad (4.22)$$

and the probability of the word sequence o_1^n is computed as follows:

$$\mathrm{P}(O_1^n) = \sum_{\substack{1 \le j \le n \\ x_j \in \mathcal{X}(o_j)}} \sum_{\substack{1 \le k \le n \\ z_k \in \mathcal{Z}(x_k)}} \prod_{i=1}^{n} \mathrm{P}(O_i|Z_i)\mathrm{P}(Z_i|Z_{i-1}) \quad (4.23)$$

This POS-based latent language model is essentially a latent bigram tagger, which is an instance of latent HMM, and can be trained in the same way as a latent bigram tagger. The split-merge latent HMM training algorithm in Section 2.4.2.3 is able to split each POS tag into a set of fine-grained latent tags, each representing a specific type of word usage that is helpful for word prediction. For example, separating animal nouns such as *cat*, *dog*, or *tiger* from vehicle nouns such as *car*, *truck*, or *SUV* is beneficial for predicting whether the following word is more likely to be *bite* or *battery*.

4.7 Experiments

4.7.1 Setup

We evaluate the performance of the POS-based latent language model on the POS-tagged WSJ Penn treebank [103]. We use sections 00-22 (∼1M words) for training, section 24 (∼33k words) for development, and section 23 (∼57k words)

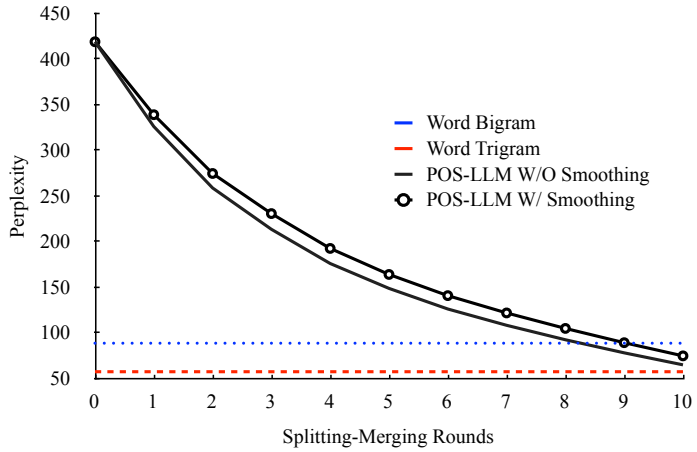


Figure 4.2: The perplexity of the POS-based latent language model with and without smoothing on the training set over split-merge iterations. The perplexities of the standard word bigram and trigram language models with modified Kneser-Ney fixed smoothing are also included for comparison.

for testing perplexity. All words that occur no more than five times in the training set are mapped to a special *UNK* token. For comparison, the SRI-LM toolkit [144] is used to build standard bigram and trigram language models using the modified Kneser-Ney fixed smoothing.

The POS-based latent language model is trained in the same way as the latent bigram POS tagger, and we experiment with or without the smoothing operation described in Section 3.5.1.

4.7.2 Results

Figure 4.2 reports the perplexity of the POS-based latent language model with and without smoothing on the training set. Before splitting POS tags, this model has a high perplexity of over 400. As the split-merge procedure proceeds, the perplexity on the training data decreases quickly and eventually becomes lower than

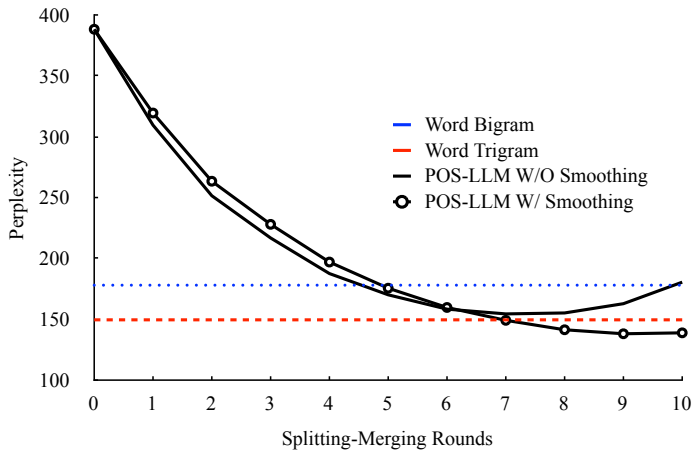


Figure 4.3: The perplexity of the POS-based latent language model with and without smoothing on the development set over split-merge iterations. The perplexities of the standard word bigram and trigram language models with modified Kneser-Ney fixed smoothing are also included for comparison.

the training perplexity of the word bigram language model with modified Kneser-Ney fixed smoothing and gets close to that of the word trigram language model. This suggests that the training algorithm can learn parameters associated with latent tags to fit the training data as well as standard word bigram and trigram models. The smoothing algorithm makes the POS-based latent language model fit the training data less well, but as we will show later, it prevents the POS-based latent language model from over-fitting the training data, and so it performs better on the test set.

Figure 4.3 reports the perplexity of the POS-based latent language model with and without smoothing on the development set. This model initially has a high perplexity of around 390 when no POS tags are split. As the split-merge procedure proceeds, the perplexity of the POS-based latent language model decreases quickly and eventually becomes lower than that of the word bigram model and gets close to the word trigram model. It is interesting to note that smoothing hurts the

Model	Dev	Test
Word Bigram	178.00	156.20
Word Trigram	149.53	125.29
POS-LLM Initial	388.24	383.97
POS-LLM Final	138.08	120.85

Table 4.1: The perplexity of four language models on the development and test set: word bigram model, word trigram model, the initial POS-based latent language model, and the final POS-based latent language model with smoothing

performance of the POS-based latent language model initially because data sparsity is not a severe issue when there are not many latent tags. However, as the number of latent tags increases, smoothing becomes important. The performance of the POS-based latent language model starts to degrade after the 8-th split-merge round when the model parameters are not smoothed. In contrast, the performance continues to improve with smoothing and achieves a perplexity that is lower than the word trigram model at the 8-th split-merge round, with the lowest perplexity obtained at the 9-th round. Table 4.1 reports the perplexity scores on the development and test sets. The POS-LLM model after 9-th split-merge training is chosen for the final results.

4.8 Conclusions

In this chapter, we briefly reviewed the traditional word N-gram models and class-based language models, discussed their weaknesses, and presented a latent language model that has the potential to address these issues. We implemented a POS-based latent language model based on a latent bigram tagger to split POS tags into fine-grained latent tags to learn contextual dependencies. Experimental results

showed that the POS-based latent language model performs slightly better than a word trigram model with modified Kneser-Ney fixed smoothing when measured by test perplexity. It is significantly better than the previously proposed class-based language models that do not involve lexical dependencies and has the potential to be interpolated with a word N-gram model to achieve higher perplexity reduction.

Chapter 5

Improvement of PCFG Grammars with Latent Annotations

5.1 Overview

There is an extensive research literature on building high quality probabilistic parsers based on probabilistic context free grammars (PCFGs) by incorporating lexical features and/or complex dependencies among treebank categories into the hierarchical generation process for sentences [31, 48, 132]. PCFG grammars with latent annotations (PCFG-LA) [106, 123, 126] are a recent enhancement that has raised considerable interest in the research community. In contrast to traditional models whose parameterization is predefined and fixed, PCFG-LA grammars model complex syntactic dependencies among units of a parse tree through the use of fine-grained latent syntactic categories that are automatically induced, and have been proven to achieve high levels of parsing accuracies.

Since the latent annotations of PCFG-LA grammars are learned automatically from the training data in a data-driven way, training PCFG-LA grammars is language independent and the resulting models can better parse a variety of languages than the traditional approaches that rely on expert knowledge for building language specific parsing models [61, 121, 123]. However, several issues need to be carefully addressed in order to train high quality PCFG-LA parsers, especially for those less commonly studied languages whose training data is more limited in quantity.

Over-fitting and out-of-vocabulary (OOV) words are two important issues when training PCFG-LA grammars. In order to address these issues, we investigate heuristic approaches to handle rare words and OOV words that arise typically when the amount of training data is limited. Our experiments on English and Chinese show that the heuristic approaches we developed result in improved parsing performance on both languages. In Chapter 7, we will present a principled approach to address these two issues using a feature-rich log-linear lexical model.

While it is always desirable to have more human annotated treebank data to train high quality parsing models, such treebank data is often limited in quantity, especially for less commonly studied languages, because human annotation requires expert knowledge and is both expensive and time-consuming to produce. Recognizing that unlabeled data is often ubiquitous and can be obtained in large quantities at a low cost, we address the data sparsity issue by combining treebank data with a large amount of automatically labeled training data to train PCFG-LA grammars. Using a comparative study, we find that a PCFG-LA parser benefits much more from self-training than a state-of-the-art lexicalized parser [31], and achieves state-of-the-art parsing accuracies for a single parser on both English (91.5 F) and Chinese (85.2 F). Analysis of results shows that the advantage of PCFG-LA grammars with self-training comes from its flexibility to adjust model complexity based on the amount and quality of the training data. In Chapter 6, we will also investigate a more effective self-training method that utilizes the variability among the PCFG-LA grammars together with greater quantities of automatically labeled training data.

The rest of this chapter is organized as follows. Parsing is briefly introduced in Section 5.2, and PCFG-LA grammars are described in Section 5.3. Several issues for PCFG-LA grammars are discussed together with our proposed solutions in Section 5.4. Experimental results are presented and discussed in Section 5.5. Section 5.6 concludes this chapter.

5.2 Introduction to Parsing

Syntactic parsing is the process of determining the grammatical structure of a sentence with respect to a given grammar, which can be broadly classified as either a phrase structure grammar introduced by Chomsky [44] or a dependency grammar proposed in (Tesnière [146]). Syntactic parsing is helpful for understanding the meaning of a sentence and has been applied to tasks such as machine translation [56, 99, 163], information extraction [113, 145], and sentence boundary detection [63].

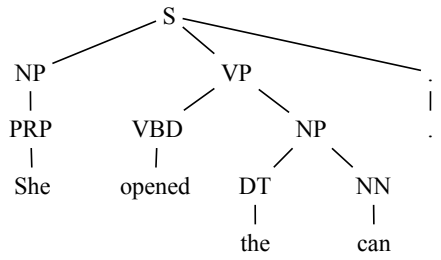


Figure 5.1: An example sentence with its syntactic parse tree

Figure 5.1 gives an example syntactic analysis for sentence "She opened the can ." given a phrase structure grammar. While this simple sentence has only one unique correct analysis, many sentences are ambiguous. Figure 5.2 gives two mean-

ingful syntactic analyses of sentence "Salespeople sold the dog biscuits"[16], which can mean the salespeople are selling "dog biscuits" or selling "biscuits to dogs". In probabilistic parsing, a grammar assigns probabilities to ambiguous parses, both meaningful and meaningless, and selects the most probable parse.

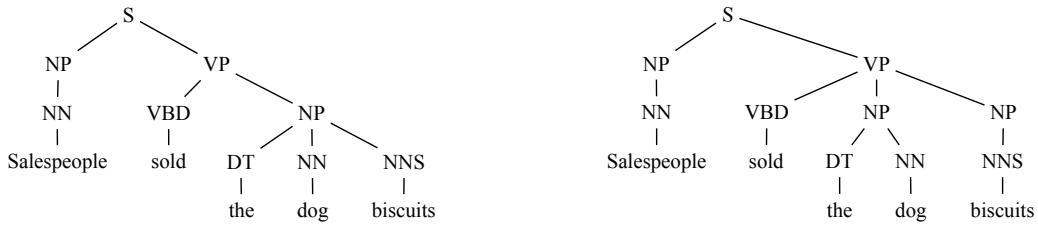


Figure 5.2: Two syntactic analyses for an ambiguous sentence

Probabilistic context-free grammars (PCFG) [12] are the simplest type of grammar for probabilistic parsing. As formally defined in Section 2.5, a PCFG grammar assigns a probability, $\mathrm{P}(r)$, to each of the grammar rules r, such as phrasal rule "NP $\rightarrow$ DT NN" and lexical rule "PRP $\rightarrow$ She". The probability of generating a grammar rule at a node is assumed to be independent of everything else given the label of the node, and the rule probabilities are typically obtained based on rule frequencies on a treebank, a collection of gold standard parse trees. The probability, $\mathrm{P}(T)$, of a parse tree, T, is computed as the product of the probabilities of the rules in the tree, i.e.,

$$\mathrm{P}(T) = \prod_{r \in \mathcal{R}(T)} \mathrm{P}(r)$$

in which $\mathcal{R}(T)$ denotes the set of grammar rules in parse tree T.

[16]The example is taken from (Charniak [29]).

Parsing with PCFG grammars can be done efficiently using the CKY algorithm [46, 80, 164]. The problem with PCFG grammars is that the context-free assumption is too strong for natural languages, which exhibit strong contextual dependencies [139], and as a result PCFG grammars achieve fairly low parsing accuracies [29]. More accurate parsers model lexical dependencies, as well as dependencies on other constituents such as parents and grandparents [31, 48]. The model of Charniak [31] assigns a probability to a parse tree T in a top-down fashion, in which for each constituent c in T ($\mathcal{C}(T)$ denotes set of constituents in T) it first generates the head preterminal $t(c)$, then the lexical head $h(c)$, and finally the sequence of constituents $e(c)$ expanded from c. In this model, the probability of a parse tree T is computed as:

$$\mathrm{P}(T) = \prod_{c \in \mathcal{C}(T)} \mathrm{P}(t(c)|l(c), H(c)) \cdot \mathrm{P}(h(c)|t(c), l(c), H(c)) \cdot \mathrm{P}(e(c)|l(c), t(c), h(c), H(c))$$

in which $l(c)$ denotes the label of a constituent and $H(c)$ denotes the relevant history information outside c that is important for determining the probability that is to be estimated. For example, in order to determine the probability distribution of preterminals of the head word of constituent c, it might be useful to include the label, head preterminal, and head of the parent $p(c)$, the label of the grandparent $g(c)$, as well as the label of c's left sibling $b(c)$ into $H(c)$, i.e.,

$$\mathrm{P}(t(c)|l(c), H(c)) = \mathrm{P}(t(c)|l(c), l(p(c)), t(p(c)), h(p(c)), l(g(c)), l(b(c))) \qquad (5.1)$$

As more information is included in the conditional history, reliable estimation of probabilities becomes more difficult. Charniak [31] employed a maximum entropy inspired approach to deal with data sparsity. Equation 5.1 was approximated by:

$$\begin{aligned} & \mathrm{P}(t|l, l_p, t_p, h_p, l_g, l_b) \\ \approx \; & \mathrm{P}(t|l)\frac{\mathrm{P}(t|l, l_p)}{\mathrm{P}(t|l)}\frac{\mathrm{P}(t|l, l_p, t_p)}{\mathrm{P}(t|l, l_p)}\frac{\mathrm{P}(t|l, l_p, t_p, l_b)}{\mathrm{P}(t|l, l_p, t_p)}\frac{\mathrm{P}(t|l, l_p, t_p, l_g)}{\mathrm{P}(t|l, l_p, t_p)}\frac{\mathrm{P}(t|l, l_p, t_p, h_p)}{\mathrm{P}(t|l, l_p, t_p)} \end{aligned} \quad (5.2)$$

where a simplified notation is used for brevity, e.g., $l(p(c))$ is abbreviated as l_p. Each conditional probability on the right-hand side of the equation is estimated by deleted interpolation [28].

Lexicalized parsers [31, 48] achieve around 90% parsing F score on English and have been used as the first-stage generative parsers for the better-performing discriminatively trained reranking parsers [33, 47]. However, these lexicalized parsers rely on human experts, together with trial and error, to decide which history information to condition on as in Equation 5.1, how to factor the conditional probabilities as in Equation 5.2, and how to smooth probabilities effectively. As a result, a configuration that performs well on one genre, domain, or language may not be optimal when applied to another. Substantial effort may be required to build a well-oiled parser model for the new condition.

5.3 PCFG Grammars with Latent Annotations

While lexicalization has proven important for achieving highly accurate parsing performance, Klein and Manning [82] showed that unlexicalized parsers can achieve

much higher performance than was previously expected by introducing linguistically motivated annotations (e.g., parent annotation and tag splitting) to better model linguistic dependencies. Matsuzaki et al. [106] extended this idea and proposed PCFG grammars with latent annotations (PCFG-LA). PCFG-LA grammars augment the observed parse trees from a treebank with a latent variable at each tree node. Each latent variable effectively refines an observed category t into a set of latent subcategories $\{t_x|x = 1, \cdots, |t|\}$, where $|t|$ denotes the number of latent tags split from t. For example, each syntactic category in the original tree in Figure 5.3(a) is split into multiple latent subcategories, and that parse tree is decomposed into many derivation trees whose non-terminals are latent categories; Figure 5.3(b) depicts one such derivation tree, where each grammar rule expands a latent non-terminal category into a sequence of latent non-terminals and/or terminal words, e.g., VP-4→VBD-5 NP-6.

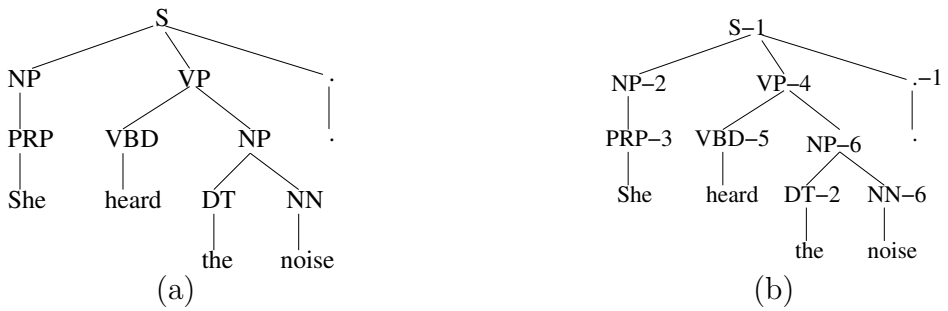


Figure 5.3: (a) original treebank tree, (b) with latent annotations

The objective of PCFG-LA training is to induce a grammar with latent variables that maximizes the probability of the training trees. Given a binarized PCFG-LA grammar with model parameter θ, $\mathcal{R}$ denotes the set of grammar rules, $\mathcal{Z}(T)$ the set of derivation trees for parse tree T, and $\mathcal{R}(T)$ and $\mathcal{R}(Z)$ the sets of rules com-

prising T and Z, respectively. The probability of T under the grammar is computed as:

$$\mathrm{P}_\theta(T) = \sum_{Z \in \mathcal{Z}(T)} \mathrm{P}_\theta(Z) = \sum_{Z \in \mathcal{Z}(T)} \prod_{r \in \mathcal{R}(Z)} \mathrm{P}_\theta(r)$$

The EM-algorithm can be used to train model parameter θ to maximize the training likelihood. The E-step computes the expected count e_r of rule r over the training set $\mathcal{T}$ under the current model parameter θ':

$$e_r \leftarrow \sum_{T \in \mathcal{T}} \sum_{r' \in \mathcal{R}(T)} \delta(r', r) \mathrm{P}_{\theta'}(r'|T) \tag{5.3}$$

where $\delta(\cdot, \cdot)$ is an indicator function that returns 1 if the two operands are identical and 0 otherwise, $\mathrm{P}_{\theta'}(r'|T)$ is the posterior probability of having (latent) rule r' in parse tree T and can be computed efficiently based on the inside-outside algorithm [91]. The M-step aims to maximize the intermediate objective:

$$l(\theta) = \sum_{r \in \mathcal{R}} e_r \log \mathrm{P}_\theta(r) \tag{5.4}$$

Given a binary phrasal rule $t_x^p \rightarrow t_y^l \, t_z^r$, the EM algorithm updates its expansion probability $\theta_{t_x^p \rightarrow t_y^l \, t_z^r} = \mathrm{P}_\theta(t_x^p \rightarrow t_y^l \, t_z^r)$ as:

$$\mathrm{P}_\theta(t_x^p \rightarrow t_y^l \, t_z^r) = \frac{e(t_x^p \rightarrow t_y^l \, t_z^r)}{e(t_x^p \rightarrow \cdot)} \tag{5.5}$$

where $e(t_x^p \rightarrow t_y^l \, t_z^r)$ denotes the expected count of rule $t_x^p \rightarrow t_y^l \, t_z^r$ and $e(t_x^p \rightarrow \cdot)$

denotes the expected count of all rules expanded from t_x^p. The unary phrasal rule probabilities are updated similarly. The lexical rule probability $\theta_{t_x \to w} = \mathrm{P}_\theta(w|t_x)$ is updated as:

$$\mathrm{P}_\theta(w|t_x) = \frac{e_{t_x,w}}{\sum_{w'} e_{t_x,w'}} \tag{5.6}$$

where $e_{t_x,w}$ denotes the expected count of lexical rule $r = t_x \to w$.

In order to allocate grammar complexity to where it is most needed, Petrov et al. [126] developed a simple split-merge (SM) procedure. In every split-merge round, each latent category is first split into two, and the model is re-estimated using several rounds of EM iterations. A likelihood criterion is then used to merge back the least useful splits. The result is that categories, such as NP (noun phrase) and VB (base verb), which occur frequently in different syntactic environments, are split more heavily than categories such as UH (interjection). This approach also creates a hierarchy of latent categories that enables efficient coarse-to-fine parsing [36, 123]. We call a grammar trained after n split-merge rounds an SMn grammar.

Given a PCFG-LA grammar with parameter θ and sentence s to be parsed, the decoding algorithm searches for the best parse tree $\hat{T}$ such that the product of posterior probabilities[17] of the original grammars rules is maximized, i.e.,

$$\hat{T} = \arg\max_T \sum_{r \in \hat{\mathcal{R}}(T)} \log \mathrm{P}_\theta(r|s) \tag{5.7}$$

[17]Finding the most probable parse tree is NP-hard. Petrov and Klein [123] studied several tractable alternative decoding methods, including the max-rule-product method in Equation 5.7.

where $\hat{\mathcal{R}}(T)$ denotes the unsplit grammar rules comprising parse tree T and the posterior rule probability $\mathrm{P}_\theta(r|s)$ can be computed efficiently using the inside-outside algorithm. We refer the readers to (Petrov et al. [126]) and (Petrov and Klein [123]) for more details on the learning and inference algorithms.

Note that at each splitting step, some randomness is introduced to break symmetry to initialize model parameters for EM training, in a way similar to the splitting operation in Section 2.4.2.3. EM is a local method, making no promises regarding the final point of convergence when initialized from different random seeds. For experiments in this chapter, we always train grammars with multiple random seeds and use the development set to pick the best grammar for evaluation. Petrov [122] was able to take advantage of the variability among grammars using a product model to achieve improved parsing accuracies. In Chapter 6, we will exploit this variability among grammars together with self-training to train highly accurate PCFG-LA grammars.

For this thesis, we implemented our own PCFG-LA parser, borrowing key ideas from the Berkeley parser [123, 126], an implementation of PCFG-LA grammars. Our parser implements a novel language-independent rare word smoothing method and language-dependent OOV word handling methods, which we will describe next in Section 5.4. Both the training and decoding algorithms are also parallelized to take advantage of multi-core machines. The parallelization of the EM algorithm is crucial for training a model with large volumes of data in a reasonable amount of time, especially for the self-training experiments[18] described in Section 5.5. Our

[18]The parallel version is able to train grammars with automatically labeled training data on

parser can also parse with a product of PCFG-LA grammars and implements the feature-rich log-linear lexical model that we will discuss in Chapter 6 and Chapter 7, respectively.

5.4 Improving PCFG-LA Grammars

Previous studies have shown that PCFG-LA grammars outperform the state-of-the-art lexicalized parsers and can be flexibly applied to a variety of languages [1, 61, 121, 123]; however, parsing accuracies on non-English languages are considerably lower than for English; they are usually in the range of 80%∼85% compared to over 90% on the English WSJ treebank [121]. Take Chinese for example; there have been several attempts to develop accurate parsers for Chinese [10, 93, 123], but the best previously known accuracy, around 83% on Chinese Penn treebank achieved by the Berkeley parser [123], falls far short of the performance on English. The intrinsic characteristics of the languages, as well as annotation consistency of the treebanks, may contribute to the challenge of parsing non-English languages, as discussed in (Levy and Manning [93]) for Chinese and in (Green and Manning [61]) for Arabic. Several issues need to be carefully addressed in order to train high quality PCFG-LA parsers, especially for those less commonly studied languages whose available treebanked materials are more limited than for English. We will next discuss these issues and how we choose to address them.

an 8-core machine within a week, while the non-parallel version was unable to finish after even 3 weeks.

5.4.1 Smoothing

The Expectation Maximization (EM) algorithm used for training the PCFG-LA grammars guarantees that each EM iteration will increase the training likelihood [95]. As the number of latent annotations increases, a PCFG-LA grammar has increasing power to fit the training data through EM training and eventually begins to over-fit. For example, when trained on sections 2-21 of the WSJ Penn treebank, the performance of the parser starts to drop after 5 split-merge rounds [126]. In order to counteract this behavior, Petrov et al. [126] introduced a linear smoothing method to smooth rule probabilities. The lexical rule probabilities are smoothed as follows:

$$\begin{aligned} \bar{\mathrm{P}} &= \frac{1}{|t|}\sum_{x}\mathrm{P}_{\theta}(w|t_x) \\ \mathrm{P}_{\theta}(w|t_x) &\leftarrow \epsilon\bar{\mathrm{P}} + (1-\epsilon)\mathrm{P}_{\theta}(w|t_x) \end{aligned}$$

and the unary phrasal rule probabilities (and similarly for binary phrasal rule probabilities) are smoothed as:

$$\begin{aligned} \bar{\mathrm{P}} &= \frac{1}{|t|}\sum_{x}\mathrm{P}_{\theta}(t_x^p \rightarrow t_y^c) \\ \mathrm{P}_{\theta}(t_x^p \rightarrow t_y^c) &\leftarrow \epsilon\bar{\mathrm{P}} + (1-\epsilon)\mathrm{P}_{\theta}(t_x^p \rightarrow t_y^c) \end{aligned}$$

This smoothing procedure allows grammars (trained on sections 2-21 of WSJ treebank) to go from 5 to 6 split-merge rounds with an increase in parsing accuracy on a held-out set, due to the combination of more robust parameter estimates and

the increased expressiveness of the model.

Although the lexical smoothing method is able to make word emission probabilities of the latent tags split from the same POS tag more alike, the EM training algorithm still strongly discriminates among word identities and can cause unreliable probability estimates for rare words. Suppose word tag pairs (w_1, t) and (w_2, t) both appear the same number of times in the training data. In a standard PCFG grammar (without latent annotations), whose parameters are estimated based on maximum likelihood estimation, the probabilities of emitting these two words given the tag t would be the same, i.e., $\mathrm{P}_\theta(w_1|t) = \mathrm{P}_\theta(w_2|t)$. After introducing latent annotation x to tag t, the emission probabilities of these two words given a latent tag t_x may no longer be the same because $\mathrm{P}_\theta(w_1|t_x)$ and $\mathrm{P}_\theta(w_2|t_x)$ are two different parameters that are trained independently by the EM algorithm. It is beneficial to learn subcategories of POS tags to model different types of words, especially for frequent words; however, it is not desirable to strongly discriminate among rare words because unreliable probability estimates could be produced to distract the model from learning about common phenomena.

Our solution to this problem is to tie the emission probabilities of rare words together so that $\mathrm{P}_\theta(w_1|t_x) = \mathrm{P}_\theta(w_2|t_x)$ for all x if both w_1 and w_2 are rare words and $\mathrm{P}_\theta(w_1|t) = \mathrm{P}_\theta(w_2|t)$. In this *rare word smoothing* method, words with a frequency less than a threshold τ are considered rare words[19] and are mapped to the *rare* symbol, and their emission probabilities $\mathrm{P}_\theta(w|t_x)$ are set in proportion to their co-

[19] τ is tuned on the development set.

occurrences with the surface POS tag:

$$\mathrm{P}_\theta(w|t_x) \;=\; \frac{c_{t,w}}{\sum_{w':e_{\cdot,w'}<\tau} c_{t,w'}}\mathrm{P}_\theta(\mathit{rare}|t_x)$$

where $c_{\cdot,w}$ and $c_{t,w}$ are the observed counts of the word and word/tag pair, respectively, and $\mathrm{P}_\theta(\mathit{rare}|t_x)$ is a free parameter tuned by the EM algorithm. This smoothing method greatly reduces the number of free parameters and has been found to significantly improve parsing accuracies.

5.4.2 OOV Handling

PCFG-LA grammars are trained to optimize likelihood on the training data and the resulting lexical model $\mathrm{P}_\theta(w|t_x)$ can only generate words observed in the training data. As typical of generative models, a separate module is needed to handle the OOV words that can appear in test sentences. A simple approach is to estimate the emission probability of an OOV word w based on how likely t_x is associated with a rare word in the training data:

$$\mathrm{P}_\theta(w|t_x) = \mathrm{P}_\theta(\mathit{rare}|t_x)$$

We call this the *simple* method. This method is used in the simple lexicon of the Berkeley parser [126].

A better approach would exploit the morphology of the language. As with other generative English parsers [31, 48], the Berkeley parser classifies OOV words

into a set of OOV signatures based on the presence of features such as capital letters, digits, dashes, as well as a list of indicative suffixes (e.g., *-ing*, *-ion*, *-er*), and estimates the emission probability of an OOV word w given a tag t as:

$$\mathrm{P}_\theta(w|t_x) \propto \mathrm{P}_\theta(s|t_x)$$

where s is the OOV signature for w and $\mathrm{P}_\theta(s|t_x)$ is computed[20] by $e_{t_x,s}/e_{t_x,\cdot}$.

While this approach performs well for English, this English OOV model is not appropriate for other languages since they have different word formation processes [24]. Nevertheless, in the multi-language study of Petrov [121], the above hand-crafted signature-based rules designed for English OOV words were used for non-English languages. For non-English languages, this is clearly suboptimal and further efforts must be expended to build appropriate methods for each language investigated.

In order to build an effective OOV model for parsing Chinese, it is important to take the Chinese morphology into account. As discussed in (Packard [120]), the word formation process for Chinese words can be quite complex. Indeed, the last characters in a Chinese word are, in some cases, most informative of the POS type, while for others, it is the characters at the beginning. Furthermore, it is not uncommon for a character in the middle of a word to provide some evidence for the POS type of the word. Hence, we developed a character-based OOV model similar to (Huang et al. [72]) to reflect the fact that characters in any position (prefix, infix,

[20] $e_{t_x,s}$ is computed based on the expected counts (i.e., $e_{t_x,w}$) of words whose signature is s.

or suffix) can be predictive of the POS type for Chinese words. In our model, the word emission probability of an OOV word w given the latent tag t_x of POS tag t is estimated by using the geometric average of the emission probabilities of all of the characters ch_k in the word (where $n = |\{ch_k \in w | \mathrm{P}_\theta(ch_k|t_x) \neq 0\}|$):

$$\mathrm{P}_\theta(w|t_x) = \sqrt[n]{\prod_{ch_k \in w, \mathrm{P}_\theta(ch_k|t_x) \neq 0} \mathrm{P}_\theta(ch_k|t_x)} \tag{5.8}$$

where $\mathrm{P}_\theta(ch_k|t_x)$ is computed in way similar to $\mathrm{P}_\theta(s|t_x)$ in the signature-based model for English. In case Equation 5.8 cannot be used to compute the emission probability (i.e., all characters are previously unknown), we back off to use the *simple* OOV handling method.

We call this the *heuristic* method because it is unable to utilize overlapping features and requires a nontrivial amount of work to develop an OOV handling method for a new language. In Chapter 7, we will present a more principled approach that uses a feature-rich log-linear lexical model to better handle OOV words, while also addressing the over-fitting problem.

5.4.3 Self-Training

Similar to POS tagging as we discussed in Section 3.6, early investigations on self-training for parsing had mixed results. Charniak [30] reported no improvements from self-training his lexicalized parser on the standard WSJ training set. Steedman et al. [143] reported some degradation using a lexicalized tree adjoining grammar parser and minor improvement using Collins lexicalized PCFG parser; however, this

gain was obtained only when the parser was trained on a small treebank data set. Reichart and Rappoport [133] obtained significant gains using Collins lexicalized parser with a different self-training protocol, but again they only looked at small treebank data sets. McClosky et al. [107] effectively utilized unlabeled data to improve parsing accuracy on the standard WSJ training set, but they used a two-stage parser comprised of Charniak's lexicalized probabilistic parser with n-best parsing and a feature-rich discriminative reranking parser [33], both requiring a significant amount of time for feature engineering in order to work well. It is worth noting that their attempts at directly self-training Charniak's lexicalized parser resulted in no improvement.

Here we provide one possible explanation of the mixed results of self-training for parser models. As we pointed out in Section 5.2, the parameterization of traditional models, such as Charniak's parser, is predetermined and fixed, as shown in Equation 5.2, and is based on extensive development on a held-out set for models trained on the WSJ Penn treebank training data. The models are excessively complex when the amount of training data is small, thus adding more data (even if automatically labeled) can help alleviate the data sparsity problem and improve the robustness of parameter estimation. However, as more and more training data becomes available, those models' ability to learn from the additional training data is limited due to the fact that the model parameterization is fixed.

We argue that the parameterization of PCFG-LA grammars can be flexibly adjusted to accommodate varying amounts of training data. In fact, we will show that self-training is able to significantly improve the performance of the PCFG-LA

parser, a single generative parser, using both small and relatively large amounts of treebank training data, for both English and Chinese. With self-training, a fraction of the WSJ or CTB6 treebank training data is sufficient to train a PCFG-LA parser that is able to achieve or even exceed the accuracies obtained using a single parser trained on the entire treebank without self-training. When self-training using the full training data, we are able to improve upon state-of-the-art parsing accuracies for a single grammar parser on both English (91.5%) and Chinese (85.2%).

5.5 Experiments

5.5.1 Setup

For the English experiments, the WSJ Penn treebank [103] is used as the gold standard data. We use sections 2-21 for training, section 22 for development, and section 23 for final evaluation. We also use 210k sentences[21] of unlabeled news articles in the BLLIP corpus for self-training the English parsers.

For the Chinese experiments, the Chinese Penn treebank 6.0 (CTB6) [162] is used as the gold standard data. CTB6 includes both news articles and transcripts of broadcast news. We partition the news articles into train/development/test sets in the same way as in the Chinese POS tagging experiments in Section 3.7. The broadcast news section is also added to the training data because it shares many of the characteristics of newswire text (e.g., fully punctuated, contains nonverbal expressions such as numbers and symbols). In addition, 210k sentences of unlabeled

[21] This amount was constrained based on both CPU and memory.

	Train	Dev	Test	Unlabeled
English	39.8k (950.0k)	1.7k (40.1k)	2.4k (56.7k)	210k (5,082.1k)
Chinese	24.4k (678.8k)	1.9k (51.2k)	2.0k (52.9k)	210k (6,254.9k)

Table 5.1: The number of sentences (and tokens in parentheses) in our experiments

Chinese news articles are used for self-training. Since the Chinese parsers in our experiments require word-segmented sentences as input, the unlabeled sentences need to be word-segmented first. As shown in (Harper and Huang [62]), the accuracy of automatic word segmentation has a great impact on Chinese parsing performance. We choose to use the Stanford segmenter [27] in our experiments because it is trained to be consistent with the treebank segmentation and provides the best performance among the segmenters tested in (Harper and Huang [62]). To minimize the discrepancy between the self-training data and the treebank data, we normalize both CTB6 and the self-training data using the UW Decatur text normalization[22] [165].

Table 7.1 summarizes the data set sizes used in our experiments. We use slightly modified versions of the treebanks with empty nodes and nonterminal-yield unary rules[23] (e.g., NP→VP) deleted using tsurgeon [92]. We train parsers on 20%, 40%, 60%, 80%, and 100% of the treebank training data to evaluate the effect of the amount of treebank training data on parsing performance, as well as to compare

[22]This normalization largely maps full-width punctuation to half-width, collapsing some of the punctuation distinctions available in the former representation. Decatur normalization results in a slight degradation (around 0.1%) in F measure when measured on CTB6. We consider this acceptable in order to be consistent with the normalization that is applied to the unlabeled data.

[23]As nonterminal-yield unary rules are less likely to be posited by a statistical parser, it is common for parsers trained on the standard Chinese treebank to have substantially worse recall than precision. This gap between bracket recall and precision is alleviated without loss of parse accuracy by deleting the nonterminal-yield unary rules.

how self-training impacts models trained with different amounts of treebank data. This allows us to simulate scenarios where a language has limited gold standard resources. The development set is used to select the best random seed and split-merge round for the PCFG-LA parser and to tune the smoothing parameters for Charniak's parser.

5.5.2 Rare Word Smoothing and OOV Word Handling

We first study the effect of rare word smoothing and OOV word handling for the PCFG-LA parsers trained on 100% of the treebank training data. The results are shown in Table 5.2. The *no+heuristic* row for WSJ represents the baseline performance of our parser as a reimplementation of the Berkeley parser. The rare word smoothing method (the *yes+heuristic* row) significantly improves the parsing F score from 90.0 to 90.6 on English. The *no+simple* row for CTB6 represents the baseline performance of our Chinese parser. The heuristic OOV word handling and the rare word smoothing improve the parsing F score of Chinese by 0.5 and 0.6, respectively, and their combination achieves an even better improvement of 0.9 F. All the improvements are statistically significant[24].

We found in our experiments that the rare word smoothing method is more effective when more latent annotations are allocated. This is probably because data sparsity becomes a more severe problem with a larger amount of latent annotations and so sharing statistics of rare words helps alleviate this problem in order to produce

[24]We use Bikel's randomized parsing evaluation comparator to determine the significance ($p < 0.05$) of the difference between two parsers' output.

[10]We do not evaluate the simple OOV handling method on English because it was already shown in the Berkeley parser that the heuristic signature-based method works better.

Rare Word Smoothing	OOV	WSJ	CTB6
no	simple	n/a	83.2
no	heuristic	90.0	83.7
yes	simple	n/a	83.8
yes	heuristic	90.6	84.2

Table 5.2: Effects of rare word smoothing (no vs. yes) and OOV word handling (simple vs. heuristic)[10] on the test set as measured in parsing F score (%)

more robust grammars. For a similar reason, the impact of rare word smoothing is also more significant when trained on a smaller data set. When trained on 20% of CTB6, rare word smoothing improves the best performing grammar (SM4 grammar with heuristic OOV handling) from 78.1 to 79.7, a 1.6 absolute improvement in F score, in contrast to the smaller improvement of 0.4 when trained on 100% of CTB6 (see Table 5.2).

Similar to rare word smoothing, the OOV word handling method also provides greater improvements on grammars trained on smaller amounts of training data, producing a 1.2 absolute improvement in F score for the best performing grammar (SM4 grammar with rare word smoothing) when trained on 20% of the CTB6 training data compared to the smaller improvement of 0.3 when trained on 100% of CTB6 (see Table 5.2).

5.5.3 A Case Study: PCFG-LA Parser vs. Charniak's Parser

We next conduct a series of studies to compare the PCFG-LA parser with Charniak's parser on both English and Chinese.

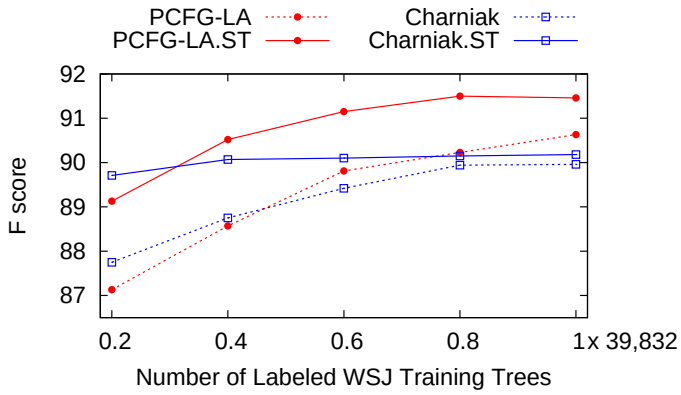
5.5.3.1 Treebank Data Only

In this section, we compare the performance of the PCFG-LA parser and Charniak's parser when trained on treebank data alone. As shown by the dotted lines in Figure 5.4, it is clear that both parsers perform much better on English than Chinese when trained on the treebank data alone,. It is true that English treebank has more trees than the Chinese treebank, however, Chinese appears to be more challenging than English [93]. In fact, the English parsers trained on 20% (8.0k trees) of the WSJ training data have much higher parsing accuracies (>87 vs. <83) than the Chinese parsers trained on 100% (24.4k trees) of the CTB6 training data. The comparison between the two parsing approaches provides two interesting insights.

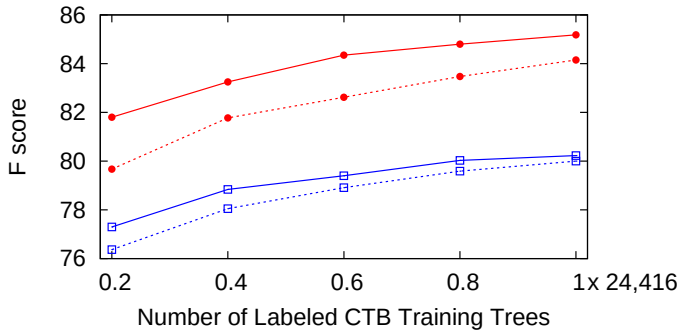
First, the PCFG-LA parser always performs significantly better than Charniak's parser on Chinese, although both model English well. Admittedly, Charniak's parser has not been optimized[26] on Chinese, but neither has the PCFG-LA parser[27]. The parameterization of Charniak's lexicalized parser was originally optimized for English and required sophisticated smoothing to deal with data sparsity. In contrast, the PCFG-LA parser automatically learns latent annotations from the data without any specification of what precisely should be modeled and how it should be modeled. This flexibility may help it to model new languages more effectively than Charniak's parser.

[26]The Chinese port includes modification of the head table, implementation of a Chinese punctuation model, etc.

[27]The PCFG-LA parser without the OOV word handling method still outperforms Charniak's parser on Chinese.



(a) English



(b) Chinese

Figure 5.4: The performance of the PCFG-LA parser and Charniak's parser when trained with different amounts of labeled training data, with and without self-training (ST), and evaluated on the test set

Second, while both parsers benefit from increased amounts of treebank training data, the PCFG-LA parser gains more. The PCFG-LA parser is initially poorer than Charniak's parser when trained on 20% of the WSJ treebank data. This is probably because this data is too small for it to learn fine-grained annotations without over-fitting given that the smoothing method is fairly simple, while the predefined parameterization of Charniak's parser is heavily smoothed and so can get by with less data. As more treebank training data becomes available, the performance of the PCFG-LA parser improves quickly and it finally outperforms Charniak's parser significantly. This is because more latent annotations can be allocated to learn more

complex dependencies. Moreover, the performance of the PCFG-LA parser continues to grow as the amount of treebank training data increases, while the performance of Charniak's parser levels out at around 80% of the treebank training data. The PCFG-LA parser improves by 3.5 absolute in F score when moving from 20% to 100% training data, compared to a 2.2 F gain for Charniak's parser. Similarly for Chinese, the PCFG-LA parser also gains more than Charniak's parser (4.5 vs 3.6 F). It is expected that if more gold training data would become available, the PCFG-LA parser would be able to continue to benefit from that additional data while the gain for Charniak's parser would probably be marginal.

5.5.3.2 Treebank Data and Self-Labeled Data

We next compare the effect of self-training on the performance of the two parsers. As shown by the solid lines in Figure 5.4, it is also clear that the PCFG-LA parser is able to benefit more from self-training than Charniak's parser. On the English data set, Charniak's parser benefits from self-training initially when there is little treebank training data, but the improvement levels out quickly as more treebank training trees become available. In contrast, the PCFG-LA parser benefits consistently from self-training[28], even when using 100% of the treebank training data. Similar trends are also found for Chinese.

It should be noted that the PCFG-LA parser trained on a fraction of the

[28]One may notice that the self-trained PCFG-LA parser with 100% WSJ training data has a slightly lower test accuracy than the self-trained PCFG-LA parser with 80% WSJ treebank data. This is due to the variance in parser performance when initialized with different seeds and the fact that the development set is used to pick the best model for evaluation. In Chapter 6, we will take advantage of the variability among the random grammars to better self-train PCFG-LA grammars.

treebank training data plus a large amount of automatically labeled training data, which comes at a much lower cost, performs comparably to or even better than grammars trained with additional treebank training data. For example, the PCFG-LA parser trained on the automatically labeled training data in addition to 60% of the treebank training data is able to outperform the grammar trained on 100% treebank training data alone for both English and Chinese. With self-training, even 40% of the WSJ treebank training data is sufficient to train a PCFG-LA parser that is comparable to the model trained on the entire WSJ training data alone. This result is of significant importance, especially for languages with limited human-labeled resources.

One might conjecture that the PCFG-LA parser benefits more from self-training than Charniak's parser because its automatically labeled data has a higher accuracy. However, as shown in Figure 5.4 (a), the PCFG-LA parser trained on 40% of the WSJ treebank data alone has a much lower performance than Charniak's parser trained on the full WSJ training set (88.6 vs 90.0 F). With the same amount of automatically labeled training data (labeled by each parser), the resulting PCFG-LA parser obtains a much higher F score than the self-trained Charniak's parser (90.5 vs 90.2 F). Similar patterns are also found on Chinese.

Table 5.3 reports the final test results when each parser is trained on the entire WSJ or CTB6 training set. For English, self-training contributes a 0.8 absolute improvement in F score to the PCFG-LA parser, which is comparable to the improvement obtained from self-training using the two-stage parser in (McClosky et al. [107]). Note that McClosky et al. [107]'s improvement is achieved with the addi-

	English	Chinese
PCFG-LA	90.6	84.2
+ Self-training	91.5	85.2

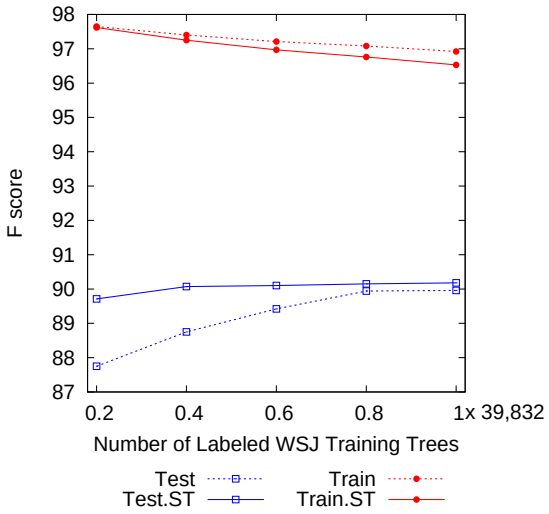
Table 5.3: Final results on the test set in F score (%)

tion of 2,000k unlabeled sentences using the combination of a generative parser and a discriminative reranker, compared to using only 210k unlabeled sentences with a single generative parser in our approach. For Chinese, self-training results in a state-of-the-art parsing model with 85.2 parsing F score (a 1.0 absolute improvement) on a representative test set. Both improvements are statistically significant.

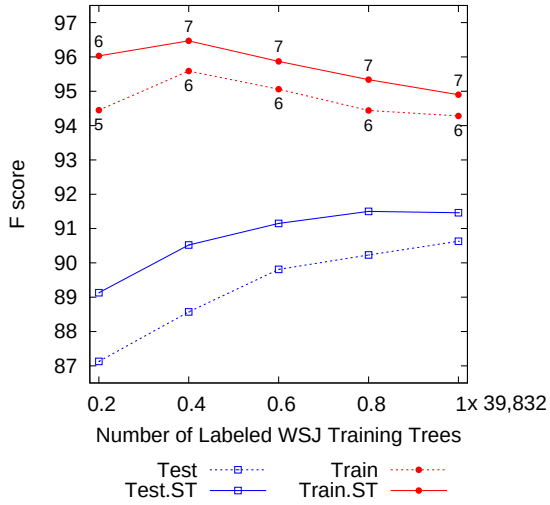
5.5.4 Analysis

We next perform a series of analyses to investigate why the PCFG-LA parser benefits more from additional data, most particularly automatically labeled data, than Charniak's parser.

Charniak's parser is a lexicalized PCFG parser that models lexicalized dependencies explicitly observable in the training data and relies on smoothing to avoid over-fitting. Although it is able to benefit from more training data because of broader lexicon and rule coverage and more robust estimation of parameters, its ability to benefit from the additional data is limited in the sense that it is unable to automatically update its parameterization to model more complex dependencies. In contrast to the PCFG-LA model, nontrivial human effort would be required to improve the model formulation. As shown in figure 5.5(a), the parsing accuracy of Charniak's parser on the WSJ test set improves as the amount of treebank train-



(a) Charniak



(b) PCFG-LA

Figure 5.5: (a) The training/test accuracy of Charniak's parser trained on varying amounts of WSJ treebank training data, with and without self-training (ST). (b) The training/test accuracy of the PCFG-LA parser trained on varying amount of WSJ treebank training data, with and without ST; the numbers along the training curves indicate the split-merge round of the grammars that are selected based on the performance on the development set.

ing data increases; however, the training accuracy[29] degrades as more data is added. Note that the training accuracy of Charniak's parser also decreases after the addition of self-training data[30]. This is expected for models with a fixed parameterization; it is harder to model more data with greater diversity. The addition of the automatically labeled training data initially helps to improve the performance of Charniak's parser on the test set, but it provides little gain when the treebank training data becomes relatively large.

Figure 5.5 (b) plots the training and test curves of the English PCFG-LA parser with varying amounts of treebank training data, with and without self-training. This figure differs substantially from Figure 5.5 (a) for Charniak's parser.

[29]This is the accuracy of the parser when parsing the treebank training data.

[30]The automatically labeled training trees are combined with the treebank trees in a weighted manner; otherwise, the training accuracy would be even lower.

First, as mentioned earlier, the PCFG-LA parser benefits much more from self-training than Charniak's parser with moderate to relatively large amounts of treebank training data. Second, in contrast to Charniak's parser for which training accuracy degrades consistently as the amount of treebank training data increases, when trained on more treebank training data, the training accuracy of the PCFG-LA parser sometimes improves by using more latent annotations (more split-merge rounds) without over-fitting. For example, the best model trained on 40% treebank training data alone, i.e., the SM6 grammar, has a higher training accuracy than the best model (at SM5) trained on 20% treebank training data. Third, the addition of automatically labeled data supports more accurate PCFG-LA grammars with more latent annotations than those trained without self-training, as evidenced by scores on both the training and test data. This suggests that the self-trained grammars are able to utilize more latent annotations to learn more complex dependencies.

In contrast to Charniak's parser, the PCFG-LA training algorithm is able to adapt the granularity of the grammar to the amount of training data available. Fewer latent annotations are allocated when the training set is small. As the size of the training data increases, it is able to allocate more latent annotations to better model the data. As shown in Figure 5.6, for a fixed amount of treebank training data (20%), the accuracy of the model on the training data (see the red curves) continues to improve as the number of latent annotation increases. Although it is important to limit the number of latent annotations to avoid over-fitting, the ability to accurately model the training data given sufficient latent annotations is desirable when more training data is available. When trained on 20% of the treebank data

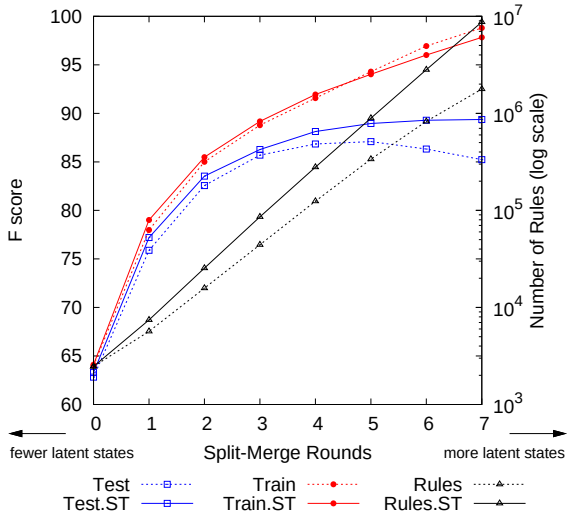
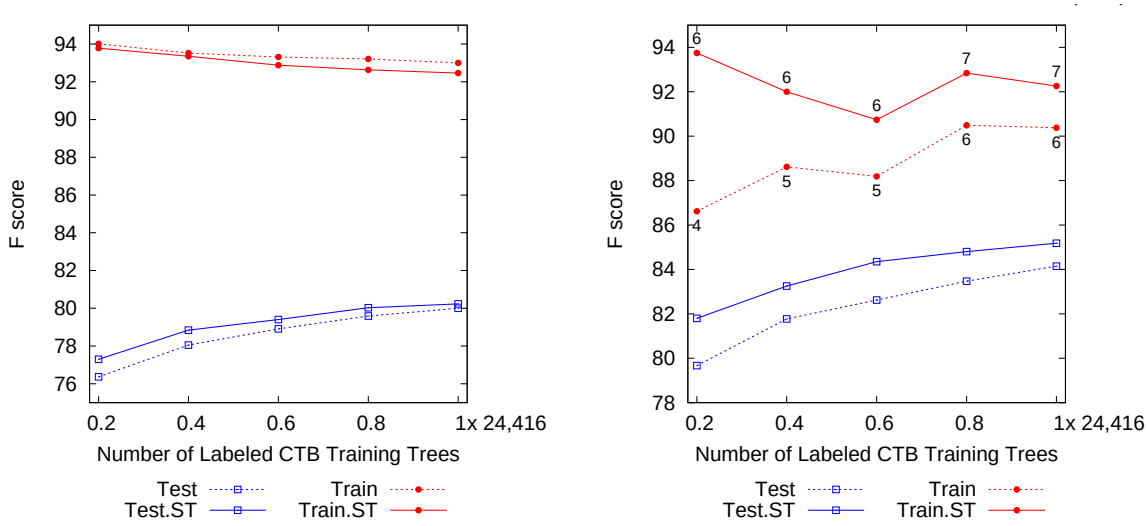


Figure 5.6: The training/test accuracy of the PCFG-LA grammars when trained on 20% of the WSJ treebank training data, with and without ST, and the number of nonzero rules.

alone, the SM5 grammar (selected using the development set) achieves its optimal test set performance and begins to degrade afterwords (see the blue curves). With the addition of the automatically labeled training data, the SM5 grammar achieves an greater accuracy on the test set and its performance continues to increase[31] with the increased number of latent annotations of the SM6 an SM7 grammars.

We have also compared the PCFG-LA parser to Charniak's Parser on Chinese and observed patterns similar to English. As shown in Figure 5.7, the training accuracy of Charniak's parser always decreases when more training data is added, due to its fixed parameterization. In contrast, the PCFG-LA parser has the potential to achieve very high training accuracies when using sufficient quantities of latent annotations, as shown in Figure 5.8. The PCFG-LA training algorithm has the flexibility to allocate latent annotations sufficient to account for varying amounts of

[31] Although the 20% self-trained grammar has a higher test accuracy at the 7-th round than the 6-th round, the development accuracy was better at the 6-th round, and thus we report the test accuracy of the 6-th round grammar in Figure 5.5 (b).



(a) Charniak (b) PCFG-LA

Figure 5.7: (a) The training/test accuracy of Charniak's parser trained on varying amounts of CTB treebank training data, with and without self-training (ST). (b) The training/test accuracy of the PCFG-LA parser trained on varying amount of CTB treebank training data, with and without ST; the numbers along the training curves indicate the split-merge round of the grammars that are selected based on the performance on the development set.

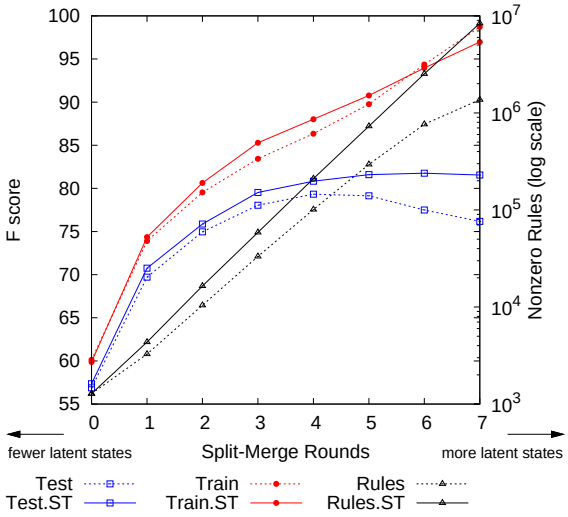


Figure 5.8: The training/test accuracy of the PCFG-LA grammars when trained on 20% of the CTB treebank training data, with and without ST, and the number of nonzero rules.

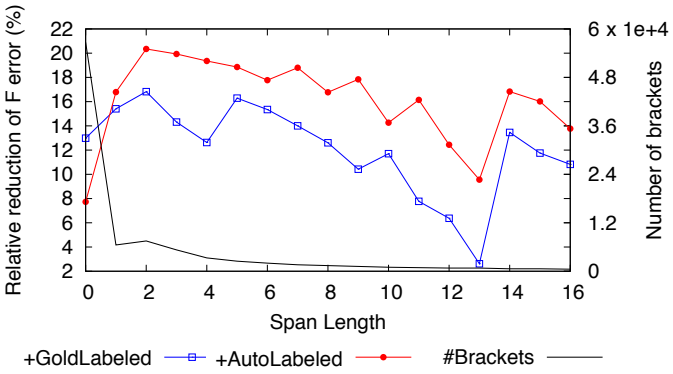


Figure 5.9: The relative reduction of bracketing errors for different span lengths, evaluated on the test set. The baseline model is the PCFG-LA parser trained on 20% of the WSJ training data. The +AutoLabeled curve corresponds to the parser trained with the additional automatically labeled data, and the +GoldLabeled curve corresponds to the parser trained with additional 20% treebank training data. The bracket counts are computed on the gold reference. Span length '0' denotes preterminal POS tags to differentiate them from the non-terminal brackets that span only one word.

training data. When the amount of treebank training data increases from 20% to 40% and then to 80%, the optimal number of split-merge iterations increases from 4 to 5 and then to 6, with increasingly higher accuracies on both the training and test sets. The addition of automatically labeled data also supports more accurate PCFG-LA grammars with more latent annotations than those trained without self-training. For example, compared to the initial grammar that achieves its optimal performance on the development set at the 4-th split-merge round when trained on 20% of the CTB treebank training data, the corresponding self-trained grammar achieves its best performance at the 6-th split-merge round with improved accuracies on both the training and test data sets.

Figure 5.9 compares the effect of additional treebank and automatically labeled data on the relative reduction of bracketing errors for different span lengths on English. It is clear from the figure that the improvement in parsing accuracy

from self-training is the result of better bracketing across all span lengths[32]. However, even though the larger amount of automatically labeled training data provides more improvement than the smaller amount of additional treebank data in terms of parsing accuracy, this data is less effective at improving tagging accuracy (see span length '0') than the additional treebank training data.

So, why does self-training improve rule estimation when training the PCFG-LA parser with more latent annotations? One possibility is that the automatically labeled data smooths the parameter estimates in the EM algorithm, enabling effective use of more parameters to learn more complex dependencies during model training. Let $\mathrm{P}(a \rightarrow b|r,T)$ be the posterior probability of expanding subcategories a to b given a expansion rule r on a treebank parse tree T. $\mathcal{T}_l$ and $\mathcal{T}_u$ are the sets of gold and automatically labeled parse trees, respectively. The update of the rule expansion probability $\mathrm{P}(a \rightarrow b)$ in self-training (with weighting parameter α) can be expressed as:

$$\frac{\sum\limits_{T\in\mathcal{T}_l}\sum\limits_{r\in\mathcal{R}(T)}\mathrm{P}(a\rightarrow b|r,T)+\alpha\sum\limits_{T\in\mathcal{T}_u}\sum\limits_{r\in\mathcal{R}(T)}\mathrm{P}(a\rightarrow b|r,T)}{\sum_{b'}(\sum\limits_{T\in\mathcal{T}_l}\sum\limits_{r\in\mathcal{R}(T)}\mathrm{P}(a\rightarrow b'|r,T)+\alpha\sum\limits_{T\in\mathcal{T}_u}\sum\limits_{r\in\mathcal{R}(T)}\mathrm{P}(a\rightarrow b'|r,T))} \tag{5.9}$$

Since the automatically labeled data is produced by a grammar with fewer latent annotations, the expected counts from the automatically labeled data can be thought of as counts from a lower-order model[33] that smooth the higher-order (with more

[32]There is a slight degradation in bracketing accuracy for some spans longer than 16 words, but the effect is negligible due to their low counts.

[33]We also trained models using only the automatically labeled data without combining it with treebank training data, but they were no more accurate than those trained on the treebank training data alone without self-training.

latent annotations) model.

We observe that many of the rule parameters of the grammar trained on WSJ training data alone have zero probabilities (rules with extremely low probabilities are also filtered to zero), as was also pointed out in (Petrov et al. [126]). On the one hand, this is what we want because the grammar should learn to avoid impossible rule expansions. On the other hand, this might also be a sign of over-fitting. As shown in Figure 5.6 (see the black curves), the grammar obtained with the addition of automatically labeled data contains many more non-zero rules, and its performance continues to improve with more latent annotations. Similar patterns also appear when using self-training for other amounts of treebank training data. As is partially reflected by the zero probability rules, the addition of the automatically labeled data enables the exploration of a broader parameter space with less danger of over-fitting the data. Also note that the benefit of the automatically labeled data is less clear in the early training stages (i.e., when there are fewer latent annotations), as can be seen in Figure 5.6. This is probably because there is a small number of free parameters and the treebank data is sufficiently large for robust parameter estimation.

5.6 Conclusions

In this chapter, we have studied several ways to enhance a state-of-the-art PCFG-LA parser that are especially useful for less common languages. We presented a heuristic rare word smoothing method to address data-sparsity and a heuristic

language-dependent OOV word handling method to better model OOV words in Chinese. We have also investigated the self-training capability of PCFG-LA parsers through a comparative study with Charniak's parser and showed that PCFG-LA parsers benefit more significantly from self-training than Charniak's parser, even when they are trained on relatively large amounts of treebank training data. We conjecture based on our analyses that the EM training algorithm is able to exploit the information available in both treebank and automatically labeled data to learn more complex grammars while being less affected by over-fitting than when training on the treebank data alone.

We would expect further improvement by combining the PCFG-LA parser with discriminative reranking approaches [33, 68] for self-training. We also expect that self-training would benefit discriminatively trained parsers with latent annotations [124], although training would be much slower compared to using generative models. In the next two chapters, we continue to investigate the issues discussed in this chapter and present two approaches to obtain further improvements in the accuracy of the PCFG-LA grammars: one that takes advantage of the variability among the PCFG-LA grammars to improve model accuracy using multiple grammars together with self-training, and one that is a more principled approach to address data sparsity and handle OOV words using a feature-rich log-linear lexical model.

Chapter 6

Improving PCFG-LA with Self-Training and Product Models

6.1 Overview

The EM algorithm is a local method that may converge to different local maxima when initialized with different random seeds. Indeed, PCFG-LA grammars trained with different random seeds perform differently. Petrov [122] took advantage of this variability among PCFG-LA grammars by parsing with multiple grammars using a model combination method called product model, which is defined formally in Section 6.2, and obtained significant improvement in parsing accuracy. As we discussed in Chapter 5, self-training is an effective method to train accurate parsing models using automatically labeled training data. The natural question to ask is whether self-training and product models are complementary to each other and can be effectively used together.

On the one hand, self-training improves the accuracy of individual grammars, which can be further combined into a product model to achieve even higher performance. On the other hand, the product model has higher accuracies than individual grammars and thus can produce more accurately labeled training data to train more accurate self-trained grammars. In this chapter, we investigate several different training protocols to exploit the complementary effect of self-training and product models and find that the following two factors contribute to the significant

improvements of PCFG-LA grammars:

1. the accuracy of the grammar used to parse the unlabeled data for retraining (single grammar versus product of grammars)

2. the diversity of the grammars that are being combined (self-trained grammars trained using the same automatically labeled subset or different subsets)

We conclude from experiments on both English newswire and English broadcast news that self-training and product models can be effectively combined to create very high quality parsing models.

The rest of this chapter is organized as follows. We briefly describe the product model introduced by Petrov [122] in Section 6.2 and discuss several different training protocols in Section 6.3. We then present experiments and analysis in Section 6.4. Section 6.5 concludes this chapter.

6.2 Product Models

As we discussed in Chapter 5, the EM algorithm is guaranteed to increase the training likelihood at each iteration and can eventually over-fit a PCFG-LA grammar with an increasing number of latent annotations. In addition, the EM algorithm is a local method that may converge to different local maxima when initialized with different random seeds. The smoothing methods described in Section 5.4.1 help to address the over-fitting issue and create models with more robust parameter estimates, but we still observed substantial differences between the learned grammars in Chapter 5 and had to use a development set to select the "best" performing

grammar. However, the best-performing grammar on the development set does not necessarily generate the most accurate parses on the test set [122].

The observation of variation is not surprising; EM's tendency to get stuck in local maxima has been studied extensively in the literature, resulting in various proposals for model selection methods [23]. What is perhaps more surprising is that PCFG-LA grammars trained only using different random seeds seem to capture complementary aspects of the data. As shown in (Petrov [122]), some grammars perform better on some syntactic categories than the other grammars, but there is not a single grammar that performs consistently better on all categories. Quite serendipitously, instead of choosing one grammar and decoding with Equation 5.7, shown below:

$$\hat{T} = \arg\max_{T} \sum_{r \in \hat{\mathcal{R}}(T)} \log \mathrm{P}_{\theta}(r|s)$$

these grammars can be combined into an unweighted product model that substantially outperforms the individual grammars:

$$\hat{T} = \arg\max_{T} \sum_{r \in \hat{\mathcal{R}}(T)} \sum_{G \in \mathcal{G}} \log \mathrm{P}(r|s, G) \tag{6.1}$$

where $\mathcal{G} = \{G_1, \cdots, G_n\}$ denotes the set of random grammars. The product model in Equation 6.1 searches for the parse tree that maximizes the product of the posterior rule probabilities under all of the grammars.

As discussed in Section 5.5.3, self-training of PCFG-LA parsers can mitigate

the data sparsity issue and significantly improve parsing accuracy, but variability still remains in self-trained grammars. Hence, we explore the use of product models together with self-training.

6.3 Training Protocols

In order to investigate the complementary effect of self-training and product models, we consider four training protocols, as illustrated in Figure 6.1.

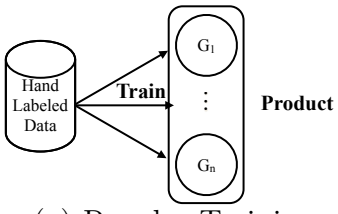
Regular Training Train regular grammars on the treebank data alone. See Figure 6.1 (a).

ST-Reg Training Use a single regular grammar selected based on development set performance to parse a single subset of the unlabeled data and train n self-trained grammars using this single set. See Figure 6.1 (b).

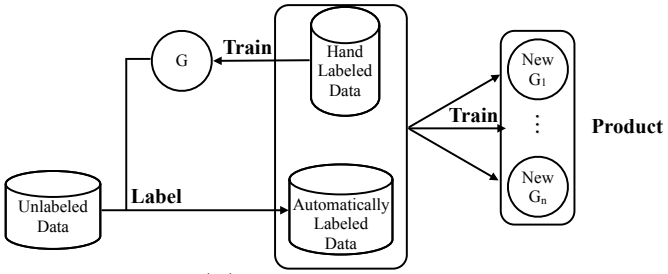
ST-Prod Training Use the product of regular grammars to parse a single subset of the unlabeled data and train n self-trained grammars using this single set. See Figure 6.1 (c).

ST-Prod-Mult Training Use the product of regular grammars to parse all k subsets of the unlabeled data and train k self-trained grammars, each using a different subset. See Figure 6.1 (d).

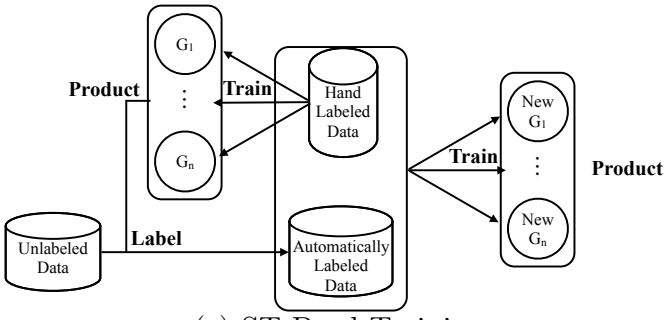
The resulting individual grammars can be either used individually or combined in a product model. We use $n = k = 10$ in our experiments.



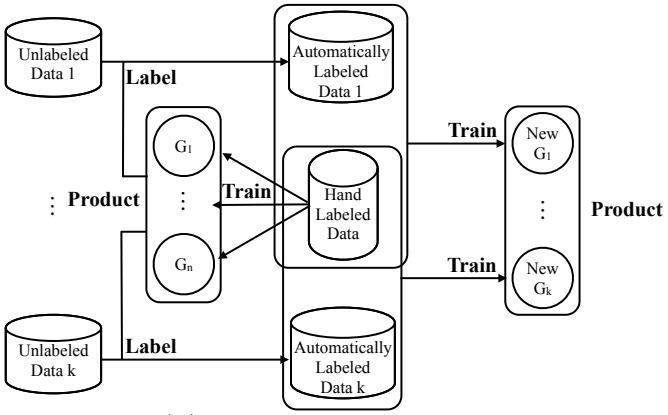
(a) Regular Training



(b) ST-Reg Training



(c) ST-Prod Training



(d) ST-Reg-Mult Training

Figure 6.1: Four training protocols

These four protocols provide different insights. The first two protocols allows us to investigate the effectiveness of product models for regular and standard self-trained grammars. The third protocol enables us to quantify how important the accuracy of the baseline parser is for self-training. Finally, the fourth protocol provides a method for injecting some additional diversity into the individual grammars to determine whether a product model is more successful when there is greater variance among the individual models.

6.4 Experiments

6.4.1 Setup

We conduct experiments on two genres: newswire text and broadcast news transcripts. For the newswire studies, we use the standard setup (sections 02-21 for training, 22 for development, and 23 for final test) of the WSJ Penn treebank [104] for supervised training. The BLLIP corpus [35] is used as the source of unlabeled data for self-training the WSJ grammars. We ignore the parse trees contained in the BLLIP corpus and retain only the sentences, which are already segmented and tokenized for parsing (e.g., contractions are split into two tokens and punctuation is separated from the words). We partition the 1,769,055 BLLIP sentences into 10 equally sized subsets[34].

For broadcast news, we utilize the Broadcast News treebank from Ontonotes

[34]We corrected some of the most egregious sentence segmentation problems in this corpus, and so the number of sentences is different than if one simply pulled the fringe of the trees. It was not uncommon for a sentence split to occur on abbreviations, such as Adm.

[157] together with the WSJ Penn treebank for supervised training because their combination results in better parser models compared to using the limited-sized broadcast news corpus alone (86.7 F vs. 85.2 F). The files in the Broadcast News treebank represent news stories collected during different time periods with a diversity of topics. In order to obtain a representative split of train/development/test sets, we divide them into blocks of 10 files sorted by alphabetical filename order. We use the first file in each block for development, the second for test, and the remaining files for training. This training set is then combined with the entire WSJ Penn treebank. We also use 10 equal size subsets from the Hub4 CSR 1996 utterances [57] for self-training. The Hub 4 transcripts are markedly noisier than the BLLIP corpus is, in part because it was produced by human transcription of spoken language, but also because there is no punctuation indicating sentence boundaries and so sentence segmentation is less precise.

The treebanks are preprocessed differently for the two genres. For newswire, we use a slightly modified version of the WSJ treebank: empty nodes and function labels are deleted and auxiliary verbs are replaced with AUXB, AUXG, AUXZ, AUXD, or AUXN to represent infinitive, progressive, present, past, or past participle auxiliaries[35]. The targeted use of the broadcast models is for parsing broadcast news transcripts for language models in speech recognition systems [53]. Therefore, in addition to applying the transformations used for newswire, we also replace symbolic expressions with verbal forms (e.g., $5 is replaced with five dollars) and remove

[35]Parsing accuracy is marginally affected. The average over 10 SM6 grammars with the transformation is 90.5 compared to 90.4 F without it, a 0.1 F average improvement.

Genre	Stats	Train	Dev	Test	Unlabeled
Newswire	# sentences	45.1k	1.7k	2.4k	1769.1k
	# words	1149.8k	40.1k	56.7k	43057.0k
	Length Avg./Std.	25.5/12.2	25.1/11.8	25.1/12.0	24.3/10.9
Broadcast News	# sentences	59.0k	986	1.1k	4386.5k
	# words	1281.1k	17.1k	19.4k	77687.9k
	Length Avg./Std.	17.3/11.3	17.4/11.3	17.7/11.4	17.7/12.8

Table 6.1: The number of words and sentences, together with average (Avg.) sentence length and its standard deviation (Std.) in our experiments

punctuation and case. The Hub4 data is segmented into utterances, punctuation is removed, words are down-cased, and contractions are tokenized for parsing. Table 7.1 summarizes the data set sizes used in our experiments, together with average sentence length and its standard deviation.

Parses from all models are compared with respective gold standard parses using SParseval bracket scoring [134]. This scoring tool produces scores that are identical to those produced by EVALB for WSJ. For broadcast news, SParseval applies Charniak and Johnson's scoring method [32] for EDITED nodes[36]. Using this method, broadcast news scores were slightly (.05-.1) lower than if EDITED constituents were treated like any other, as in EVALB. We use Dan Bikel's randomized parsing evaluation comparator to determine the significance ($p < 0.05$) of the difference between two parsers' outputs.

Our initial set of experiments and analysis will focus on the development set of WSJ. We will then follow up with an analysis of broadcast news to determine whether the findings generalize to a second, less structured type of data. It is

[36]Non-terminal subconstituents of EDITED nodes are removed so that the terminal constituents become immediate children of a single EDITED node, adjacent EDITED nodes are merged, and they are ignored for span calculations of the other constituents.

Regular	Best	Average	Product
SM6	90.8	90.5	92.0
SM7	90.4	90.1	92.2

Table 6.2: Performance of the regular grammars and their products on the WSJ development set in F score (%)

important to construct grammars capable of parsing this type of data accurately and consistently in order to support structured language modeling [53, 155].

6.4.2 Newswire Results

We compare single grammars and their products that are trained in the standard way on the WSJ treebank training data, as well as the three self-training protocols discussed in Section 6.3. We report the F scores of both SM6 and SM7 grammars on the development set in order to observe the effect of model complexity on the performance of the self-trained and product models. Note that we use 6th round grammars to produce the automatic parse trees for the self-training experiments. Parsing with the product of the SM7 grammars is slow and requires a large amount of memory (32GB). Since we have limited access to machines with this amount of memory, it was infeasible to parse all of the unlabeled data with the product of SM7 grammars.

6.4.2.1 Regular Training

We begin by training ten PCFG-LA grammars initialized with different random seeds using the WSJ treebank training data. Results are presented in Table 6.2.

ST-Reg	Best	Average	Product
SM6	91.5	91.2	92.0
SM7	91.6	91.5	92.4

Table 6.3: Performance of the ST-Reg grammars and their products on the WSJ development set in F score (%)

The best F score attained by the individual SM6 grammars on the development set is 90.8 F, with an average score of 90.5 F. The product of grammars achieves a significantly improved accuracy at 92.0 F. Note that the individual SM7 grammars perform worse on average (90.1 vs. 90.5 F) due to over-fitting, but their product achieves higher accuracy than the product of the SM6 grammars (92.2 vs. 92.0 F). We will further investigate the causes for this effect in Section 6.4.3. Given the ten SM6 grammars from this experiment, we next investigate the three self-training protocols.

6.4.2.2 ST-Reg Training

In the first self-training regime (ST-Reg), we use the best single grammar (90.8 F) chosen based on the development set to parse a single subset of the BLLIP data. We then train ten grammars from different random seeds, using an equally weighted combination of the WSJ training set with this single set. These self-trained grammars are then combined into a product model. As reported in Table 6.3, the use of additional automatically labeled training data enables the individual SM6 ST-Reg grammars to perform significantly better than the individual SM6 grammars (91.2 vs. 90.5 F on average), and the individual SM7 ST-Reg grammars to perform

ST-Prod	Best	Average	Product
SM6	91.7	91.4	92.2
SM7	91.9	91.7	92.4

Table 6.4: Performance of the ST-Prod grammars and their products on the WSJ development set in F score (%)

even better, achieving an average F score of 91.5.

The product of ST-Reg grammars performs significantly better than the individual grammars; however, the improvement is much smaller than that obtained by the product of regular grammars. In fact, the product of ST-Reg grammars performs quite similarly to the product of regular grammars despite the higher average accuracy of the individual grammars. This is probably caused by the fact that self-training on the same data tends to reduce the variability among the self-trained grammars, as we will confirm in Section 6.4.3. The diversity among the individual grammars is an important contributor to the improvements attained by product models.

6.4.2.3 ST-Prod Training

Since products of PCFG-LA grammars perform significantly better than individual PCFG-LA grammars, it is natural to utilize the product model for parsing the unlabeled data. To investigate whether the higher accuracy of the automatically labeled data translates into a higher accuracy of the self-trained grammars, we used the product of SM6 grammars to parse the same subset of the unlabeled data as in the previous experiment. We then trained ten self-trained grammars, which we call

ST-Prod-Mult	Best	Average	Product
SM6	91.7	91.4	92.5
SM7	91.8	91.7	92.8

Table 6.5: Performance of the ST-Prod-Mult grammars and their products on the WSJ development set in F score (%)

ST-Prod grammars. As can be seen in Table 6.4, using the product of the regular grammars for labeling the self-training data results in improved individual ST-Prod grammars when compared with the ST-Reg grammars, with 0.2 and 0.3 improvements for the best SM6 and SM7 grammars, respectively. Interestingly, the best individual SM7 ST-Prod grammar (91.9 F) performs comparably to the product of the regular grammars (92.0 F) that was used to label the BLLIP subset used for self-training. This is very useful for practical reasons because a single grammar is faster to parse with and requires less memory than the product model.

The product of the SM6 ST-Prod grammars also achieves a 0.2 higher F score compared to the product of the SM6 ST-Reg grammars, but the product of the SM7 ST-Prod grammars has the same performance as the product of the SM7 ST-Reg grammars. This could be partially due to the fact that the ST-Prod grammars are no more diverse than the ST-Reg grammars, as we will confirm in Section 6.4.3.

6.4.2.4 ST-Prod-Mult Training

When creating a product model of regular grammars, Petrov [122] used a different random seed for each model and conjectured that the effectiveness of the product grammars stems from the resulting diversity of the individual grammars.

Two ways to systematically introduce bias into individual models are to either modify the feature sets [3, 140] or to change the training distributions of the individual models [16]. Petrov [122] attempted to use the second method to train individual grammars on either disjoint or overlapping subsets of the treebank, but observed a performance drop in individual grammars resulting from training on less data, as well as in the performance of the product model. Rather than reducing the amount of gold training data (or having treebank experts annotate more data to support the diversity), we employ the self-training paradigm to train models using a combination of the same gold training data with different sets of the self-labeled training data, each of which has the same size of the single set used in the other self-training protocols. This approach also allows us to utilize a much larger amount of low-cost automatically labeled data than can be used to train one model[37] by partitioning the data into smaller subsets and then training models with individual subsets. Hence, in the fourth training protocol, we use the product of the regular grammars to parse all ten subsets of the unlabeled data and train ten grammars, which we call ST-Prod-Mult grammars, each using a different subset.

As shown in Table 6.5, the individual ST-Prod-Mult grammars perform similarly to the individual ST-Prod grammars. However, the product of the ST-Prod-Mult grammars achieves significantly higher accuracies than the product of the ST-Prod grammars, with 0.3 and 0.4 improvements in F score for SM6 and SM7 grammars, respectively, suggesting that the use of multiple self-training subsets

[37]The amount of automatically labeled training data used to self-train a PCFG-LA grammar is constrained by CPU and memory because self-training PCFG-LA grammars on large quantities of automatically labeled training data can be very slow and would require a lot of memory.

plays an important role in model combination.

6.4.3 Analysis

We conducted a series of analyses to develop an understanding of the factors affecting the effectiveness of combining self-training with product models.

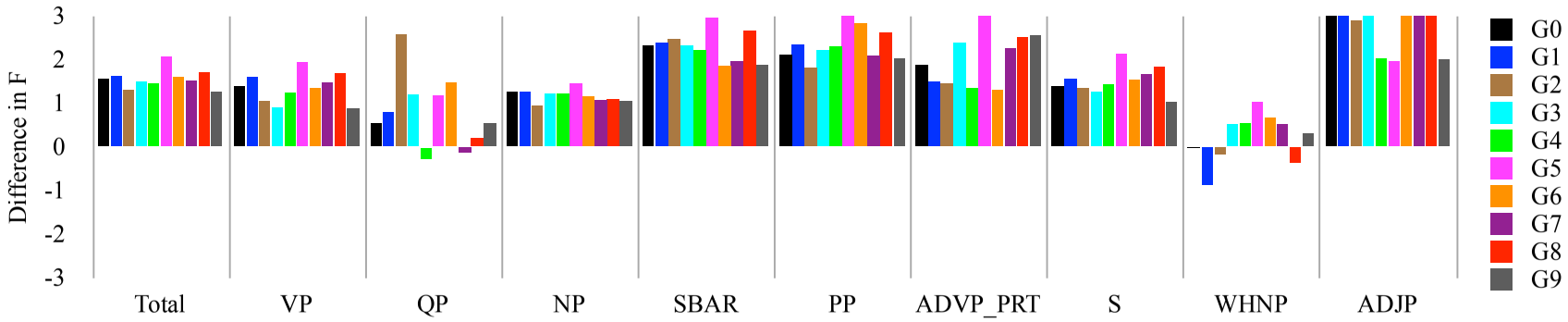
6.4.3.1 What Has Improved?

Figure 6.2 (a) depicts the difference between the product and the individual SM6 regular grammars on overall F score, as well as individual constituent F scores. As can be observed, there is significant variation among the individual grammars, and the product of the regular grammars improves almost all categories, with a few exceptions (some individual grammars do better on QP and WHNP constituents).

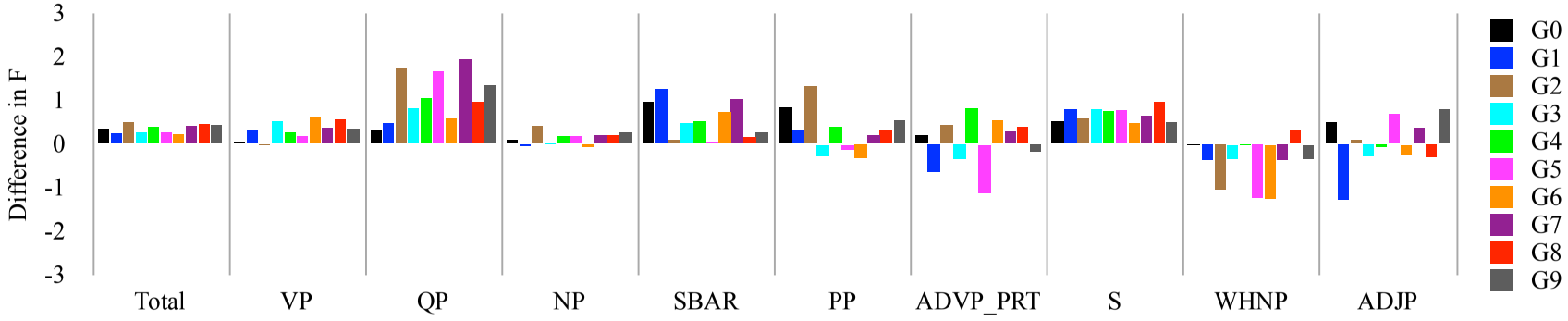
Figure 6.2 (b) depicts the difference between the product of the SM6 regular grammars and the individual SM7 ST-Prod-Mult grammars. Self-training dramatically improves the quality of the single SM7 ST-Prod-Mult grammars. In most of the categories, some individual ST-Prod-Mult grammars perform comparably or slightly better than the product of SM6 regular grammars used to automatically label the unlabeled training set.

6.4.3.2 Over-Fitting vs. Smoothing

Figure 6.3 (a) and 6.3 (b) depict the learning curves of the regular and the ST-Prod-Mult grammars. As more latent variables are introduced through the iterative



(a) Difference in F score between the product of SM6 regular grammars and the individual SM6 regular grammars.



(b) Difference in F score between the product of SM6 regular grammars and the individual SM7 ST-Prod-Mult grammars.

Figure 6.2: Difference in F scores between various individual grammars and representative product grammars. Each individual grammar is represented by a unique color.

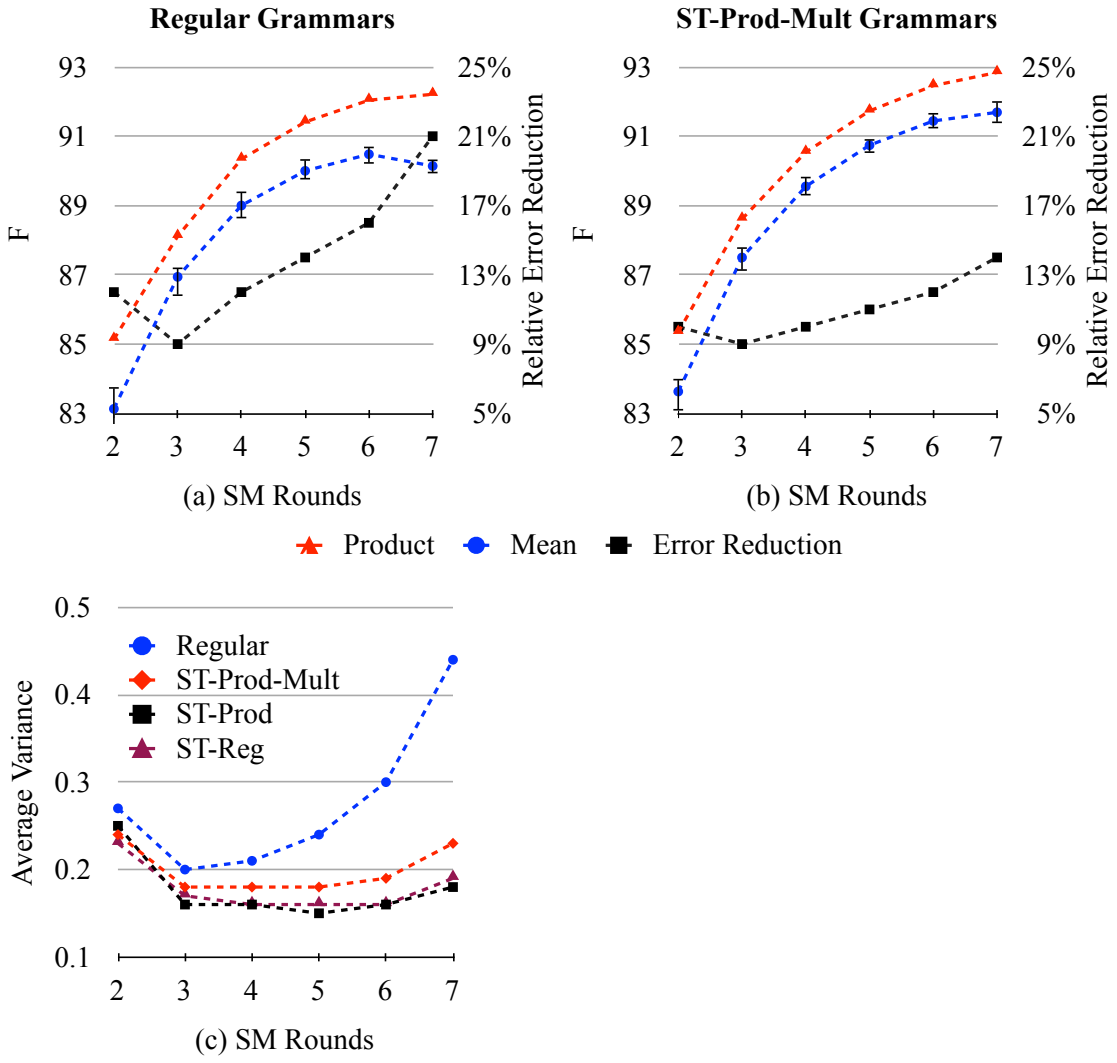


Figure 6.3: Learning curves of (a) the individual regular and (b) ST-Prod-Mult grammars (average performance, with minimum and maximum values indicated by bars) and their products before and after self-training on the WSJ development set. The relative error reductions of the products are also reported. The measured average empirical variance among the grammars trained on WSJ is reported in (c).

split-merge training algorithm, the modeling capacity of the grammars increases, leading to improved performance. However, the performance of the regular grammars drops after 6 split-merge rounds, as was also previously observed in (Huang and Harper [71], Petrov [121]), suggesting that the regular SM7 grammars have over-fit the treebank training data. In contrast, the performance of the self-trained grammars continues to improve on the 7th split-merge round. As discussed in Chapter 5, this improvement may be due to the fact that the additional self-labeled training data adds a smoothing effect to the grammars, supporting an increase in model complexity without over-fitting.

Although the product model consistently helps both regular and self-trained grammars, there is a non-negligible difference between the improvement achieved by the product model over its component grammars. The regular product model improves upon its individual grammars more than the ST-Prod-Mult product does in the later split-merge rounds, as illustrated by the relative error reduction curves in Figures 6.3 (a) and (b). In particular, the product of the SM7 regular grammars gains a remarkable 2.1 absolute improvement in F over the average performance of the individual regular SM7 grammars and a 0.2 absolute improvement in F over the product of the regular SM6 grammars, despite the fact that the individual regular SM7 grammars perform less accurately than the SM6 grammars. This suggests that the product model is able to effectively exploit less smooth, over-fit grammars. We will further examine this issue next.

6.4.3.3 Diversity

For effectiveness of Products of Experts [67] or Logarithmic Opinion Pools [141], it is important that each individual expert learns complementary aspects of the training data. The product model [122] enforces that the joint prediction of their product has to be licensed by all individual experts. One possible explanation of the high accuracy achieved by a product of over-fit grammars is that with the addition of more latent annotations, the individual grammars become more deeply specialized on certain aspects of the training data. This specialization leads to greater diversity in their prediction preferences, especially in the presence of a small training set. On the other hand, the self-labeled training set size is much larger, and so the specialization process is therefore slowed down.

Petrov [122] showed that the individually learned grammars are indeed very diverse by looking at the distribution of latent annotations across the treebank categories, as well as the variation in overall and individual category F scores (e.g., see Figure 6.2). However, these measures do not directly relate to the diversity of the prediction preferences of the grammars, as we have observed similar patterns in the regular and self-trained models.

Given a sentence s and a set of grammars $\mathcal{G} = \{G_1, \cdots, G_n\}$, recall that parsing with a product model searches for the best tree T such that the following objective function is maximized:

$$\sum_{r \in \hat{\mathcal{R}}(T)} \sum_{G \in \mathcal{G}} \log \mathrm{P}(r|s, G)$$

where $\log \mathrm{P}(r|s,G)$ is the log posterior probability of rule r given sentence s and grammar G. The power of the product model comes directly from the diversity in $\log \mathrm{P}(r|s,G)$ among individual grammars. If there is little diversity, the individual grammars would make similar predictions and there would be little or no benefit from using a product model. We use the average empirical variance of the log posterior probabilities of the rules among the learned grammars on a held-out set S as an approximate measure of the diversity among the grammars:

$$\frac{\sum_{s\in S}\sum_{G\in\mathcal{G}}\sum_{r\in\hat{\mathcal{R}}(G,s)}\mathrm{P}(r|s,G)\mathrm{VAR}(\log(\mathrm{P}(r|s,\mathcal{G})))}{\sum_{s\in S}\sum_{G\in\mathcal{G}}\sum_{r\in\hat{\mathcal{R}}(G,s)}\mathrm{P}(r|s,G)}$$

where $\hat{\mathcal{R}}(G,s)$ represents the set of rules extracted from the chart when parsing sentence s with grammar G, and $\mathrm{VAR}(\log(\mathrm{P}(r|s,\mathcal{G})))$ is the variance of $\log(\mathrm{P}(r|s,G))$ among all grammars $G \in \mathcal{G}$.

Note that the average empirical variance is only an approximate of the diversity among grammars. It tends to be biased to produce larger numbers when the posterior probabilities of rules are smaller because small differences in probability produce large changes in the log scale. This happens for coarser grammars produced in early split-merge stages because there is more uncertainty about what rules to apply and thus many low probability rules remain in the parsing chart.

As shown in Figure 6.3 (c), the average variances all start at a high value and then drop, probably due to the aforementioned bias. However, as the split-merge iterations continue, the average variances increase despite the bias. More

interestingly, the variance among the regular grammars grows at a much faster rate and is consistently greater when compared to the self-trained grammars. This suggests that there is greater diversity among the regular grammars than among the self-trained grammars, and it explains the larger absolute improvement obtained by the regular product model. It is also important to note that there is more variance among the ST-Prod-Mult grammars, which were trained on disjoint self-labeled training data, and a greater improvement in their product model relative to the ST-Reg and ST-Prod grammars, further supporting the diversity hypothesis. Finally, the trend seems to indicate that the variance of the self-trained grammars would continue increasing if EM training was extended by a few more split-merge rounds, potentially resulting in even better product models. It is currently impractical to test this due to the dramatic increase in computational requirements for an SM8 product model, and so we leave it for future work.

6.4.4 Broadcast News Results

We conducted a similar set of experiments on the broadcast news data set[38]. While the development set results in Table 6.6 show trends similar to the WSJ results, the benefit from the combination of self-training and product models is more pronounced in this domain. The best single ST-Prod-Mult grammar (89.2 F) alone is able to outperform the product of SM7 regular grammars (88.9 F), and

[38]We chose to not run the full ST-Reg experiment on broadcast news in order to focus our computing resources on the ST-Prod and ST-Prod-Mult experiments for achieving high-quality parsing models on broadcast news. As a result, we do not report the ST-Reg results. However, our preliminary experiments showed that the broadcast news ST-Reg grammars are indeed more accurate than the regular grammars but less accurate than the ST-Prod grammars, similar to the observation on the newswire data set.

Model	Rounds	Best	Product
Regular	SM6	87.1	88.6
	SM7	87.1	88.9
ST-Prod	SM6	88.5	89.0
	SM7	89.0	89.6
ST-Prod-Mult	SM6	88.8	89.5
	SM7	**89.2**	**89.9**

Table 6.6: F scores (%) for various models on the broadcast news development set

their product achieves another 0.7 absolute improvement, resulting in a significantly better accuracy of 89.9 F.

Figure 6.4 shows again that the benefits of self-training and product models are complementary and can be stacked. As can be observed, the self-trained grammars have increasing F scores as the split-merge rounds increase, while the regular grammars have a slight decrease in F score after round 6. In contrast to the newswire models, it appears that the individual ST-Prod-Mult grammars trained on broadcast news always perform comparably to the product of the regular grammars at all split-merge rounds, including the product of SM7 regular grammars. This is noteworthy, given that the ST-Prod-Mult grammars are trained on the output of the product of the SM6 regular grammars, which are less accurate than the product of SM7 regular grammars. Although we used more treebank trees for training the initial broadcast news grammars, the greatest number of trees are taken from the WSJ treebank, which is different in many ways than broadcast news. It is possible that the addition of the genre-matched automatically labeled broadcast news data (from Hub4 CSR) provides some adaptation guidance that enables learning of en-

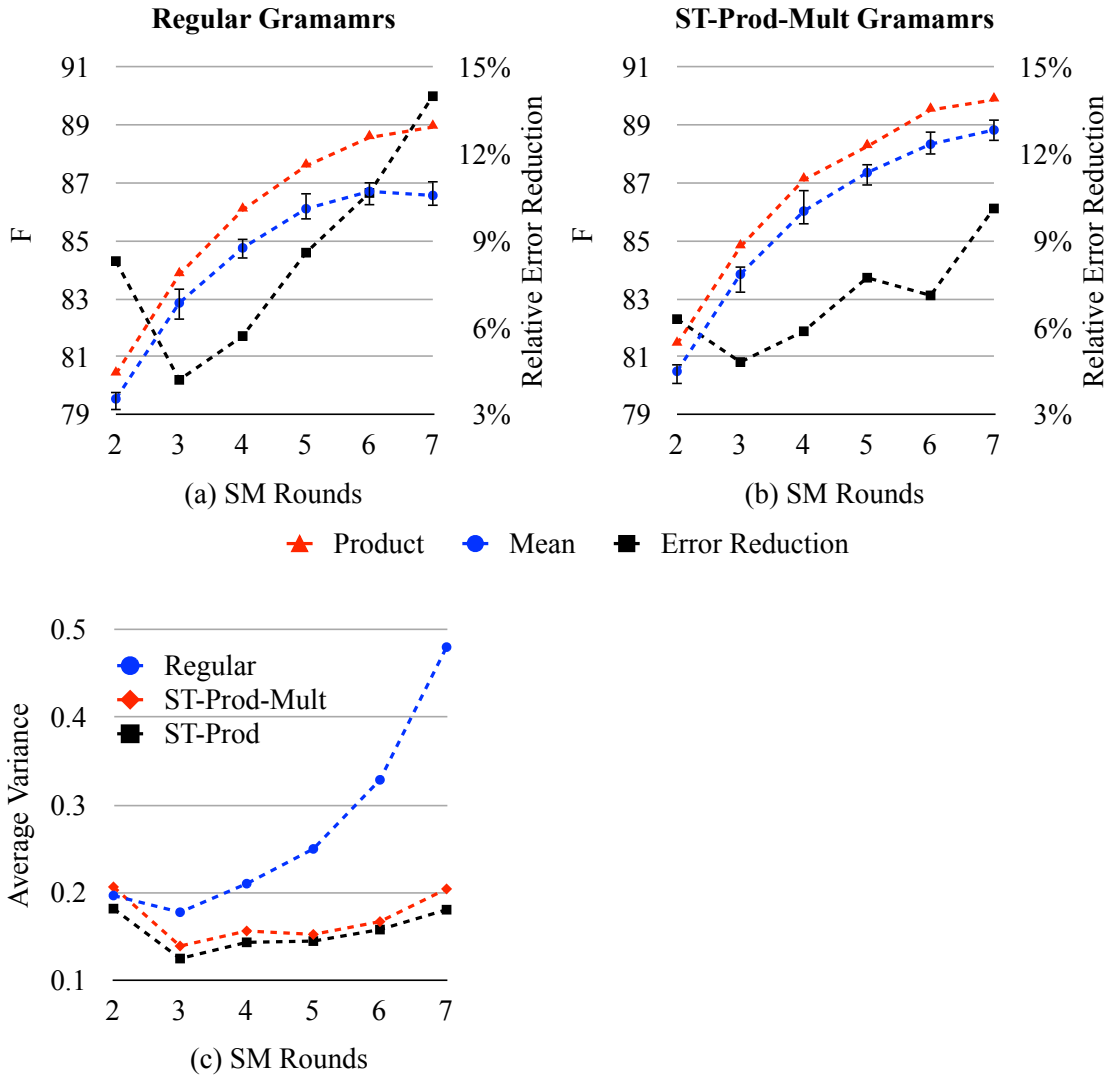


Figure 6.4: Learning curves of (a) the individual regular and (b) ST-Prod-Mult grammars (average performance, with minimum and maximum values indicated by bars) and their products before and after self-training on the broadcast news development set. The relative error reductions of the products are also reported. The measured average empirical variance among the grammars trained on broadcast news is reported in (c).

hanced grammars for broadcast news. The product of the ST-Prod-Mult grammars also provides a further and significant improvement in F score.

6.4.5 Final Results

We evaluated the best single self-trained grammar (SM7 ST-Prod), as well as the product of the SM7 ST-Prod-Mult grammars on the WSJ test set. Table 6.7 compares these two grammars to a large body of related parsing results grouped into single parsers (SINGLE), discriminative reranking approaches (RE), self-training (SELF), and system combinations (COMBO).

Our best single grammar achieves an accuracy that is only slightly worse (91.6 vs. 91.8 in F score) than the product model in (Petrov [122]). This is made possible by self-training using the output of a high quality product model. The higher quality of the automatically parsed data results in a 0.3 point higher final F score (91.6 vs. 91.3) over the self-training results in (Huang and Harper [71]), which used a single regular grammar for parsing the unlabeled data. The product of the self-trained ST-Prod-Mult grammars achieves significantly higher accuracies with an F score of 92.5, a 0.7 improvement over the product model in (Petrov [122]).

Although our models are based on purely generative PCFG grammars, our best product model performs competitively to the self-trained two-step discriminative reranking parser of McClosky et al. [107], which makes use of many non-local reranking features. Our parser also performs comparably to other system combina-

[8]Our ST-Reg grammars are trained in the same way as in (Huang and Harper [71]) and Chapter 5 except that we keep all unary rules. The reported numbers are from the best single ST-Reg grammar in this work.

Type	Parser		LP	LR	EX
SINGLE	Charniak [31]		89.9	89.5	37.2
	Petrov and Klein [123]		90.2	90.1	36.7
	Carreras et al. [26]		91.4	90.7	-
RE	Charniak and Johnson [33]		91.8	91.2	44.8
	Huang [68]		92.2	91.2	43.5
SELF	[8]Huang and Harper [71]		91.6	91.1	40.4
	McClosky et al. [107]		92.5	92.1	45.3
COMBO	Petrov [122]		92.0	91.7	41.9
	Sagae and Lavie [136]		93.2	91.0	-
	Fossum and Knight [55]		93.2	91.7	-
	Zhang et al. [167]		93.3	92.0	-
	ST-Prod-Mult	Single Grammar	91.8	91.4	40.3
		Product Model	92.7	92.2	43.1

Table 6.7: Final test set accuracies on WSJ

tion approaches [55, 136, 167] with a higher recall and a lower precision, but again without using a discriminative reranking step. We expect that replacing the first-step generative parsing model in (McClosky et al. [107]) with a product of latent variable grammars would produce even higher parsing accuracies.

On the broadcast news test set, our best performing single and product grammars (bolded in Table 6.6) obtained F scores of 88.7 and 89.6, respectively. While there is no prior work using our setup, we expect these numbers to set a high baseline.

6.5 Conclusions

In this chapter, we extended the self-training approach described in Chapter 5 to build high accuracy products of PCFG-LA grammars with large amounts of genre-matched data. We demonstrated empirically on newswire and broadcast news genres

that very high accuracies can be achieved by self-training products of PCFG-LA grammars on disjoint sets of automatically labeled data. Two factors appear to play an important role. First, the accuracy of the model used for parsing the unlabeled data contributes directly to the accuracy of the resulting individual self-trained grammars. Second, the diversity of the individual grammars affects the gains that can be obtained when combining multiple grammars into a product model. Our most accurate single grammar achieves an F score of 91.6 on the WSJ test set, rivaling discriminative reranking approaches [33] and products of latent variable grammars [122], despite being a single generative PCFG. Our most accurate product model achieves an F score of 92.5 without the use of discriminative reranking and comes close to the best known numbers on this test set [167].

While self-training allows more training data to be exploited during grammar training, it does not change the way a grammar learns from the training data. In the next chapter, we will present a feature-rich log-linear lexical model for PCFG-LA grammars that not only provides a principled solution to address data sparsity and OOV words, but also a means to learn from arbitrary local features that are helpful for further improving parsing performance.

Chapter 7

Improving PCFG-LA with Log-Linear Lexical Models

7.1 Overview

Context-free grammars with latent annotations (PCFG-LA) are effective for parsing a variety of languages; however, as we discussed in Chapter 5, their lexical model may be subject to over-fitting and requires language engineering to handle OOV words. The rare word smoothing and OOV word handling methods studied in Chapter 5 were effective at improving parsing performance; however, both approaches may be suboptimal due to fact that they are based on heuristics, and expert tailoring of a language-specific OOV module may also be unavailable for new languages.

Inspired by the previous studies [8, 37] that have incorporated rich features into generative models, we have developed a feature-rich log-linear lexical model that can be easily adapted to new languages. Our lexical model has three advantages: over-fitting is alleviated via regularization, OOV words are modeled using rich features, and lexical features are exploited during grammar induction. Our approach results in significantly more accurate PCFG-LA grammars that are easy to apply to different languages and achieve significant absolute improvements of 1%, 1.7%, and 2.7% in F score on English, Chinese, and Arabic, respectively.

The rest of this chapter is structured as follows. We first review the related lit-

erature and motivate the use of feature rich models in Section 7.2. We then describe our proposed feature rich log-linear lexical model in Section 7.3 and the training algorithm in Section 7.4. Experiments are presented in Section 7.5. Section 7.6 concludes this chapter.

7.2 Motivation for Using Feature Rich Models

The lexical model of a standard PCFG-LA grammar is represented as a generative multinomial distribution $\mathrm{P}_\theta(w|t_x)$ that only models the word identity itself, and so is incapable of taking advantage of the rich morphological features of the word. Moreover, it is not explicitly regularized and may be subject to over-fitting. The rare word smoothing method and OOV handling method introduced in Chapter 5 partially address these problems to improve parsing performance; however, both are heuristic and are likely to be suboptimal. A more principled model should be able to learn in a data-driven fashion from over-lapping rich features in order to handle data sparsity and model OOV words.

Some previous research [54, 124] addressed this problem by completely replacing the generative models with a discriminative model, so that overlapping features can be incorporated. For discriminative log-linear grammars with latent variables [124, 125], the conditional probability of a parse tree T given a input sentence s is parametrized by the following log-linear model:

$$\mathrm{P}_\lambda(T|s) = \sum_{T' \in \mathcal{Z}(T)} \mathrm{P}_\lambda(T'|s) = \sum_{T' \in \mathcal{Z}(T)} \frac{\exp\langle \lambda, \mathbf{f}(T') \rangle}{Z(\lambda, s)} \tag{7.1}$$

where $\mathcal{Z}(T)$ is the set of derivation trees of T, $\mathbf{f}(T')$ is a feature vector extracted from the derivation tree T', λ is the vector of feature weights, $\langle \cdot, \cdot \rangle$ represents the dot product of two vectors, and $Z(\lambda, s)$ is the partition function that ensures that $\mathrm{P}_\lambda(T|s)$ is a proper distribution over all derivation trees for sentence s. In the lexical part of the discriminative grammar, Petrov and Klein [124] used simple prefixes and suffixes of up to length 3 in addition to universal features such as the presence of digits and dashes to handle OOV words. An advantage of applying this type of model to new languages is that the focus is on feature-engineering rather than how to develop a heuristic method that is appropriate for the language. On the other hand, discriminative training is much slower than generative training because the partition function $Z(\lambda, s)$ requires an expensive summation over all possible derivations trees of all possible parse trees over a sentence. Moreover, as shown in (Petrov and Klein [124]), a discriminatively trained parser is less accurate than its strong generative counterpart on WSJ. Nevertheless, it does perform better on some other languages, possibly due to the use of regularization and multi-scale grammars to alleviate data sparsity and rich features to improve OOV word handling.

Berg-Kirkpatrick et al. [8] recently demonstrated that log-linear models with rich features can be successfully applied to unsupervised learning of generative models such as HMM POS taggers. In an unsupervised generative learning task, the joint probability of a sequence of observed random variables $\mathbf{Y}$ are computed with

reference to a sequence of hidden random variables $\mathbf{Z}$,

$$\mathrm{P}(\mathbf{Y} = \mathbf{y}) = \sum_{\mathbf{z} \in \mathcal{Z}(\mathbf{y})} \mathrm{P}(\mathbf{Z} = \mathbf{z}, \mathbf{Y} = \mathbf{y}) \tag{7.2}$$

Let $\mathbf{X} = (\mathbf{Z}, \mathbf{Y})$. The joint probability of $\mathrm{P}(\mathbf{X})$ can be factored into the product of conditional probabilities:

$$\mathrm{P}(\mathbf{X} = \mathbf{x}) = \prod_{i \in I} \mathrm{P}_\lambda(X_i = x_i | X_{\pi(i)} = x_{\pi(i)}) \tag{7.3}$$

where $X_{\pi(i)}$ denotes the parents of X_i and I is the set of indexes of the variables in $\mathbf{X}$. In an HMM-based unsupervised POS tag learning task, if x_i represents a word, then $x_{\pi(i)}$ represents the hidden POS tag that generates the word. Instead of modeling the conditional probability $\mathrm{P}_\lambda(X_i = x_i | X_{\pi(i)} = x_{\pi(i)})$ using a multinomial distribution, as is commonly done in natural language processing tasks, Berg-Kirkpatrick et al. [8] treated the distribution as the output of a local log-linear model:

$$\mathrm{P}_\lambda(X_i = d | X_{\pi(i)} = c) = \mathrm{P}_\lambda(d|c) = \frac{\exp\langle \lambda, \mathbf{f}(d, c) \rangle}{\sum_{d'} \exp\langle \lambda, \mathbf{f}(d', c) \rangle} \tag{7.4}$$

and integrated rich features into the model learning process by optimizing on the regularized likelihood of the training data:

$$L(\lambda) = \log \mathrm{P}_\lambda(\mathbf{Y} = \mathbf{y}) - \kappa ||\lambda||^2$$

This approach was found to significantly outperform the traditional generative mod-

els on several unsupervised learning tasks.

Inspired by the work of Berg-Kirkpatrick et al. [8], we propose a locally normalized log-linear lexical model for generative PCFG-LA grammars. Our log-linear lexical model maintains the advantages of generative models, while providing a principled way to: 1) alleviate over-fitting via regularization, 2) handle OOV words using rich features, and 3) incorporate lexical features into grammar induction.

7.3 Design of the Log-Linear Lexical Model

When designing a log-linear lexical model based on the work of Berg-Kirkpatrick et al. [8], someone might try to model $\mathrm{P}_\theta(w|t_x)$, where θ is the full grammar parameter, directly using a log-linear model:

$$\mathrm{P}_\phi(w|t_x) = \frac{\exp\langle\phi, \mathbf{f}(t_x, w)\rangle}{\sum_{w'} \exp\langle\phi, \mathbf{f}(t_x, w')\rangle} \tag{7.5}$$

where $\mathbf{f}(t_x, w)$ represents the feature vector extracted from the pair (t_x, w), ϕ represents the feature weight vector, and the denominator sums over all words that appear on the training data. Note that θ is the parameter set presenting all of the free parameters of the grammar that must be tuned, but as a convention that we will apply throughout this chapter, we replace the subscript θ in $\mathrm{P}_\theta(w|t_x)$ with ϕ in $\mathrm{P}_\phi(w|t_x)$ to emphasize that the parameter ϕ in θ fully controls the emission probability of word w given latent tag t_x.

The model formulation in Equation 7.5 does not have the ability to estimate the probability of OOV words given a latent tag; however, for parsing, we must

model OOV words that will undoubtedly appear in new sentences. One might compute the numerator for an OOV word based on its features and divide it by a denominator approximated using the words in the training data, but such an estimate did not perform well in our preliminary experiments.

An alternative is to model $\mathrm{P}_\theta(t_x|w)$ directly using a log-linear model:

$$\mathrm{P}_\phi(t_x|w) = \frac{\exp\langle\phi, \mathbf{f}(t_x, w)\rangle}{\sum_{t'}\sum_{x'}\exp\langle\phi, \mathbf{f}(t'_{x'}, w)\rangle}$$

and compute $\mathrm{P}_\theta(w|t_x)$ via Bayes' rule. However, such a model cannot guarantee that the probability $\mathrm{P}_\theta(t|w)$ computed by $\sum_x \mathrm{P}_\theta(t_x|w)$ is equal to the maximum likelihood estimate, which is a reasonable constraint. Moreover, it is difficult to initialize the feature weights to reflect the prior knowledge that each word has its own preferences for POS tags. Simply setting the initial weights to zero is not a good choice because it forces the conditional distribution of latent tags to be uniform across the POS tags.

Hence, we choose to model $\mathrm{P}_\theta(w|t_x)$ in multiple steps. We first model the conditional probability of latent tag t_x given the surface POS tag t and word w using a log-linear model:

$$\mathrm{P}_\phi(t_x|t, w) = \frac{\exp\langle\phi, \mathbf{f}(t_x, w)\rangle}{\sum_{x'}\exp\langle\phi, \mathbf{f}(t_{x'}, w)\rangle} \tag{7.6}$$

where $\mathbf{f}(t_x, w)$ represents the feature vector extracted from the pair (t_x, w), ϕ is the feature weight vector, and the denominator sums over all latent tags for POS tag t.

This model is applicable to both known and OOV words as long as there are active features; otherwise, a uniform latent tag distribution is assumed. We call $\mathrm{P}_\phi(t_x|t,w)$ of Equation 7.6 the *latent lexical model* because it models the distribution of latent tags.

The conditional probability of t_x given word w is expressed as:

$$\mathrm{P}_\theta(t_x|w) = \mathrm{P}_\theta(t_x, t|w) = \mathrm{P}_\phi(t_x|t,w)\mathrm{P}(t|w)$$

and the word emission probability given a latent tag is computed via Bayes' rule:

$$\mathrm{P}_\theta(w|t_x) = \frac{\mathrm{P}_\phi(t_x|t,w)\mathrm{P}(t|w)\mathrm{P}(w)}{\sum_{w'} \mathrm{P}_\phi(t_x|t,w')\mathrm{P}(t|w')\mathrm{P}(w')} \tag{7.7}$$

Our new lexical model $\mathrm{P}_\theta(w|t_x)$ shown in Equation 7.7 is composed of the latent lexical model $\mathrm{P}_\phi(t_x|t,w)$ and two other terms: $\mathrm{P}(t|w)$ and $\mathrm{P}(w)$, which are computed differently for known and OOV words.

For words observed in the training data, both $\mathrm{P}(t|w)$ and $\mathrm{P}(w)$ are computed using the maximum-likelihood estimation (based on the observed training trees) so that $\mathrm{P}_\theta(w|t_x)$ forms a proper distribution of observed words during grammar induction.

For OOV words, we use a *log-linear OOV model* to estimate the POS tag distribution:

$$\mathrm{P}_\gamma(t|w) = \frac{\exp\langle\gamma, \mathbf{g}(t,w)\rangle}{\sum_{t'} \exp\langle\gamma, \mathbf{g}(t',w)\rangle} \tag{7.8}$$

where $\mathbf{g}(t,w)$ represents the feature vector extracted from the pair (t,w), γ is the

feature weight vector, and the denominator sums over all POS tags with active features. The *simple* approach (described in Section 5.4.2) is used when no feature is active. $\mathrm{P}(w)$ is approximated by one over the number of training tokens. It should be noted that $\mathrm{P}_\gamma(t|w)$ may use different features than $\mathrm{P}_\phi(t_x|t,w)$.

Compared with modeling $\mathrm{P}_\theta(w|t_x)$ directly as a multinomial distribution, the new lexical model separates $\mathrm{P}(t|w)$ and $\mathrm{P}_\phi(t_x|t,w)$, offering three important advantages:

- The parameter ϕ of the latent lexical model $\mathrm{P}_\phi(t_x|t,w)$ can be smoothed through regularization to address data sparsity.
- Rich features can be utilized in the OOV model $\mathrm{P}_\gamma(t|w)$ to estimate POS tag distributions for OOV words. This is important when working on new languages as it provides a rich and unified model to address OOV words.
- Rich features can also be utilized in the latent lexical model $\mathrm{P}_\phi(t_x|t,w)$ to guide the induction of latent POS tags.

7.4 Model Training

The parameter θ for our parser model is a tuple consisting of ϕ for the log-linear latent lexical model, γ for the log-linear OOV model, and ψ for the phrasal rule expansion probabilities. The other parameters (e.g., $\mathrm{P}(t|w)$ and $\mathrm{P}(w)$ for known words and $\mathrm{P}(\mathit{rare}|t_x)$) can be computed based on observable or fractional counts once θ is determined.

Because the parameter γ of the OOV model is independent of the latent categories, we simply use a gradient-based optimization approach to maximize the following objective:

$$l'(\gamma) = \sum_{t,w} c_{t,w} \log \mathrm{P}_\gamma(t|w) - \kappa' ||\gamma||^2$$

where $c_{t,w}$ is the count of the pair (t, w) in the training data, and κ' is the regularization weight.

For parameters ψ and ϕ, we apply the split-merge training procedure in (Petrov et al. [126]) to induce latent categories. Given a set of latent categories, the goal is to find θ that maximizes the regularized training likelihood:

$$L(\theta) = \sum_{T \in \mathcal{T}} \log \mathrm{P}_\theta(T) - \kappa ||\phi||^2 \tag{7.9}$$

where $\kappa ||\phi||^2$ is the regularization term[40] for the feature weights of the latent lexical model.

The two optimization approaches described in (Berg-Kirkpatrick et al. [8]) can be extended naturally to our problem. The first approach is EM-based with an E-step identical to Equation 5.3 in Section 5.3. The objective of the M-step

[40]Both κ' and κ are tuned on the development set. We could have also used L1 regularization, but leave this alternative to future work.

becomes:

$$
\begin{aligned}
l(\theta) &= \sum_{w \to t_x \in \mathcal{R}_l} e_{t_x,w} \log \mathrm{P}_\phi(w|t_x) - \kappa||\phi||^2 \\
&\quad + \sum_{r \in \mathcal{R}_p} e_r \log \mathrm{P}_\psi(r)
\end{aligned}
$$

where we separate the set of rules $\mathcal{R}$ into lexical rules $\mathcal{R}_l$ and phrasal rules $\mathcal{R}_p$. Note that we replace the subscript θ in $\mathrm{P}_\theta(w|t_x)$ (Equation 7.7) with ϕ in $\mathrm{P}_\phi(w|t_x)$ because ϕ fully controls the emission probability of words on the training data. The phrasal rule parameter ψ is updated as before by normalizing the expected rule counts and is smoothed in the same way as in (Petrov et al. [126]). Let $l(\phi)$ be the part of $l(\theta)$ that depends on ϕ, i.e.,

$$
l(\phi) = \sum_{w \to t_x \in \mathcal{R}_l} e_{t_x,w} \log \mathrm{P}_\phi(w|t_x) - \kappa||\phi||^2 \tag{7.10}
$$

The gradient of $l(\phi)$ with respect to ϕ can be computed as follows. We first compute the derivative of $\mathrm{P}_\phi(t_x|t,w)$ (Equation 7.6) with respect to the i-th parameter ϕ_i:

$$
\begin{aligned}
\frac{\partial}{\partial \phi_i} \mathrm{P}_\phi(t_x|t,w) &= \frac{\partial}{\partial \phi_i} \frac{\exp\langle \phi, \mathbf{f}(t_x,w)\rangle}{\sum_{x'} \exp\langle \phi, \mathbf{f}(t_{x'},w)\rangle} \\
&= \frac{\exp\langle \phi, \mathbf{f}(t_x,w)\rangle \mathbf{f}_i(t_x,w)}{\sum_{x'} \exp\langle \phi, \mathbf{f}(t_{x'},w)\rangle} \\
&\quad - \frac{\exp\langle \phi, \mathbf{f}(t_x,w)\rangle}{\sum_{x'} \exp\langle \phi, \mathbf{f}(t_{x'},w)\rangle} \frac{\sum_{x'} \exp\langle \phi, \mathbf{f}(t_{x'},w)\rangle \mathbf{f}_i(t_{x'},w)}{\sum_{x'} \exp\langle \phi, \mathbf{f}(t_{x'},w)\rangle} \\
&= \mathrm{P}_\phi(t_x|t,w)(\mathbf{f}_i(t_x,w) - \sum_{x'} \mathrm{P}_\phi(t_{x'}|t,w)\mathbf{f}_i(t_{x'},w))
\end{aligned} \tag{7.11}
$$

then the derivative of $\mathrm{P}_\phi(w|t_x)$ (Equation 7.7) with respect to ϕ_i can be computed

as:

$$\begin{aligned}
\frac{\partial}{\partial \phi_i}\mathrm{P}_\phi(w|t_x) &= \frac{\partial}{\partial \phi_i}\frac{\mathrm{P}_\phi(t_x|t,w)\mathrm{P}(t|w)\mathrm{P}(w)}{\sum_{w'}\mathrm{P}_\phi(t_x|t,w')\mathrm{P}(t|w')\mathrm{P}(w')} \\
&= \frac{\frac{\partial}{\partial \phi_i}\mathrm{P}_\phi(t_x|t,w)\mathrm{P}(t|w)\mathrm{P}(w)}{\sum_{w'}\mathrm{P}_\phi(t_x|t,w')\mathrm{P}(t|w')\mathrm{P}(w')} \\
&\quad -\frac{\mathrm{P}_\phi(t_x|t,w)\mathrm{P}(t|w)\mathrm{P}(w)}{\sum_{w'}\mathrm{P}_\phi(t_x|t,w')\mathrm{P}(t|w')\mathrm{P}(w')}\frac{\sum_{w'}\frac{\partial}{\partial \phi_i}\mathrm{P}_\phi(t_x|t,w')\mathrm{P}(t|w')\mathrm{P}(w')}{\sum_{w'}\mathrm{P}_\phi(t_x|t,w')\mathrm{P}(t|w')\mathrm{P}(w')} \\
&= \mathrm{P}_\phi(w|t_x)(\mathbf{f}_i(t_x,w) - \sum_{x'}\mathrm{P}_\phi(t_{x'}|t,w)\mathbf{f}_i(t_{x'},w)) \\
&\quad -\mathrm{P}_\phi(w|t_x)\sum_{w'}\mathrm{P}_\phi(w'|t_x)(\mathbf{f}_i(t_x,w') - \sum_{x'}\mathrm{P}_\phi(t_{x'}|t,w')\mathbf{f}_i(t_{x'},w'))
\end{aligned}$$

and finally the derivative of $l(\phi)$ (Equation 7.10) with respect to ϕ_i can be computed as:

$$\begin{aligned}
\frac{\partial}{\partial \phi_i}l(\phi) &= \sum_{w\rightarrow t_x\in\mathcal{R}_l} e_{t_x,w}\frac{\partial}{\partial \phi_i}\log \mathrm{P}_\phi(w|t_x) - 2\kappa\phi_i \\
&= \sum_{w\rightarrow t_x\in\mathcal{R}_l} e_{t_x,w}\frac{1}{\mathrm{P}_\phi(w|t_x)}\frac{\partial}{\partial \phi_i}\mathrm{P}_\phi(w|t_x) - 2\kappa\phi_i \\
&= \sum_{w\rightarrow t_x\in\mathcal{R}_l} e_{t_x,w}(\mathbf{f}_i(t_x,w) - \sum_{x'}\mathrm{P}_\phi(t_{x'}|t,w)\mathbf{f}_i(t_{x'},w)) \\
&\quad \underline{- \sum_{w\rightarrow t_x\in\mathcal{R}_l} e_{t_x,w}\sum_{w'}\mathrm{P}_\phi(w'|t_x)(\mathbf{f}_i(t_x,w') - \sum_{x'}\mathrm{P}_\phi(t_{x'}|t,w')\mathbf{f}_i(t_{x'},w'))} - 2\kappa\phi_i
\end{aligned}$$

Now take a look at the underlined term on the right hand side of the above equation. First, it can be written as follows because the inner summation is independent of w,

$$\sum_{t_x} e_{t_x,\cdot}\sum_{w'}\mathrm{P}_\phi(w'|t_x)(\mathbf{f}_i(t_x,w') - \sum_{x'}\mathrm{P}_\phi(t_{x'}|t,w')\mathbf{f}_i(t_{x'},w'))$$

Second, since $\mathrm{P}_\phi(w'|t_x)$ forms a proper conditional distribution of words in the training data that co-occur with POS tag t, $\mathrm{P}_\phi(w'|t_x) = 0$ if $w' \rightarrow t_x \notin \mathcal{R}_l$ and thus the two summations $\sum_{t_x}$ and $\sum_{w'}$ can be merged into $\sum_{w' \rightarrow t_x \in \mathcal{R}_l}$ and this term can be further simplified to:

$$\sum_{w' \rightarrow t_x \in \mathcal{R}_l} e_{t_x,\cdot} \mathrm{P}_\phi(w'|t_x)(\mathbf{f}_i(t_x, w') - \sum_{x'} \mathrm{P}_\phi(t_{x'}|t, w')\mathbf{f}_i(t_{x'}, w'))$$

Hence, $\frac{\partial}{\partial \phi_i} l(\phi)$ can be simplified to:

$$\begin{aligned} \frac{\partial}{\partial \phi_i} l(\phi) &= \sum_{w \rightarrow t_x \in \mathcal{R}_l} (e_{t_x,w} - e_{t_x,\cdot} \mathrm{P}_\phi(w|t_x))(\mathbf{f}_i(t_x, w) - \sum_{x'} \mathrm{P}_\phi(t_{x'}|t, w)\mathbf{f}_i(t_{x'}, w)) - 2\kappa\phi_i \\ &= \sum_{w \rightarrow t_x \in \mathcal{R}_l} e^*_{t_x,w}(\mathbf{f}_i(t_x, w) - \sum_{x'} \mathrm{P}_\phi(t_{x'}|t, w)\mathbf{f}_i(t_{x'}, w)) - 2\kappa\phi_i \end{aligned} \quad (7.12)$$

where $e^*_{t_x,w} = e_{t_x,w} - e_{t_x,\cdot} \mathrm{P}_\phi(w|t_x)$.

In summary, the gradient of $l(\phi)$ with respect to ϕ has the following form:

$$\begin{aligned} \nabla l(\phi) &= \sum_{w \rightarrow t_x \in \mathcal{R}_l} e^*_{t_x,w} \Delta_{t_x,w}(\phi) - 2\kappa \cdot \phi \\ \Delta_{t_x,w}(\phi) &= \mathbf{f}(t_x, w) - \sum_{x'} \mathrm{P}_\phi(t_{x'}|t, w)\mathbf{f}(t_{x'}, w) \end{aligned}$$

and $l(\phi)$ can be optimized by a gradient descent optimization algorithm, such as LBFGS [97] that we use in our experiments.

It can be shown that $l(\phi)$ is not a concave function with respect to ϕ, but this created no problems in our experiments. It should be noted that if we set the regularization weight κ to 0, the maximum of $l(\phi)$ is achieved when $\mathrm{P}_\phi(w|t_x)$ is set

to $e_{t_x,w}/e_{t_x,\cdot}$, which is identical to the update formula in Equation 5.6, and would thus be unable to use rich features. This is less of an issue when regularization takes effect, as it favors common discriminative features to reduce the penalty term.

The second approach, which was found to outperform the EM-based approach in (Berg-Kirkpatrick et al. [8]), optimizes on the regularized log-likelihood (Equation 7.9) directly by updating both ψ and ϕ using a gradient descent approach. This is a constrained optimization problem because the elements of ψ are constrained to form proper probability distributions. In order to convert this to an unconstrained optimization problem, we set each phrasal rule expansion probability ψ_i as the output of a log-linear model, i.e., $\psi_i = \exp(\psi'_i)/Z$ with Z being the normalization factor, and treat ψ' as the parameter for the phrasal rules to be optimized. The gradient of $L(\theta)$ with respect to ϕ turns out to be the same as in the first approach [137]. The gradient of $L(\theta)$ with respect to ψ' can be derived similarly.

In the original EM-based training approach [126], many of the rule expansion probabilities become very small and are pruned to dramatically reduce the grammar size. The phrasal rule probabilities computed from the log-linear model with parameter ψ' are not usually low enough to be pruned, due to the fact that a large decrease in ψ'_i results in a much smaller change in ψ_i when ψ_i is already relatively small. In order to address this problem, we combine the two optimization approaches together: we first run rounds of EM-based optimization to initialize the grammar parameters and prune many of the useless phrasal rules, and then switch to the direct gradient descent optimization approach. This combined approach outperforms the standalone EM-based approach in our study and is used in the experiments

reported in this chapter.

7.5 Experiments

7.5.1 Setup

We experiment with three different languages: English, Chinese, and Arabic. For English, we use the WSJ Penn treebank [104] and the commonly used data splits [31], i.e., sections 02-21 for training, 22 for development, and 23 for final test. For Chinese, we use the Penn Chinese treebank 6.0 (CTB6) [162] and the preparation steps and data splits in Section 5.5. For Arabic, we use the Penn Arabic treebank (ATB) [100] and the preparation steps and data splits[41] in (Chiang et al. [42], Green and Manning [61]).

Table 7.1 provides gross statistics for each treebank. As can be seen, CTB6 and ATB both have a higher OOV rate than WSJ, and hence have greater need for effective OOV handling.

Due to the variability (caused by random initialization) among the grammars [122], we train 10 grammars with different seeds in each experiment and report their average F score on the development set. The best grammar selected using the development set is used for evaluation on the test set.

[41]The ATB corpus is split into 80% for training, 10% for development, and the remaining 10% for evaluation. The preprocessing steps include: removing function tags, traces, and trees dominated by X, collapsing unary chains that expand non-terminals to themselves, mapping preterminal morphological analyses to "Bies" tags, adding "DT" to the tags for definite nouns and adjectives, and performing orthographic normalization. We did not remove clitic marks, which results in about 0.3 degradation in F score.

	Statistics	Train	Dev	Test
English (WSJ)	#sents	39832	1700	2416
	#tokens	950.0k	40.1k	56.7k
	%oov_types	-	12.8%	13.2%
	%oov_tokens	-	2.8%	2.5%
Chinese (CTB6)	#sents	24416	1904	1975
	#tokens	678.8k	51.2k	52.9k
	%oov_types	-	20.6%	20.9%
	%oov_tokens	-	5.0%	5.3%
Arabic (ATB)	#sents	18818	2318	2313
	#tokens	597.9k	70.7k	70.1k
	%oov_types	-	15.6%	16.7%
	%oov_tokens	-	3.2%	3.4%

Table 7.1: Gross statistics of the treebanks

7.5.2 Standard PCFG-LA Grammars

In Section 5.5.2, we investigated the effect of heuristic rare word smoothing (see Section 5.4.1) and OOV handling (see Section 5.4.2) for the standard PCFG-LA grammars on English and Chinese. In order to establish baselines for the feature rich log-linear model, we re-evaluate these methods in a more detailed manner for all the three languages studied in this chapter.

The results are presented in Table 7.2. The *no+simple* row represents the baseline, for which the grammars are trained without rare word smoothing and OOV words are handled by the simple method. Each language-dependent heuristic-based OOV word handling method improves parsing accuracies, and the rare word smoothing method provides even greater improvement across the languages. Their combination results in further improvement. This reconfirms that both over-fitting and OOV words are issues to consider for training accurate PCFG-LA grammars.

Rare Word Smoothing	OOV	WSJ	CTB6	ATB
no	simple	89.9	82.5	79.1
no	heuristic	90.1	83.0	79.4
yes	simple	90.5	83.3	80.3
yes	heuristic	90.7	83.7	80.6

Table 7.2: The effect of rare word smoothing and OOV handling on parsing F scores evaluated on the respective development set

Predicate	Explanation
$\delta(w = \cdot)$	word identity (wid)
$\delta(\text{hasDigit}(w) = \cdot)$	contains a digit?
$\delta(\text{hasHyphen}(w) = \cdot)$	contains a hyphen?
$\delta(\text{initCap}(w) = \cdot)$	first letter capitalized?
$\delta(\text{prefix}_k(w) = \cdot)$	prefix of length $k \leq 3$
$\delta(\text{suffix}_k(w) = \cdot)$	suffix of length $k \leq 3$

Table 7.3: Predicate templates on word w

7.5.3 Log-Linear Lexical Models

A core set of features that have proven effective for POS tagging will be applied to demonstrate the effectiveness of our model and its robustness across languages. Table 7.3 lists the templates we use to extract predicates on words. For the log-linear OOV model, we use the *full* feature set, i.e., (t, pred) pairs extracted using all of the predicates. For the log-linear latent lexical model, we experiment with two feature sets: 1) the word identity (*wid*) feature set containing only (t_x, wid) pairs, which are the same as those used in the standard PCFG-LA grammars, 2) the *full* feature set using all of the predicates.

We first evaluate the effectiveness of regularization and the log-linear OOV model by training the latent lexical model using the *wid* feature set with regulariza-

Latent Lexical	OOV	WSJ	CTB6	ATB
wid	simple	90.5	83.2	80.3
wid	heuristic	90.7	83.6	80.7
wid	full	90.8	83.9	81.4
full	full	90.9	84.2	81.8

Table 7.4: The effect of features (wid vs. full) for training the latent lexical model and the OOV handling methods (simple, heuristic, or the log-linear model using the full feature set) on parsing performance on the development set

tion and examining different OOV handling methods. As shown in Table 7.4, the *wid+simple* and *wid+heuristic* approaches[42] produce results comparable to the corresponding PCFG-LA grammars trained with rare word smoothing and respective OOV handling. This suggests that regularizing the latent lexical model alleviates data sparsity, however, we will illustrate in Subsection 7.5.4 that this is achieved in a different way than rare word smoothing.

The log-linear OOV model using the *full* feature set results in improved parsing performance over all languages, with the greatest improvement seen on Arabic (0.7 F), followed by Chinese (0.3 F), confirming that the log-linear OOV model is more accurate than the heuristic approach, and can be flexibly used for different languages. The improvement on English is marginal possibly because our simple features are not significantly better than the signature-based OOV features that are quite effective at handling English unknown words after years of expert crafting.

We next investigate the effect of training the latent lexical model using the *full* feature set. Compared with the *wid+full* model, the *full+full* model improves 0.4

[42]Training the latent lexical model using the *wid* feature set and handling OOV words using the *simple* or *heuristic* approach.

F on Arabic and 0.3 F on Chinese, despite the fact that the additional features are very simple, mostly prefixes and suffixes of words. The improvement on English is again marginal possibly because the features do not provide insight on fine-grained syntactic subcategories (e.g., suffix *-ed* is indicative of past tense verbs, but not their sub-categories). Admittedly, many of the features are noisy, but as we will show in Subsection 7.5.4, some of the features are able to guide the learning of the latent categories to reflect the similarity among syntactically similar words of the same POS type.

Compared with the baseline (*no+simple* in Table 7.2), the feature-rich *full+full* model significantly improves parsing F scores by 1, 1.7, and 2.7 absolute on English, Chinese, and Arabic, respectively.

Table 7.5 compares the final test results of our best grammars (the *full+full* approach) with the literature[43]. Our PCFG-LA grammars with a feature-rich lexical model significantly outperform the standard PCFG-LA grammars of Petrov and Klein [123] for all of the three languages, especially on Chinese (+1.6 F) and Arabic (+2.2 F). Compared to the discriminatively trained PCFG-LA grammar of Petrov and Klein [124], which is 0.7 F worse than its generative counterpart on English (89.4 vs. 90.1 F), our grammar achieves a significantly higher accuracy of 90.5 and improves upon the standard PCFG-LA grammar of Petrov and Klein [123] by 0.4 F.

[43]All of the parsers from the referenced papers are trained and evaluated using the data splits in our experiments.

TB	Parser	LP	LR	F
WSJ	Charniak [31]	89.9	89.5	89.7
	Petrov and Klein [123]	90.2	90.1	90.1
	Petrov and Klein [124]	-	-	89.4
	Huang and Harper [71]	90.4	89.9	90.1
	PCFG-LA with Feature Rich Lexical Model	**90.8**	**90.3**	**90.5**
CTB6	Charniak [31]	80.5	79.5	80.0
	Petrov and Klein [123]	84.0	82.9	83.4
	Huang and Harper [71]	85.1	83.2	84.1
	PCFG-LA with Feature Rich Lexical Model	**85.9**	**84.2**	**85.0**
ATB	Petrov and Klein [123]	80.5	78.9	79.7
	PCFG-LA with Feature Rich Lexical Model	**82.7**	**81.2**	**81.9**

Table 7.5: Final test set accuracies

7.5.4 Analysis

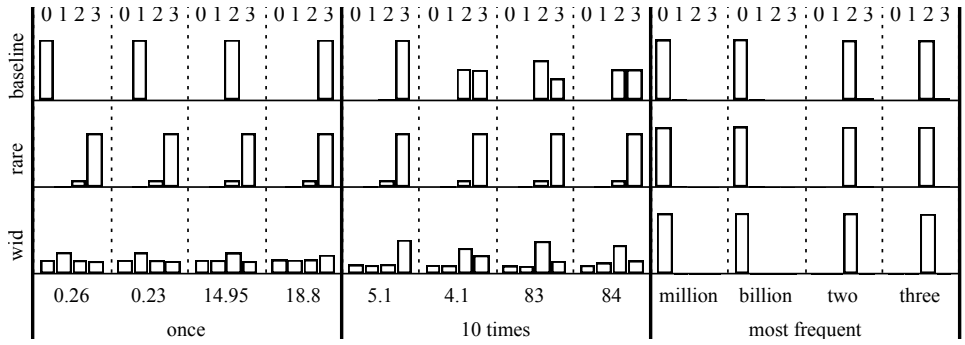


Figure 7.1: The conditional distribution $\mathrm{P}(t_x|t,w)$ of latent tags for selected cardinal numbers (e.g., 0.26, million) that appear only once, 10 times, or more frequently for standard PCFG-LA grammars trained with (labeled rare) or without (labeled baseline) rare word smoothing, as well as for PCFG-LA grammars with regularized feature-rich lexical model using the *wid* feature set (labeled wid). The distribution is represented by the four bars separated by dotted vertical lines, and each bar represents the conditional probability of a latent tag.

Figure 7.1 shows the effect of regularization and rare word smoothing on the learned rules by depicting the distribution $\mathrm{P}(t_x|t,w)$ for PCFG-LA grammars trained in three different ways[45]. For standard PCFG-LA grammars trained without rare word smoothing (labeled baseline), rare words have sparse distributions of latent

[45]For standard PCFG-LA grammars, $\mathrm{P}(t_x|t,w)$ is simply computed by $e_{t_x,w}/e_{t,w}$; whereas, for the feature-rich lexical model, $\mathrm{P}(t_x|t,w)$ is computed from the latent lexical model.

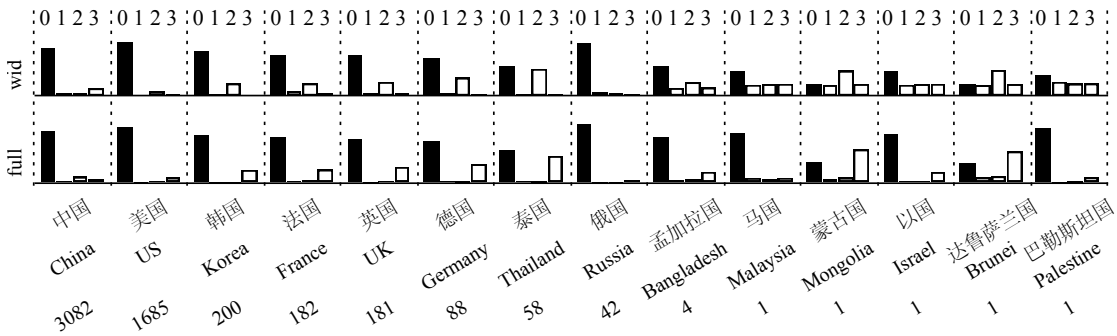


Figure 7.2: The conditional distribution $\mathrm{P}(t_x|t,w)$ of latent tags for selected country names (proper nouns) that are listed in order of decreasing frequency from the Chinese treebank (The English translation and word frequency are provided under each Chinese name), based on training using the *wid* or the *full* feature set. The distribution is represented by the four bars separated by dotted vertical lines, and each bar represents the conditional probability of a latent tag. The preferred latent tag for country names is highlighted in black.

tags, which are determined by the EM algorithm solely based on limited contexts and are thus unreliable. The rare word smoothing approach (labeled rare) collapses all rare words into a single token so that $\mathrm{P}(t_x|t,w) = \mathrm{P}(t_x|t,\mathit{rare})$ is identical for any rare word w. This constraint greatly reduces data sparsity; however, treating all rare words as one token could eliminate too much lexical information (e.g., the distribution of latent tags is the same for all rare cardinal numbers no matter whether they appear only once or 10 times). Regularization of the log-linear latent lexical model (labeled wid) favors a uniform distribution (zero penalty when all feature weights are zero). There is not much evidence to skew the distribution from uniform for rare words. However, when more evidence is available, the distribution becomes smoothly skewed to reflect the different syntactic preferences of the individual words, and it can eventually become as spiky as in the other approaches given sufficient evidence.

In order to provide some insight into why parsing accuracies are improved for

Chinese by using the *full* feature set when training the latent lexical model, we look at the country names that end with the character 国 (country) in the Chinese treebank. These names appear in similar contexts in the treebank and would be expected to favor a certain latent tag or tags (which are highlighted in black in Figure 7.2 for illustration purposes). When training using the *wid* feature set, however, this is only true for the frequent names as shown in Figure 7.2. For the rare names, since each word is modeled independently from others, they are unable to share any statistics with each other and thus there is not much evidence to divert the distribution away from uniform. When training with the *full* feature set, the suffix1=国 predicate is active for all country names and has a large feature weight associated with the preferred latent tag. As a result, the distribution of latent tags for the rare names is skewed more toward the preferred latent tag due to strong evidence from that suffix feature.

7.6 Conclusions

In this chapter, we have presented a feature-rich lexical model for PCFG-LA grammars to: 1) alleviate over-fitting via regularization, 2) handle OOV words using rich features, and 3) exploit lexical features for grammar induction. Experiments show that this approach allows us to train more effective PCFG-LA grammars for more accurate and robust parsing of three different languages.

Chapter 8

Machine Translation with Latent Variables

8.1 Overview

Syntax-based translation models have recently shown promising progress at improving translation quality, thanks to the incorporation of phrasal translation adopted from the widely used phrase-based models to handle local fluency and the engagement of synchronous context-free grammars (SCFG) to handle non-local phrase reordering. Approaches to syntax-based translation models can largely be categorized into linguistic syntax-based models that utilize structures defined based on linguistic theory and human annotation (e.g., Penn treebank) to guide the derivation of SCFG rules with explicit parsing on at least one side of the parallel corpus, and formal syntax-based models that extract synchronous grammars from parallel corpora based on the hierarchical structure of natural language pairs without any explicit linguistic knowledge or annotation. In this chapter, we aim to bring the advantages of both types of models together and focus on learning a set of linguistically-guided latent syntactic categories to enrich hierarchical phrase-based models, a type of formal syntax-based model.

We augment rules in hierarchical phrase-based translation models with novel syntactic features. Unlike previous studies that directly use explicit treebank categories such as NP, NP/PP (NP missing a PP on the right) to annotate phrase pairs,

we induce a set of latent categories to capture the syntactic dependencies inherent in the hierarchy of phrase pairs, and derive a real-valued feature vector for each X nonterminal of an SCFG rule based on the distribution of the latent categories. Moreover, we replace the equality test of two sequences of syntactic categories, which are either identical or different, with a similarity score between their corresponding feature vectors. In our model, two *symbolically different* sequences of syntactic categories could have a high similarity score in the feature vector representation if they are *syntactically similar*, and a low score otherwise. In decoding, these feature vectors are utilized to measure the similarity between the syntactic analysis of the source side and the syntax of the SCFG rules that are applied to derive translations. Our approach maintains the advantages of hierarchical phrase-based translation systems, while at the same time incorporating soft syntactic constraints. To the best of our knowledge, this is the first use of real-valued syntactic feature vectors for machine translation.

The rest of the chapter is organized as follows. Section 8.2 briefly reviews hierarchical phrase-based translation models and motivates our proposed approach, which is summarized in Section 8.3. Section 8.4 describes the hierarchy of aligned phrase pairs, and Section 8.5 describes how to induce latent syntactic categories. Experimental results are reported in Section 8.6. Section 8.7 concludes the chapter.

8.2 Introduction to Hierarchical Phrase-Based Translation

An SCFG is a synchronous rewriting system that generates source and target side string pairs simultaneously based on a context-free grammar. Each synchronous production (i.e., rule) rewrites a nonterminal into a pair of strings, γ and α, where γ (or α) contains terminal and nonterminal symbols from the source (or target) language, and there is a one-to-one correspondence between the nonterminal symbols on both sides. The hierarchical model of Chiang [40] explores hierarchical structures of natural language and utilizes only a single nonterminal symbol X in the grammar,

$$X \rightarrow \langle \gamma, \alpha, \sim \rangle$$

where $\sim$ is the one-to-one correspondence between X's in γ and α, which is indicated by subscript co-indexes. Two example English-to-Chinese translation rules are represented as follows:

$$X \rightarrow \langle \text{give the pen to me}, \text{钢笔 给 我} \rangle \tag{8.1}$$

$$X \rightarrow \langle \text{give } X_1 \text{ to me}, X_1 \text{ 给 我} \rangle \tag{8.2}$$

The SCFG rules of hierarchical phrase-based models are extracted automatically from corpora of word-aligned parallel sentence pairs [22, 118]. An aligned sentence pair is a tuple (E, F, A), where $E = e_1 \cdots e_n$ can be interpreted as an English sentence of length n, $F = f_1 \cdots f_m$ is its translation of length m in a foreign language, and A is a set of links between words of the two sentences. Figure 8.1

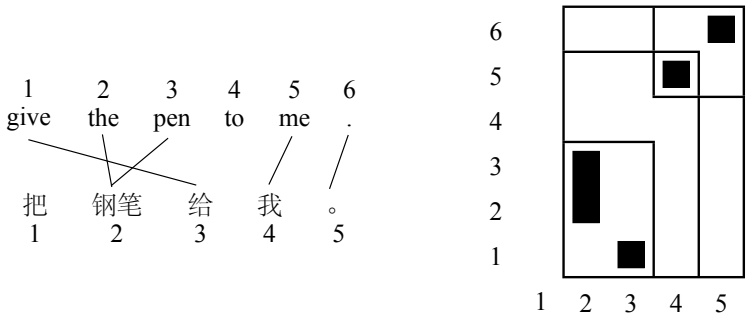


Figure 8.1: An example of a word-aligned sentence pair in (a) with tight phrase pairs marked (in black) in a matrix representation shown in (b)

(a) shows an example of an aligned English-to-Chinese sentence pair. As is widely adopted in phrase-based models [119], a pair of consecutive sequences of words from E and F is a *phrase pair* if all words are aligned only within the sequences and not to any word outside of them. We call a sequence of words a *phrase* if it corresponds to either side of a phrase pair, and a *non-phrase* otherwise. Note that the boundary words of a phrase pair may not be aligned to any other word. We call the phrase pairs with all boundary words aligned *tight* phrase pairs [166]. A tight phrase pair is the minimal phrase pair among all phrase pairs that share the same set of alignment links. Figure 8.1 (b) highlights the tight phrase pairs in the example sentence pair.

The extraction of SCFG rules proceeds as follows. In the first step, all phrase pairs below a maximum length are extracted as phrasal rules. In the second step[46], abstract rules are extracted from tight phrase pairs that contain other tight phrase pairs by replacing the sub-phrase pairs with co-indexed X-nonterminals. In our earlier example, rule (8.2) can be extracted from rule (8.1) with the following sub-

[46]Chiang [40] also introduced several requirements (e.g., there are at most two nonterminals on the right hand side of a rule) to safeguard the quality of the abstract rules, as well as keeping decoding efficient.

phrase pair:

$$X \rightarrow \langle \text{the pen}, \text{钢笔} \rangle$$

Each SCFG rule is associated with a set of features such as rule and lexicalized rule translation probabilities [40] that describe the quality of the rule. When translating an input foreign sentence F using an SCFG, the goal is to find the English sentence E with derivation D such that the following objective is maximized:

$$l(D) = \lambda_{\text{LM}} f_{\text{LM}}(E) + \sum_{X \rightarrow \langle \gamma, \alpha, \sim \rangle \in D} \sum_{i} \lambda_i \phi_i (X \rightarrow \langle \gamma, \alpha, \sim \rangle) \tag{8.3}$$

where $f_{\text{LM}}(E)$ is the language model score, each ϕ_i is a feature defined on an SCFG rule, and the λ's are the feature weights, which can be optimized on a development set using, for example, minimum-error-rate training [117].

Despite the complete lack of linguistic guidance, the performance of hierarchical phrase-based models is competitive with linguistic syntax-based models, which are unavoidably affected by errors in syntactic annotations. As shown in (Mi and Huang [110]), hierarchical phrase-based models significantly outperform tree-to-string models [70, 99], even when attempts are made to alleviate parsing errors using either forest-based decoding [111] or forest-based rule extraction [110]. However, when properly used, syntactic constraints can provide invaluable guidance to improve translation quality. The tree-to-string models of Mi and Huang [110] significantly outperforms hierarchical phrase-based models when using forest-based

rule extraction together with forest-based decoding. Chiang [41] also obtained a significant improvement over his hierarchical baseline by inducing fuzzy (inexact) tree-to-tree rules based on syntactic parse trees on both source and target sides and by allowing syntactically mismatched substitutions.

The use of a single X nonterminal makes hierarchical phrase-based models flexible at capturing non-local reordering of phrases. However, such flexibility also comes at the cost that it is unable to differentiate between different syntactic usages of phrases. Suppose rule $X \rightarrow \langle \text{give the pen to } X_1, \cdots \rangle$ is extracted from a phrase pair with "give the pen to me" on the source side where X_1 is abstracted from the pronoun phrase pair. If this rule is used to translate "give the pen to the boy over there", it would be better to apply it over the entire string or the sub-string "give the pen to the boy" than other sub-strings such as "give the pen to the boy over". This is because the nonterminal X_1 in the rule was abstracted from a pronoun on the source side of the training data and would thus be better (more informative) if it were applied to phrases of similar type, e.g., a noun phrase. However, hierarchical phrase-based models are unable to distinguish syntactic differences like this.

Zollmann and Venugopal [169] attempted to address this problem by annotating phrase pairs with treebank categories based on automatic parse trees. They introduced an extended set of categories (e.g., NP+V for "she went" and DT\NP for "great wall", a noun phrase with a missing determiner on the left) to annotate phrase pairs that do not align with syntactic constituents. Their hard syntactic constraint requires that the nonterminals should match exactly to rewrite a nonterminal. As a result, potentially correct derivations could be ruled out due to data

sparsity and errors in the parse trees. For example, NP cannot be instantiated with phrase pairs of type DT+NN, despite their syntactic similarity. Venugopal et al. [152] addressed this problem by directly introducing soft syntactic preferences into SCFG rules using a preference grammar, but this came with the computational challenge of processing large preference vectors. Chiang [41] also avoided hard constraints by taking a soft alternative that directly models the cost of mismatched rule substitutions. This solution, however, requires a large number of parameters to be tuned on a generally small-size held-out set, and it could thus suffer from over-tuning.

8.3 Our Approach

We take a different approach to improve a hierarchical phrase-based translation system by introducing linguistic syntax to SCFG rules and using soft syntactic constraints between derivation rules and the parse tree of the source side at decoding time. In this section, we describe how to enrich each X nonterminal of an SCFG rule with a feature vector computed based on a given set of latent syntactic categories and how to impose soft syntactic constraints. The set of latent syntactic categories are automatically induced from a source-side parsed, word-aligned parallel corpus based on the syntax-enriched hierarchy among phrase pairs. We defer the details on how to obtain latent syntactic categories until Sections 8.4 and 8.5.

For each phrase pair extracted from a sentence pair of a source-side parsed parallel corpus, we represent its syntax using a *tag sequence*, which is the sequence

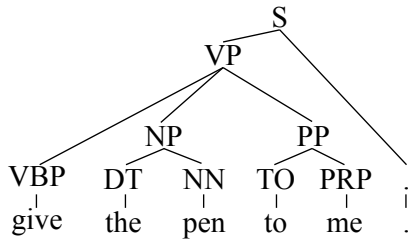


Figure 8.2: A source side parse tree

of highest syntactic categories of the source-side parse tree that exactly dominates the source-side phrase[47]. Figure 8.2 shows the source-side parse tree of a sentence pair. The tag sequences for "the pen" and "give the pen" are "NP" and "VBP NP", respectively, because the former is simply a noun phrase while the latter is dominated by a verb followed by a noun phrase. Let $TS = \{ts_1, \cdots, ts_m\}$ be the set of all tag sequences extracted from a parallel corpus. The syntax of each X nonterminal[48] in an SCFG rule can then be characterized by the distribution of tag sequences $\vec{\mathrm{P}}_X(TS) = (\mathrm{p}_X(ts_1), \cdots, \mathrm{p}_X(ts_m))$, based on the phrase pairs it is abstracted from. Table 8.1 shows an example distribution of tag sequences for X_1 in $X \rightarrow \langle \text{give the pen to } X_1, \cdots \rangle$.

Instead of directly using tag sequences, given the disadvantages discussed in Section 8.2, we represent each of them by a real-valued feature vector. Suppose[49] we have a collection of n latent syntactic categories $\mathcal{C} = \{c_1, \cdots, c_n\}$, and we know how they are distributed for each tag sequence ts, i.e., $\vec{\mathrm{P}}_{ts}(\mathcal{C}) = (\mathrm{p}_{ts}(c_1), \cdots, \mathrm{p}_{ts}(c_n))$. For example, $\vec{\mathrm{P}}_{\text{"NP VP"}}(\mathcal{C}) = \{0.5, 0.2, 0.3\}$ means that the latent syntactic categories

[47]In case of a non-tight phrase pair, we only consider its largest tight part.

[48]There are three X nonterminals (one on the left and two on the right) for binary abstract rules, two for unary abstract rules, and one for phrasal rules.

[49]Details about how the latent syntactic categories are induced and how they are distributed for each tag sequence ts will be presented in Sections 8.4 and 8.5.

Tag Sequence	Probability
PRP	0.40
NP	0.35
NP IN	0.25

Table 8.1: The distribution of tag sequences for X_1 in $X \rightarrow \langle \text{give the pen to } X_1, \cdots \rangle$

c_1, c_2, and c_3 are distributed as $\mathrm{p}(c_1) = 0.5$, $\mathrm{p}(c_2) = 0.2$, and $\mathrm{p}(c_3) = 0.3$ for tag sequence "NP VP". We convert each distribution $\vec{\mathrm{P}}_{ts}(\mathcal{C})$ to a normalized feature vector $\vec{F}(ts)$:

$$\begin{aligned} \vec{F}(ts) &= (f_1(ts), \cdots, f_n(ts)) \\ &= \frac{(\mathrm{p}_{ts}(c_1), \cdots, \mathrm{p}_{ts}(c_n))}{\|(\mathrm{p}_{ts}(c_1), \cdots, \mathrm{p}_{ts}(c_n))\|} \end{aligned}$$

The degree of similarity between any pair of tag sequences ts and ts' in the space of the latent syntactic categories $\mathcal{C}$ can then be computed as a dot-product[50] of their feature vectors:

$$\vec{F}(ts) \cdot \vec{F}(ts') = \sum_{1 \leq i \leq n} f_i(ts) f_i(ts')$$

which ranges between 0 (totally syntactically different) and 1 (completely syntactically identical).

Similarly, we can represent the syntax of each X nonterminal of a rule by a

[50] Other measures such as KL-divergence in the probability space would also be feasible.

feature vector $\vec{F}(X)$ based on its distribution of tag sequences:

$$\vec{F}(X) = \sum_{ts \in TS} \mathrm{p}_X(ts)\vec{F}(ts)$$

Now we can impose soft syntactic constraints using these feature vectors when an SCFG rule is used to translate a parsed source sentence. We compute the following syntactic similarity score on the fly when an X nonterminal of a rule is applied to a span whose tag sequence is ts as determined by parse tree of the source sentence[51]:

$$\mathrm{SynSim}(X, ts) = -\log(\vec{F}(X) \cdot \vec{F}(ts)) \tag{8.4}$$

We denote $\phi_{\mathrm{SYN}}(X \rightarrow \langle \gamma, \alpha, \sim \rangle)$ as the sum of the above syntactic similarity score (Equation 8.4) for all of the nonterminals in rule $X \rightarrow \langle \gamma, \alpha, \sim \rangle$ and add it to the decoding objective $l(D)$ of Equation 8.3, which now becomes:

$$\begin{aligned} l(D) &= \lambda_{\mathrm{LM}} f_{\mathrm{LM}}(E) + \sum_{X \rightarrow \langle \gamma, \alpha, \sim \rangle \in D} \sum_i \lambda_i \phi_i(X \rightarrow \langle \gamma, \alpha, \sim \rangle) \\ &\quad + \sum_{X \rightarrow \langle \gamma, \alpha, \sim \rangle \in D} \lambda_{\mathrm{SYN}} \phi_{\mathrm{SYN}}(X \rightarrow \langle \gamma, \alpha, \sim \rangle) \end{aligned}$$

Even though the feature ϕ_{SYN} is computed on the fly, it can be used in the same way as the standard features in a hierarchical translation system to determine the best translation, and its feature weight λ_{SYN} can be tuned together with the other feature weights on a held-out data set.

[51] A normalized uniform feature vector is used for tag sequences (of parsed test sentences) that are not seen in the training corpus.

In what follows, we will describe the two central aspects of our approach to obtain the latent syntactic categories, i.e., 1) how to construct the syntax-enriched hierarchy among all phrase pairs in a source-side parsed sentence pair, and 2) how to induce the latent syntactic categories from the hierarchy to capture the syntactic dependencies among the phrase pairs.

8.4 Alignment-based Hierarchy

Given two non-disjoint tight phrase pairs that share at least one common alignment link, there are only two relationships: either one completely includes the other or they do not include one another but have a non-empty overlap, which we call a non-trivial overlap. In the second case, the intersection, differences, and union of the two phrase pairs are also tight phrase pairs (see Figure 8.1 (b) for an example), and the two phrase pairs, as well as their intersection and differences, are all sub-phrase pairs of their union. This property implies that there is a hierarchy of phrase pairs. We next describe how to identify the hierarchy of phrase pairs and how to annotate the hierarchy with linguistic information that is necessary for inducing latent syntactic categories, which we will describe in Section 8.5.

Our hierarchy construction algorithm follows Heber and Stoye [64]. Let $\mathcal{P}$ be the set of tight phrase pairs extracted from a sentence pair. We call a sequentially-ordered list[52] $L = (p_1, \cdots, p_k)$ of unique phrase pairs $p_i \in \mathcal{P}$ a *chain* if every two successive phrase pairs in L have a non-trivial overlap. A chain is *maximal* if it

[52]The phrase pairs are sequentially ordered first by the boundary positions of the source-side phrase and then by the boundary positions of the target-side phrase.

cannot be extended to its left or right with other phrase pairs. Note that any subsequence of phrase pairs in a chain generates a tight phrase pair. In particular, chain L generates a tight phrase pair that corresponds exactly to the union of the alignment links in $p \in L$. We call the phrase pairs generated by maximal chains *maximal* phrase pairs and call the other phrase pairs *non-maximal*. Non-maximal phrase pairs have non-trivial overlaps with some other phrase pairs while maximal phrase pairs do not, and it can be shown that any non-maximal phrase pair can be generated by a sequence of maximal phrase pairs. Note that the largest tight phrase pair that includes all alignment links in A is also a maximal phrase pair. Lemma 8.4.1 follows directly from the definition of maximal phrase pairs.

Lemma 8.4.1 *Given two different maximal phrase pairs p_1 and p_2, exactly one of the following alternatives is true: p_1 and p_2 are disjoint, p_1 is a sub-phrase pair of p_2, or p_2 is a sub phrase pair of p_1.*

According to Lemma 8.4.1, there is a unique decomposition tree $T = (N, E)$ that covers all of the tight phrase pairs of a sentence pair, where N is the set of maximal phrase pairs and E is the set of edges that connect pairs of maximal phrase pairs if one is a sub-phrase pair of another. All of the tight phrase pairs of a sentence pair can be either extracted directly from the nodes of the decomposition tree (these phrase pairs are maximal) or generated by a sequence of consecutive sibling nodes[53] (these phrase pairs are non-maximal). Figure 8.3 shows the decomposition tree as well as all of the tight phrase pairs that can be extracted from the example sentence

[53]Unaligned words may be added.

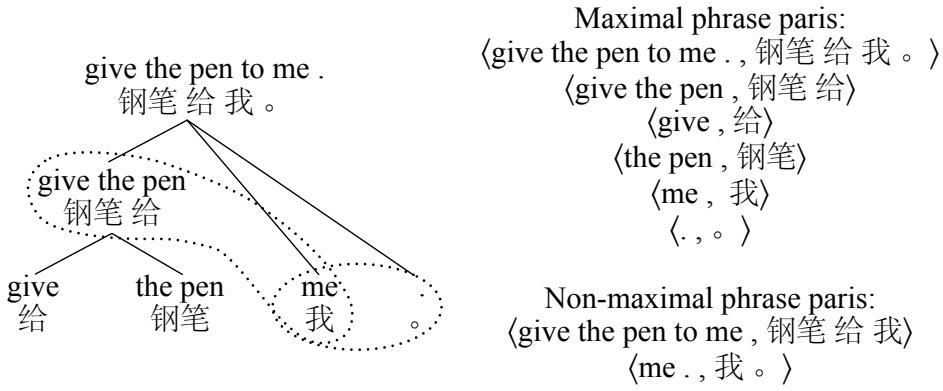


Figure 8.3: A decomposition tree of tight phrase pairs with all tight phrase pairs listed on the right. As highlighted by the dotted curves, the two non-maximal phrase pairs are generated by consecutive sibling nodes.

pair in Figure 8.1.

In our current approach, we only consider the syntactic constraints on the source side and thus focus on the source side of the decomposition tree[54]. In order to include all of the source words into the decomposition tree, we attach each non-phrase single word as a child of the lowest node of the decomposition tree that contains the word. Now there are two types of nodes in the parse tree. We abstract them with two symbols, X for phrases and B for non-phrases, and call the result the *decomposition tree* of the source side phrases. Figure 8.4 (a) depicts such a tree for the English side of our example sentence pair in Figure 8.1. We further recursively binarize the decomposition tree into a binarized decomposition forest such that each phrase is directly represented as a node in the forest[55]. Figure 8.4 (c) shows two of the many binarized decomposition trees in the forest. The binarized decomposition forest compactly encodes the hierarchy among phrases and non-phrases. The coarse

[54]We leave it to future work to consider syntactic constraints on both sides.

[55]The intermediate binarization nodes are also labeled as either X or B based on whether they exactly cover a phrase or not.

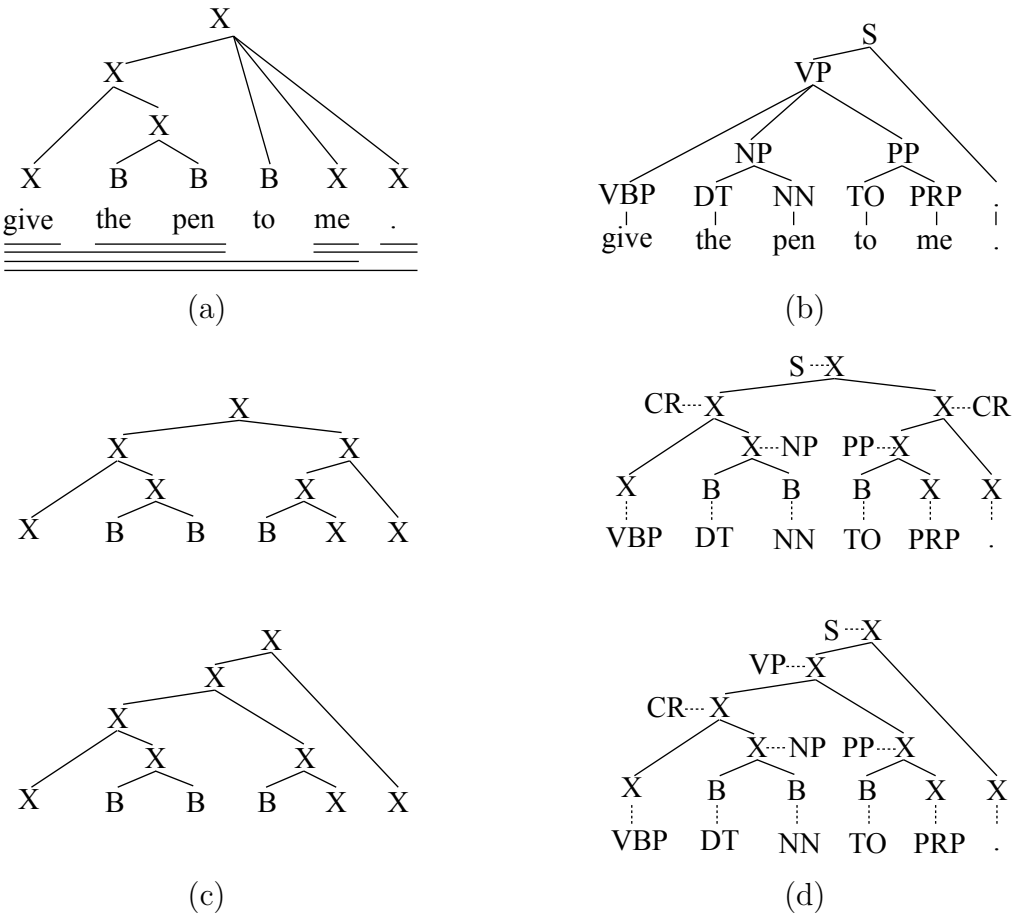


Figure 8.4: (a) decomposition tree for the English side of the example sentence pair with all phrases underlined, (b) automatic parse tree of the English side, (c) two example binarized decomposition trees with syntactic emissions depicted in (d).

abstraction of phrases with X and non-phrases with B provides little information on the constraints of the hierarchy. In order to bring in syntactic constraints, we annotate each node in the decomposition forest with a syntactic observation based on the automatic parse tree of the source side. If a node aligns with a constituent in the parse tree, we add the syntactic category (e.g., NP) of the constituent as an emitted observation of the node; otherwise, it crosses constituent boundaries, and we add a designated crossing category CR as its observation. We call the resulting forest a *syntactic decomposition forest*. Figure 8.4 (d) shows two syntactic decomposition trees of the forest based on the parse tree in Figure 8.4 (b).

In summary, we have constructed a syntax-enriched hierarchy that describes how longer phrases are composed of shorter phrases and non-phrases using the following steps:

1. Construct the decomposition tree of phrase pairs based on word alignment (see Figure 8.3).

2. Take the source side of the decomposition tree, attach the non-phrase single words, abstract the tree nodes with X for phrases and B for non-phrases (see Figure 8.4 (a)), and binarize the decomposition tree to obtain the binarized decomposition forest (see Figure 8.4 (c)).

3. Annotate each node of the binarized decomposition forest with a syntactic observation based on the source-side parse tree (see Figure 8.4 (d)).

We will next describe how to induce latent syntactic categories to capture the hierarchical syntactic constraints among phrases.

8.5 Inducing Latent Syntactic Categories

If we designate a unique symbol S as the new root of the syntactic decomposition forests introduced in the previous section, it can be shown that these forests can be generated by a probabilistic context-free grammar $G = (V, \Sigma, S, \mathcal{R}, \mathrm{P}_\phi)$, where

- $V = \{S, X, B\}$ is the set of nonterminals,
- Σ is the set of terminals comprising treebank categories plus the CR tag (the crossing category),
- $S \in V$ is the unique start symbol,
- $\mathcal{R}$ is the union of the set of product rules each rewriting a nonterminal to a sequence of nonterminals and the set of emission rules each generating a terminal from a nonterminal,
- P_ϕ assigns a probability score to each rule $r \in R$.

In contrast to the grammars for syntactic parsing described in Chapter 5, where terminals are only generated by preterminals, each internal nonterminal of our grammar also emits a terminal. Such a grammar can be derived from the set of syntactic decomposition forests extracted from a source-side parsed parallel corpus, with rule probabilities estimated as the relative frequencies of the product and emission rules.

The X and B nonterminals in the grammar are coarse representations of phrases and non-phrases and do not carry any syntactic information at all. In an attempt to introduce syntax to these nonterminals, we incrementally split each

of them into a set of latent categories ($\{X_1, \cdots, X_n\}$ for X and $\{B_1, \cdots, B_n\}$ for B) and then learn a set of rule probabilities[56] P_ϕ on the latent categories so that the likelihood of the training forests is maximized. As we will describe next, the latent categories are induced in a similar way to the fine-grained syntactic categories for the latent bigram tagger in Chapter 3 and for the PCFG-LA grammar in Chapter 5. The motivation is to allow the latent categories to learn different preferences of (emitted) syntactic categories as well as structural dependencies along the hierarchy so that they can carry syntactic information. We call these *latent syntactic categories*. The induced latent categories for the X nonterminal, i.e., the X_i's, represent syntax-enriched fine-grained categories of phrases and correspond to the set of latent syntactic categories $\mathcal{C}$ in Section 8.3. In this study, we induce 16 latent categories for both X and B nonterminals[57].

We use a variant of Expectation-Maximization (EM) algorithm to optimize the likelihood of the training forests. Our EM algorithm is similar to the inside-outside algorithm [126] for training PCFG-LA grammars in Chapter 5 (and also the forward-backward algorithm for training latent bigram taggers in Chapter 3), except that we work with forests instead of trees and each internal nonterminal also emits a terminal in addition to expanding to a sequence of nonterminals.

Recall that our decomposition forests are fully binarized (except for the root). In the hypergraph representation [69], the hyperedges of our forests all have the

[56] Each binary production rule is now associated with a 3-dimensional matrix of probabilities, and each emission rule is associated with a 1-dimensional array of probabilities.

[57] We incrementally split each nonterminal to 2, 4, 8, and finally 16 categories, with each splitting followed by several EM iterations to tune model parameters. Based on our preliminary experiments, we choose to use 16 latent categories, not too few so that they can differentiate among different syntactic uses and not so many that computational and storage costs become exorbitant.

same format[58] $\langle (V, W), U \rangle$, meaning that node U expands to nodes V and W with production rule $U \rightarrow VW$. Given a forest F with root node R, we denote $e(U)$ as the emitted syntactic category at node U and $\text{LR}(U)$ (or $\text{PL}(W)$, or $\text{PR}(V)$)[59] as the set of node pairs (V, W) (or (U, V), or (U, W)) such that $\langle (V, W), U \rangle$ is a hyperedge of the forest. Now consider node U, which is either S, X, or B in the forest. Let U_x be the latent syntactic category[60] of node U. We define $\text{I}(U_x)$ as the part of the forest (includes $e(U)$ but not U_x) inside U, and $\text{O}(U_x)$ as the other part of the forest (includes U_x but not $e(U)$) outside U, as illustrated in Figure 8.5. Given this, the inside-outside probabilities are defined as:

$$\begin{aligned} \text{P}_{\text{IN}}(U_x) &= \text{P}(\text{I}(U_x)|U_x) \\ \text{P}_{\text{OUT}}(U_x) &= \text{P}(\text{O}(U_x)|S) \end{aligned}$$

They can be computed recursively as:

$$\begin{aligned} \text{P}_{\text{IN}}(U_x) &= \sum_{(V,W)\in\text{LR}(U)} \sum_{y,z} \text{P}_\phi(U_x \rightarrow e(U))\text{P}_\phi(U_x \rightarrow V_y W_z)\text{P}_{\text{IN}}(V_y)\text{P}_{\text{IN}}(W_z) \\ \text{P}_{\text{OUT}}(U_x) &= \sum_{(V,W)\in\text{PL(U)} } \sum_{y,z} \text{P}_\phi(V_y \rightarrow e(V))\text{P}_\phi(V_y \rightarrow W_z U_x)\text{P}_{\text{OUT}}(V_y)\text{P}_{\text{IN}}(W_z) \\ &+ \sum_{(V,W)\in\text{PR(U)} } \sum_{y,z} \text{P}_\phi(V_y \rightarrow e(V))\text{P}_\phi(V_y \rightarrow U_x W_z)\text{P}_{\text{OUT}}(V_y)\text{P}_{\text{IN}}(W_z) \end{aligned}$$

[58] The hyperedge corresponding to the root node has a different format because it is unary, but it can be handled similarly. When clear from context, we use the same variable to present both a node and its label.

[59] LR stands for the left and right children, PL for the parent and left children, and PR for the parent and right children.

[60] We never split the start symbol S, and denote $S_0 = S$.

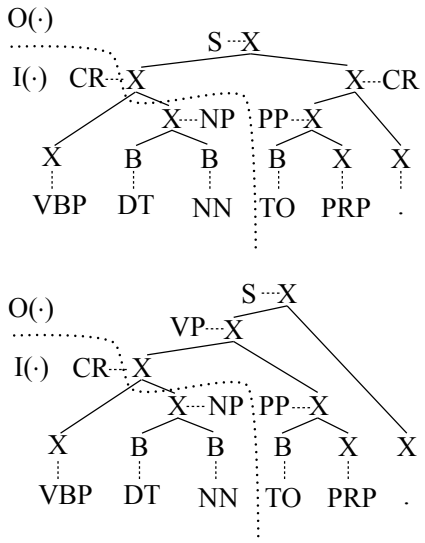


Figure 8.5: An example of I(·) and O(·) that separate the forest into two parts

In the E-step, the posterior probability of the occurrence of production rule[61] $U_x \to V_y W_z$ is computed as:

$$\mathrm{P}(U_x \to V_y W_z | F) = \frac{\mathrm{P}_\phi(U_x \to e(U))\mathrm{P}_\phi(U_x \to V_y W_z)\mathrm{P}_{\mathrm{OUT}}(U_x)\mathrm{P}_{\mathrm{IN}}(V_y)\mathrm{P}_{\mathrm{IN}}(W_w)}{\mathrm{P}_{\mathrm{IN}}(R)}$$

and it is accumulated into the expected count $c(U_x \to V_y W_z)$.

In the M-step, the probability estimation of $\mathrm{P}_\phi(U_x \to V_y W_z)$ is updated by normalizing the expected count:

$$\mathrm{P}_\phi(U_x \to V_y W_z) = \frac{c(U_x \to V_y W_z)}{\sum\limits_{(V',W')} \sum\limits_{y,z} c(U_x \to V_y W_z)}$$

Recall that each node U labeled as X in a forest is associated with a phrase whose syntax is represented by a tag sequence ts. Once a grammar is learned, we

[61] The emission rules can be handled similarly.

can compute the posterior probability that the latent category of node U is X_i as follows:

$$\mathrm{P}(X_i|ts) = \frac{\mathrm{P}_{\mathrm{OUT}}(U_i)\mathrm{P}_{\mathrm{IN}}(U_i)}{\mathrm{P}_{\mathrm{IN}}(R)}$$

and add this probability to the expected count $c(X_i, ts)$ that the tag sequence ts belongs to latent category X_i. The probability that the latent category of ts is X_i is then calculated as follows:

$$\mathrm{p}_{ts}(X_i) = \frac{c(X_i, ts)}{\sum_i c(X_i, ts)}$$

As described in Section 8.3, the distributions of latent categories for tag sequences are used to compute the syntactic feature vectors for the X nonterminals of SCFG rules.

8.6 Experiments

8.6.1 Setup

We conduct experiments on two tasks, English-to-German and English-to-Chinese, both involving speech-to-speech translation. The training data for the English-to-German task is a filtered subset of the Europarl corpus [85], containing ∼300K parallel bitext with ∼4.5M tokens on each side. The development and test sets both contain 1K sentences with one reference for each. The training data for the English-to-Chinese task is collected from transcription and human translation of conversations in travel domain. It consists of ∼500K parallel bitext with ∼3M

tokens[62] on each side. Both development and test sets contain ∼1.3K sentences, each with two references. Both corpora are also preprocessed with punctuation removed and words down-cased to make them suitable for speech translation.

The baseline system is our implementation of the hierarchical phrase-based model of Chiang [40], and it includes basic features such as rule and lexicalized rule translation probabilities, language model scores, rule counts, etc. We use 4-gram language models in both tasks, and conduct minimum-error-rate training [117] to optimize feature weights on the development set. Our baseline hierarchical model has 8.3M and 9.7M rules for the English-to-German and English-to-Chinese tasks, respectively.

The English side of the parallel data is parsed using our PCFG-LA parser trained on the combination of Broadcast News treebank from Ontonotes [157] and a speechified[63] version of the WSJ treebank [104] to achieve higher parsing accuracy [73].

8.6.2 Results

Our algorithm identifies ∼180K unique tag sequences for the English side of phrase pairs in both tasks. Table 8.2 shows examples for which our syntactic feature vector representation is able to identify similar and dissimilar tag sequences. For instance, it determines that the sequence "DT JJ NN" is syntactically very similar to "DT ADJP NN" and very dissimilar to "NN CD VP". Note that our

[62]The Chinese sentences are automatically segmented into words. However, BLEU scores are computed at character level for tuning and evaluation.

[63]Symbolic expressions are replaced with verbal forms (e.g., $5 was replaced with five dollars), and punctuation and case are removed.

	Very similar $\vec{F}(ts) \cdot \vec{F}(ts') > 0.9$	**Not so similar** $0.4 \leq \vec{F}(ts) \cdot \vec{F}(ts') \leq 0.6$	**Very dissimilar** $\vec{F}(ts) \cdot \vec{F}(ts') < 0.1$
DT JJ NN	DT NN DT JJ JJ NN DT ADJP NN	DT JJ JJ NML NN DT JJ CC INTJ VB DT NN NN JJ	PP NP NN NN CD VP RB NP IN CD
VP	VB VB RB VB PP VB DT DT NN	VP PP JJ NN VB NN NN VB VB RB IN JJ	JJ NN TO VP JJ WHNP DT NN IN INTJ NP
ADJP	JJ PDT JJ RB JJ	ADJP JJ JJ CC ADJP VB JJ JJ ADVP WHNP JJ	ADJP IN NP JJ AUX RB ADJP ADJP VP

Table 8.2: Examples of similar and dissimilar tag sequences

latent categories are learned automatically to maximize the likelihood of the training forests extracted based on alignment and are not explicitly instructed to discriminate between syntactically different tag sequences. Our approach is not guaranteed to always assign similar feature vectors to syntactically similar tag sequences; however, as shown in Table 8.3 for the English-to-German task and in Table 8.4 for the English-to-Chinese task, the latent categories are able to capture some similarities among tag sequences that are beneficial for translation. The addition of the syntactic feature achieves a statistically significant improvement of 0.6 ($p \leq 0.01$) in BLEU on the test set of the English-to-German task. This improvement is substantial given that only one reference is used for each test sentence. On the English-to-Chinese task, the syntax feature achieves a smaller improvement of 0.41 BLEU on the test set. One potential explanation for the smaller improvement is that the sentences on the English-to-Chinese task are much shorter, with an average of only 6 words per sentence, compared to 15 words in the English-to-German task. The hypothesis

	Baseline	+Syntax	Δ
Dev	16.26	17.06	0.80
Test	16.41	17.01	0.60

Table 8.3: BLEU scores of the English-to-German task (one reference)

	Baseline	+Syntax	Δ
Dev	46.47	47.39	0.92
Test	45.45	45.86	0.41

Table 8.4: BLEU scores of the English-to-Chinese task (two references)

space of translating a longer sentence is much larger than that of a shorter sentence; therefore, there is more potential gain from using syntactic features to rule out unlikely derivations of longer sentences. On the other hand, phrasal rules may be adequate for shorter sentences, leaving less room for syntax to help in the case of the English-to-Chinese task.

8.6.3 Discussion

The incorporation of the syntactic feature into the hierarchical phrase-based translation system increases the memory load and computational cost. In the worst case, our algorithm must store one feature vector for each tag sequence and one feature vector for each nonterminal of an SCFG rule, with the latter using the majority of the extra memory storage. We observed that about 90% of the X nonterminals in the rules have only one tag sequence, and thus the required memory space can be significantly reduced by only storing a pointer to the feature vector of the tag sequence for these nonterminals. Our algorithm also requires computing one dot-product of two feature vectors for each nonterminal when an SCFG rule

is applied to a source span. This cost can be reduced, however, by caching the dot-products of the tag sequences that are frequently accessed.

There have been other successful approaches that impose soft syntactic constraints to hierarchical phrase-based models by either introducing syntax-based rule features such as the prior derivation model of Zhou et al. [168] or by imposing constraints on translation spans at decoding time, e.g., (Marton and Resnik [105], Xiong et al. [159, 160]). These approaches are all orthogonal to ours, and it is expected that they could be combined with our approach to achieve greater improvement.

8.7 Conclusions

We have presented a novel method to enhance hierarchical phrase-based machine translation systems with real-valued linguistically motivated feature vectors. Our method maintains the advantages of hierarchical phrase-based translation systems, while at the same time naturally incorporating soft syntactic constraints. Experimental results show that this approach improves the baseline hierarchical phrase-based translation models on both English-to-German and English-to-Chinese tasks.

Chapter 9

Contributions and Future Work

In this chapter, we summarize our contributions and outline directions for future work.

9.1 Contributions

We have discussed the strong independence assumptions made by traditional models for natural language processing and discussed how these independence assumptions can be corrected at least in part by introducing latent variables. In contrast to traditional models that often have a fixed parameterization, latent variable models are able to automatically learn complex dependencies in a data-driven way and have the flexibility to adjust the number of parameters based on the type and the amount of training data available to learn the most important dependencies. We have created several different types of latent variable models to capture dependencies that otherwise could not be captured using the conventional Markov, hidden Markov, context-free, and synchronous context-free assumptions, and applied these models to a diverse set of natural language processing applications, including POS tagging, language modeling, parsing, and machine translation.

POS Tagging: We have created a latent bigram POS tagger that significantly outperforms conventional bigram and trigram HMM taggers when evaluated on

both the Chinese Penn treebank and the English WSJ Penn treebank.

Language Modeling: We have created a latent language model that achieves a lower perplexity than a strong trigram word language model when evaluated on the English WSJ Penn treebank.

Parsing: We have improved a state-of-the-art PCFG-LA parser by developing a language-independent rare word smoothing method and a heuristic Chinese OOV word handling method.

Machine Translation: We have created a novel approach to induce latent syntactic categories to capture the syntactic dependencies inherent in the hierarchical structure of phrase pairs. We have also developed an effective method to improve hierarchical phrase-based translation models using soft syntactic constraints based on the latent syntactic categories.

To further improve our models, we have also created and evaluated three different methods for improving the performance of latent variable models. Although our experiments focused on parsing with PCFG-LA grammars, these methods can be applied to any of the other applications of latent variable modeling.

- We have investigated self-training for our latent bigram taggers and PCFG-LA grammars and found that these models benefit more significantly from self-training than conventional models whose parameterization is fixed. We conclude that the success of latent variable models with self-training is due to their flexibility to adjust their model parameterization to learn more accu-

rate models from the additional automatically labeled training data. Using self-training, we have obtained state-of-the-art parsing accuracies for a single parser on the English WSJ treebank (91.5% F) and the Chinese Penn treebank (85.2% F).

- We have created methods to effectively combine product models, which take advantage of the variability among latent variable models, and self-training to further improve parsing accuracy. We have found that the combination of these two approaches provides an effective avenue to utilize large quantities of automatically labeled data to train very high quality parsing models with accuracies of 92.5% F on the English WSJ treebank and 89.6% F on the English Broadcast News treebank.

- We have created a feature-rich log-linear lexical model for PCFG-LA grammars to provide a principled solution to address data sparsity, handle out-of-vocabulary (OOV) words, and exploit overlapping features during model induction. With this additional method, we found the resulting PCFG-LA grammars have the flexibility to incorporate overlapping features and are able to more accurately model different languages, achieving significant absolute improvements of 1%, 1.7%, and 2.7% in F score on English, Chinese, and Arabic, respectively.

9.2 Future Work

There are several possible directions for future research based on this thesis.

- Given that self-training is effective for training more accurate individual PCFG-LA grammars and the product model is able to exploit the variability among individual grammars for enhanced performance, it should be possible to iterate the process of self-training new PCFG-LA grammars based on the output of the product model of the self-trained grammars trained at the current iteration . We attempted this using the full WSJ Penn treebank training data and a large amount of unlabeled data and found that the second iteration does not provide further improvement. However, we observed that the second iteration does provide a significant improvement in a preliminary experiment that uses a much smaller set of gold standard training data (i.e., sections 2 and 3 of the WSJ Penn treebank). This bootstrapping training method is worth further investigation, especially for low-resource languages.

- The feature-rich log-linear lexical model for PCFG-LA grammars supports any local features that can be extracted from the pair (t_x, w). In addition to the basic features that we studied, language-dependent features studied in [1] and features related to word semantics (e.g., using WordNet [52]) or word clusters (e.g., using unsupervised clustering [21, 59, 86]) might also be beneficial for improving performance. Features extracted from (t, w) could also provide smoothing across the latent tags. Moreover, it might be beneficial to perform feature selection prior to training. It is also expected that even more accurate parsers can be produced by using this approach together with self-training and/or product models.

- The methods for improving the PCFG-LA parser can also be applied to improve the other latent variable models that we have developed. The latent bigram tagger, which has been found to benefit from self-training like the PCFG-LA parser, can also directly benefit from both product models and the feature-rich log-linear lexical model. While the latent language model uses POS tags as the initial classes for learning fine-grained tags, the training sentences do not have to be manually labeled. Indeed, automatically induced word clusters can also be used as the initial classes. This would make it possible to train a latent language model on a large amount of training data for different languages. These methods can also be applied to other latent variable models, e.g., in speech recognition [78, 127].

- While our PCFG-LA grammars are able to achieve very high parsing accuracies for a variety of languages, the performance levels do not come without a cost. Despite the fact that we have already parallelized both the training and parsing algorithms using multi-threading, the self-trained product models require a significant amount of memory and time to train and to parse larger data sets. On the one hand, cloud computing techniques could be utilized to speed up training and decoding. On the other hand, it is also possible to exploit methods to reduce the size of the grammars with minimal loss in parsing accuracy. This could be potentially achieved by using the multi-scale approach [124] to tie rule parameters during grammar induction or by merging similar latent tags after a grammar is trained.

- Our work on using latent syntactic categories to enhance hierarchical phrase-based translation models can be continued in many directions. First, while the current approach imposes soft syntactic constraints between the parse structure of the source sentence and the SCFG rules used to derive the translation, the real-valued syntactic feature vectors can also be used to impose soft constraints between SCFG rules during rule rewrite. In this case, target side parse trees could also be used alone or together with the source side parse trees to induce the latent syntactic categories. Second, instead of using single parse trees during both training and decoding, our approach is likely to benefit from utilizing parse forests as in [110]. Third, in addition to the treebank categories obtained by syntactic parsing, lexical cues [170] directly available in sentence pairs could also be exploited using the feature-rich approach to guide the learning of latent categories. Last but not the least, it would be interesting to investigate discriminative training approaches to learn latent categories that directly optimize on translation quality.

Bibliography

[1] Mohammed Attia, Jennifer Foster, Deirdre Hogan, Joseph Le Roux, Lamia Tounsi, and Josef van Genabith. Handling unknown words in statistical latent-variable parsing models for Arabic, English and French. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference*, 2010.

[2] Lalit Bahl, James Baker, Paul Cohen, Frederick Jelinek, Burn Lewis, and Robert L. Mercer. Recognition of continuously read natural corpus. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1978.

[3] Jason Baldridge and Miles Osborne. Active learning and logarithmic opinion pools for HPSG parse selection. *Natural Language Engineering*, 2008.

[4] Srinivas Bangalore. 'Almost parsing' technique for language modeling. In *Proceedings of the International Conference on Spoken Language Processing*, 1996.

[5] David J. Bartholomew and Martin Knott. *Latent variable models and factor analysis*. Charles Griffin & Co. Ltd, 1987.

[6] Alexandria T. Basilevsky. *Statistical factor analysis and related methods*. John Wiley & Sons, Inc., 1994.

[7] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 1970.

[8] Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. Painless unsupervised learning with features. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2010.

[9] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 1996.

[10] Daniel M. Bikel and David Chiang. Two statistical parsing models applied to the chinese treebank. In *Proceedings of the Second Chinese Language Processing Workshop*, 2000.

[11] Christopher M. Bishop. Latent variable models. *Learning in Graphical Models*, 1999.

[12] Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, 1973.

[13] Sean Borman. The expectation maximization algorithm: A short tutorial, 2004.

[14] Léon Bottou. *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis, Université de Paris XI, 1991.

[15] Thorsten Brants. TnT a statistical part-of-speech tagger. In *Proceedings of the Conference on Applied Natural Language Processing*, 2000.

[16] Leo Breiman. Bagging predictors. *Machine Learning*, 1996.

[17] Broňa Brejová, Daniel G. Brown, and Tomáš Vinař. The most probable labeling problem in HMMs and its application to bioinformatics. *Algorithms in Bioinformatics*, 2004.

[18] Eric Brill. Some advances in transformation-based part of speech tagging. In *Proceedings of the National Conference on Artificial Intelligence*, 1994.

[19] Daniel G. Brown and Jakub Truszkowski. New decoding algorithms for hidden markov models using distance measures on labellings. *BMC Bioinformatics*, 2010.

[20] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 1990.

[21] Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 1992.

[22] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 1993.

[23] Kenneth P. Burnham and David R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer Verlag, 2002.

[24] Hadumod Bussmann. *Routledge dictionary of language and linguistics*. Routledge, 1996.

[25] Olivier Cappé, Eric Moulines, and Tobias Ryden. *Inference in hidden markov models*. Springer Verlag, 2005.

[26] Xavier Carreras, Michael Collins, and Terry Koo. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Conference on Computational Natural Language Learning*, 2008.

[27] Pi-Chuan Chang, Michel Gally, and Christopher Manning. Optimizing chinese word segmentation for machine translation performance. In *Proceedings of the Workshop on Statistical Machine Translation*, 2008.

[28] Eugene Charniak. Expected-frequency interpolation. Technical report, Department of Computer Science, Brown University, 1996.

[29] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 1997.

[30] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the National Conference on Artificial Intelligence*, 1997.

[31] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2000.

[32] Eugene Charniak and Mark Johnson. Edit detection and parsing for transcribed speech. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2001.

[33] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2005.

[34] Eugene Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz. Equations for part-of-speech tagging. In *Proceedings of the National Conference on Artificial Intelligence National Conference on Artificial Intelligence*, 1993.

[35] Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. *BLLIP 1987-89 WSJ corpus release 1.* Linguistic Data Consortium, 2000.

[36] Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. Multilevel coarse-to-fine PCFG parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference*, 2006.

[37] Stanley F. Chen. Conditional and joint models for grapheme-to-phoneme conversion. In *Proceedings of the European Conference on Speech Communication and Technology*, 2003.

[38] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical report, Harvard University, 1998.

[39] Stanley F. Chen, Douglas Beeferman, and Ronald Rosenfeld. Evaluation metrics for language models. In *DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[40] David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 2007.

[41] David Chiang. Learning to translate with source and target syntax. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2010.

[42] David Chiang, Mona Diab, Nizar Habash, Owen Rambow, and Safiullah Shareef. Parsing Arabic dialects. In *Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

[43] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 1956.

[44] Noam Chomsky. *Aspects of the theory of syntax*. MIT Press, 1965.

[45] Stephen Clark, James R. Curran, and Miles Osborne. Bootstrapping POS taggers using unlabelled data. In *Proceedings of the Conference on Computational Natural Language Learning*, 2003.

[46] John Cocke and Jacob T. Schwartz. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.

[47] Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 2005.

[48] Michael John Collins. *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania, 1999.

[49] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd revised edition edition, 2001.

[50] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 1977.

[51] Steven J. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 1988.

[52] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[53] Denis Filimonov and Mary Harper. A joint language model with fine-grain syntactic tags. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

[54] Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2008.

[55] Victoria Fossum and Kevin Knight. Combining constituent parsers. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

[56] Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. What's in a translation rule. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2004.

[57] John Garofolo, Jonathan Fiscus, William Fisher, and David Pallett. *CSR-IV HUB4*. Linguistic Data Consortium, 1996.

[58] Joshua T. Goodman. A bit of progress in language modeling. *Computer Speech and Language*, 2001.

[59] Amit Goyal and Hal Daume. Approximate scalable bounded space sketch for large data NLP. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[60] Barbara B. Green and Gerald M. Rubin. Automated grammatical tagging of english. Technical report, Depart of Linguistics, Brown University, 1971.

[61] Spence Green and Christopher D. Manning. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proceedings of the International Conference on Computational Linguistics*, 2010.

[62] Mary Harper and Zhongqiang Huang. Chinese statistical parsing. In Joseph Olive, Caitlin Christianson, and John McCary, editors, *Handbook of Natural Language Processing and Machine Translation*. Springer Verlag, 2011.

[63] Mary P. Harper, Bonnie J. Dorr, John Hale, Brian Roark, Izhak Shafran, Matthew Lease, Yang Liu, Matthew Snover, Lisa Yung, Anna Krasnyanskaya, and Robin Stewart. Johns Hopkins summer workshop final report on parsing and spoken structural event detection. Technical report, Johns Hopkins Summer Workshop, 2005.

[64] Steffen Heber and Jens Stoye. Finding all common intervals of k permutations. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, 2001.

[65] Peter A. Heeman. POS tags and decision trees for language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1999.

[66] Peter A. Heeman and James F. Allen. Speech repairs, intonational phrased and discourse markers: Modeling speakers' utterances in spoken dialogue. *Computational Linguistics*, 1999.

[67] Geoffrey E. Hinton. Products of experts. In *Proceedings of the International Conference on Artificial Neural Networks*, 1999.

[68] Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2008.

[69] Liang Huang and David Chiang. Better k-best parsing. In *International Workshop on Parsing Technology*, 2005.

[70] Liang Huang, Kevin Knight, and Aravind Joshi. A syntax-directed translator with extended domain of locality. In *Proceedings of the Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, 2006.

[71] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

[72] Zhongqiang Huang, Mary Harper, and Wen Wang. Mandarin part-of-speech tagging and discriminative reranking. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2007.

[73] Zhongqiang Huang, Mary Harper, and Slav Petrov. Self-training with products of latent variable. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2010.

[74] Jonathon Hull. Combining syntactic knowledge and visual text recognition: A hidden markov model for part of speech tagging in a word recognition algorithm. In *AAAI Symposium: Probabilistic Approaches to Natural Language*, 1992.

[75] W. John Hutchins and Harold L. Somers. *An introduction to machine translation*. Academic Press, 1992.

[76] Philip M. Lewis II and Richard E. Stearn. Syntax directed transduction. *Journal of the ACM*, 1968.

[77] Rukmini M. Iyer and Mari Ostendorf. Modeling long distance dependence in language: topic mixtures versus dynamic cache models. *IEEE Transactions on Speech and Audio Processing*, 2002.

[78] Frederick Jelinek. *Statistical methods for speech recognition*. MIT Press, 1997.

[79] Fernando Pereira John Lafferty, Andrew McCallum. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, 2001.

[80] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab, 1965.

[81] Mark D. Kernighan, Kenneth W. Church, and William A. Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 1990.

[82] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2003.

[83] Sheldon Klein and Robert F. Simmons. A computational approach to grammatical coding of english words. *Journal of the ACM*, 1963.

[84] Reinhard Kneser. Statistical language modeling using avariable context length. In *Proceedings of the International Conference on Spoken Language Processing*, 1996.

[85] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*, 2005.

[86] Terry Koo, Xavier Carrera, and Michael Collins. Simple semi-supervised dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2008.

[87] Anders Krogh. Hidden markov models for labeled sequences. In *Proceedings of the International Conference on Computer Vision Image Processing*, 1994.

[88] Anders Krogh. Two methods for improving performance of an hmm and their application for gene finding. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, 1997.

[89] Anders Krogh, Björn Larsson, Gunnar von Heijne, and Erik L. Sonnhammer. Predicting transmembrane protein topology with a hidden markov model: Application to complete genomes. *Journal of Molecular Biology*, 2001.

[90] Thomas Kuhn, Heinrich Niemann, and Ernst G. Schukat-Talamazzini. Ergodic hidden markov models and polygrams for language modeling. In *Processing of the International Conference on Acoustics, Speech and Signal Processing*, 1994.

[91] Karim Lari and Steve J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 1990.

[92] Roger Levy and Galen Andrew. Tregex and tsurgeon: Tools for querying and manipulating tree data structures. In *Proceedings of the Language Resources and Evaluation Conference*, 2006.

[93] Roger Levy and Christopher Manning. Is it harder to parse chinese, or the chinese treebank. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2003.

[94] Percy Liang and Dan Klein. Analyzing the errors of unsupervised learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2008.

[95] Percy Liang, Slav Petrov, Michael I. Jordan, and Dan Klein. The infinite PCFG using hierarchical dirichlet processes. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2007.

[96] Pietro Liò, Nick Goldman, Jeffrey L. Thorne, and David T. Jones. Passml: combining evolutionary inference and protein secondary structure prediction. *Bioinformatics*, 1998.

[97] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 1989.

[98] Yang Liu, Andreas Stolcke, Elizabeth Shriberg, and Mary Harper. Using conditional random fields for sentence boundary detection in speech. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2005.

[99] Yang Liu, Qun Liu, and Shouxun Lin. Tree-to-string alignment template for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2006.

[100] Mohamed Maamouri, Ann Bies, Sondos Krouna, Fatma Gaddeche, and Basma Bouziri. Penn Arabic treebank guidelines. Technical report, Linguistic Data Consortium, 2009.

[101] Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[102] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Combridge University Press, 2008.

[103] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 1993.

[104] Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. *Treebank-3*. Linguistic Data Consortium, 1999.

[105] Yuval Marton and Philip Resnik. Soft syntactic constraints for hierarchical phrased-based translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2008.

[106] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2005.

[107] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2006.

[108] Iain A. McCowan, Darren Moore, John Dines, Daniel Gatica-Perez, Mike Flynn, Pierre Wellner, and Hervé Bourlard. On the use of information retrieval measures for speech recognition evaluation. Technical report, IDIAP, 2004.

[109] Marie Meteer and J. Robin Rohlicek. Statistical language modeling combining n-gram and context-free grammars. In *Processing of the International Conference on Acoustics, Speech and Signal Processing*, 1993.

[110] Haitao Mi and Liang Huang. Forest-based translation rule extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008.

[111] Haitao Mi, Liang Huang, and Qun Liu. Forest-based translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2008.

[112] Andrei Mikheev, Marc Moens, and Claire Grover. Named entity recognition wihtout gazetters. In *Proceedings of the Conference on European Chapter of the Association for Computational Linguistics*, 1999.

[113] Scott Miller, Heidi Fox, Lance Ramshaw, and Ralph Weischedel. A novel use of statistical parsing to extract information from text. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference*, 2000.

[114] Roger K. Moore. Evaluating speech recognizers. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1977.

[115] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 1994.

[116] Thomas Niesler and Phil Woodland. Variable-length category-based n-gram language model. In *Processing of the International Conference on Acoustics, Speech and Signal Processing*, 1996.

[117] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2003.

[118] Franz Josef Och and Hermann Ney. Improved statistical alignment models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2000.

[119] Franz Josef Och and Hermann Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 2004.

[120] Jerome Packard. *The Morphology of Chinese*. Cambridge University Press, 2000.

[121] Slav Petrov. *Coarse-to-fine natural language processing*. PhD thesis, University of California at Bekeley, 2009.

[122] Slav Petrov. Products of random latent variable grammars. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2010.

[123] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2007.

[124] Slav Petrov and Dan Klein. Sparse multi-scale grammars for discriminative latent variable parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008.

[125] Slav Petrov and Dan Klein. Discriminative log-linear grammars with latent variables. In *Advances in Neural Information Processing Systems*, 2008.

[126] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2006.

[127] Slav Petrov, Adam Pauls, and Dan Klein. Learning structured models for phone recognition. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

[128] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval*, 1998.

[129] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in Speech Recognition*, 1990.

[130] Lawrence R. Rabiner and Biing-Hwang Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 1986.

[131] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1996.

[132] Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1997.

[133] Roi Reichart and Ari Rappoport. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2007.

[134] Brian Roark, Mary Harper, Yang Liu, Robin Stewart, Matthew Lease, Matthew Snover, Izhak Shafran, Bonnie J. Dorr, John Hale, Anna Krasnyanskaya, and Lisa Yung. Sparseval: Evaluation metrics for parsing speech. In *Proceedings of the Language Resources and Evaluation Conference*, 2006.

[135] Ronald Rosenfeld. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 2000.

[136] Kenji Sagae and Alon Lavie. Parser combination by reparsing. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2006.

[137] Ruslan Salakhutdinov, Sam Roweis, and Zoubin Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *Proceedings of the International Conference on Machine Learning*, 2003.

[138] Christer Samuelsson. Morphologial tagging based entirely on bayesian inference. In *Nordic Conference on Computational Linguistics*, 1993.

[139] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 1985.

[140] Andrew Smith and Miles Osborne. Diversity in logarithmic opinion pools. *Lingvisticae Investigationes*, 2007.

[141] Andrew Smith, Trevor Cohn, and Miles Osborne. Logarithmic opinion pools for conditional random fields. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2005.

[142] Rohini Srihari and Charlotte Baltus. Combining statistical and syntactic methods in recognizing handwritten sentences. In *AAAI Symposium: Probabilistic Approaches to Natural Language*, 1992.

[143] Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. Bootstrapping statistical parsers from small datasets. In *Proceedings of the Conference on European Chapter of the Association for Computational Linguistics*, 2003.

[144] Andreas Stockle. SRILM – an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, 2002.

[145] Mihai Surdeanu, Sanda Harabagiu, John Williams, and Paul Aarseth. Using predicate-argument structures for information extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2003.

[146] Lucien Tesnière. *Éléments de syntaxe structurale*. Klincksieck, 1959.

[147] Scott M. Thede and Mary P. Harper. A second-order hidden markov model for part-of-speech tagging. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 1999.

[148] Kristina Toutanova and Christopher D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2000.

[149] Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2003.

[150] Huihsin Tseng, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. A conditional random field word segmenter. In *Proceedings of the SIGHAN Workshop on Chinese Language Processing*, 2005.

[151] Huihsin Tseng, Daniel Jurafsky, and Christopher Manning. Morphological features help pos tagging of unknown words across language varieties. In *Proceedings of the SIGHAN Workshop on Chinese Language Processing*, 2005.

[152] Ashish Venugopal, Andreas Zollmann, Noah A. Smith, and Stephan Vogel. Preference grammars: softening syntactic constraints to improve statistical machine translation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

[153] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 1967.

[154] Atro Voutilainen. A syntax-based part-of-speech analyser. In *Proceedings of the Conference on European Chapter of the Association for Computational Linguistics*, 1995.

[155] Wen Wang and Mary P. Harper. The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2002.

[156] Wen Wang, Zhongqiang Huang, and Mary Harper. Semi-supervised learning for part-of-speech tagging of mandarin transcribed speech. In *Processing of the International Conference on Acoustics, Speech and Signal Processing*, 2007.

[157] Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, and Ann Houston. *OntoNotes release 2.0*. Linguistic Data Consortium, 2008.

[158] Ian H. Witten and Timothy C. Bell. Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 1991.

[159] Deyi Xiong, Min Zhang, Aiti Aw, and Haizhou Li. A syntax-driven bracketing model for phrase-based translation. In *Proceedings of the Joint Conference of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, 2009.

[160] Deyi Xiong, Min Zhang, and Haizhou Li. Learning translation boundaries for phrase-based decoding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2010.

[161] Nianwen Xue, Fu-dong Chiou, and Martha Palmer. Building a large-scale annotated chinese corpus. In *Proceedings of the International Conference on Computational Linguistics*, 2002.

[162] Nianwen Xue, Fei Xia, Fu-dong Chiou, and Marta Palmer. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 2005.

[163] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2001.

[164] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 1967.

[165] Bin Zhang and Jeremy G. Kahn. Evaluation of decatur text normalizer for language model training. Technical report, University of Washington, 2008.

[166] Hao Zhang, Daniel Gildea, and David Chiang. Extracting synchronous grammar rules from word-level alignments in linear time. In *Proceedings of the International Conference on Computational Linguistics*, 2008.

[167] Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. K-best combination of syntactic parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

[168] Bowen Zhou, Bing Xiang, Xiaodan Zhu, and Yuqing Gao. Prior derivation models for formally syntax-based translation using linguistically syntactic parsing and tree kernels. In *Proceedings of the Workshop on Syntax and Structure in Statistical Translation*, 2008.

[169] Andreas Zollmann and Ashish Venugopal. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, 2006.

[170] Andreas Zollmann and Stephan Vogel. A word-class approach to labeling pscfg rules for machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2011.