SELF-TRAINED DISCRIMINATIVE CONSTITUENCY PARSER WITH HIERARCHICAL JOINT LEARNING APPROACH

by

Arda Çelebi

B.S., Computer Engineering and Information Science, Bilkent University, 2002

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2012

SELF-TRAINED DISCRIMINATIVE CONSTITUENCY PARSER WITH HIERARCHICAL JOINT LEARNING APPROACH

APPROVED BY:

Assist. Prof. Arzucan Özgür .....
(Thesis Supervisor)

Assoc. Prof. Brian Roark .....

Suzan Üsküdarlı, Ph.D. .....

DATE OF APPROVAL: 27.07.2012

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis supervisor Assist. Prof. Arzucan Özgür for her guidance and support in this work. I appreciate her very much for being so helpful, understanding and patient to me.

I would also like to thank Assoc. Prof. Brian Roark and Ph.D. Suzan Üsküdarlı for kindly accepting to be in my thesis committee and for their support and guidance afterward.

I am also indebted to the many faculty who provided me computers to run my experiments throughout my thesis. Thanks to Assoc. Prof. Murat Saraçlar, Prof. Ethem Alpaydın and Assoc. Prof. Tunga Güngör.

I am deeply thankful for the help and support I have received from other fellow graduate students. My thanks goes to Haşim Sak, Göker Erdoğan and Murat Semerci.

Above all, I consider myself lucky to have such a wonderful family. My whole family stood by me at all times and supported all my decisions in every possible way. I'm deeply grateful to my mother and father for trusting me fully and believing in my success. Mankind hasn't been able to come up with the word or sentence that I can express my love for them. Nevertheless, I don't need to tell them what they already know.

ABSTRACT

SELF-TRAINED DISCRIMINATIVE CONSTITUENCY PARSER WITH HIERARCHICAL JOINT LEARNING APPROACH

Determining the syntactic structure of a sentence is a fundamental step towards understanding what is conveyed in that sentence. The syntactic parse tree of a sentence can be used in several tasks such as information extraction, machine translation, summarization and question answering. Therefore, syntactic parsing has been one of the most studied topics in the literature. Today's top performing parsers employ statistical approaches and achieve over 90% accuracy. While statistical approaches reach their highs in supervised settings, semi-supervised approaches like self-training of parsers is starting to emerge as a next challenge. Such parsers train on their own outputs with the goal of achieving better results by learning on their own. However, only a small number of self-trained parsers have met this goal so far. In this thesis, we tackle the problem of self-training a feature-rich discriminative constituency parser, which to our knowledge has never been studied before. We approach the self-training problem with the assumption that we can't expect the whole parse tree given by a parser to be completely correct but, rather, some parts of it are more likely to be. We hypothesize that instead of feeding the parser the whole guessed parse trees of its own, we can break them down into smaller ones, namely n-gram trees, and perform self-training on them. We thus have an n-gram parser and transfer the distinct expertise of the n-gram parser to the full sentence parser by using the Hierarchical Joint Learning (HJL) approach. The resulting parser is called a jointly self-trained parser. We first study joint learning in completely supervised setting and observe slight improvement of the jointly trained parser over the baseline. When the real n-gram trees are replaced with guessed ones, the resulting jointly self-trained parser performs no differently than the baseline.

ÖZET

HİYERARŞİK BİRLİKTE ÖĞRENME YÖNTEMİYLE KENDİ KENDİNİ EĞİTEN AYIRDEDİCİ SÖZDİZİM ÇÖZÜMLEYİCİSİ

Cümlede geçen kelimelerin dilbilgisi kurallarına uygun olarak o cümleyi nasıl meydana getirdiklerini bulmak o cümleyi anlamak için en önemli adımlardan biridir. Sözdizim çözümleyicisi ile elde edeceğimiz bu bilgiyle örneğin cümle içinden istenilen bilgileri çıkartabilir veya o cümleyi başka bir dile çevirebiliriz. Bundandır ki sözdizim çözümleyicileri doğal dil işleme alanında en çok çalışılan konulardandır. Bugün en iyi çözümleyiciler denetimli istatistiksel yöntemleri kullanarak %90 başarı seviyelerini aşsalar da, kendi kendisini eğitebilen yarı-denetimli modelle çalışan çözümleyiciler ile yapılan başarılı çalışma sayısı çok azdır. Bu tür çözümleyiciler eğitim verilerini kendi çıktıları ile genişletip kendi kendilerine öğrenerek daha iyi sonuç almaya çalışırlar. Bu tezde amacımız daha önce literatürde yapılmamış birçok öznitelik kullanarak kendi kendisini eğitebilen ayırtedici sözdizim çözümleyicisi geliştirmektir. Bunu yapmak için çözümleyicinin çıktı olarak verdiği sözdizim ağacının tamamının doğru olmadığı fakat bazı parçalarının, ki biz bunlara n-gram ağaçları diyoruz, doğru olabileceğini varsayıyoruz. Buna göre hipotezimiz ise bu parçalar ile çözümleyicinin kendi kendisini eğitebilmesiyle daha iyi sonuç alabileceğimizdir. n-gram'ların çözümlemesini buna özgü n-gram çözümleyicisi ile yapacağımızdan, asıl çözümleyicinin kendi kendisini eğitmesi yerine n-gram çözümleyicisi asıl çözümleyicinin çıktısı ile eğitilecektir. Bu esnada da bu iki çözümleyicinin hiyerarşik birlikte öğrenme yöntemiyle birbirlerine öğrendiklerini aktarmaları sağlanarak asıl çözümleyicinin beraberce kendi kendisine öğrenmesi sağlanır. Sadece gerçek eğitim verileri ile yapılan deneylerde asıl çözümleyicinin başarısının n-gram çözümleyici ile eğitildiğinde arttığı, eğitim verisine asıl çözümleyicinin çıktıları eklendiğinde ise beraberce kendi kendisini eğitemediği gözlemlenmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . ix
LIST OF TABLES . . . x
LIST OF SYMBOLS . . . xiii
LIST OF ACRONYMS/ABBREVIATIONS . . . xiv
1. INTRODUCTION . . . 1
1.1. Problem Statement . . . 2
1.2. Motivation and Objective . . . 3
1.3. Road Map . . . 4
2. BACKGROUND AND RELATED WORK . . . 5
2.1. Natural Language Parsing . . . 5
2.1.1. Grammar . . . 7
2.1.2. Grammar Binarization and Markovization . . . 8
2.1.3. Grammar Annotation . . . 9
2.1.4. Evaluation . . . 10
2.2. n-gram Trees . . . 12
2.3. Hierarchical Joint Learning and Parsing . . . 13
2.4. Self-Training for Parsing . . . 14
2.5. Data . . . 16
2.5.1. Penn Treebank . . . 16
2.5.2. Reuters Corpus . . . 17
3. DISCRIMINATIVE CONSTITUENCY PARSING ON FULL SENTENCES 18
3.1. Parsing with Conditional Random Fields . . . 18
3.1.1. Objective Function and Parameter Estimates . . . 19
3.2. Optimizing the Parser . . . 21
3.2.1. Parallelization . . . 21
3.2.2. Chart Caching . . . 22

3.3. Features . . . 22
3.3.1. Lexicon Features . . . 23
3.3.2. Grammar Features . . . 24
3.3.3. Distributional Similarity Clusters . . . 25
3.3.4. Unknown Word Classes . . . 26
3.4. Data . . . 26
3.5. Experiments . . . 27
3.5.1. Accuracy Analysis on Constituents . . . 28
3.5.2. Length- and Height-based Accuracy Analysis of Constituents . . 29
3.6. Discussion . . . 31
4. N-GRAM PARSING . . . 32
4.1. n-gram Trees . . . 32
4.2. Data . . . 34
4.3. Features . . . 37
4.4. Experiments . . . 38
4.4.1. Accuracy Analysis of Constituents . . . 41
4.4.2. Length- and Height-based Accuracy Analysis of Constituents . . 43
4.4.3. Analysis of Incomplete Constituents . . . 45
4.5. Discussion . . . 46
5. HIERARCHICAL JOINT LEARNING OF N-GRAM AND FULL PARSING TASKS . . . 48
5.1. Hierarchical Joint Learning . . . 48
5.1.1. Formal Model . . . 49
5.2. Hierarchical Joint Learning of Full and n-gram Parsing . . . 51
5.3. Data . . . 51
5.4. Experiments . . . 52
5.4.1. Accuracy Analysis on Constituents . . . 54
5.4.2. Length- and Height-based Accuracy Analysis of Constituents . . 56
5.5. Discussion . . . 57
6. SELF-TRAINING WITH N-GRAM TREES . . . 59
6.1. Self-training Pipeline . . . 59

6.2. Data 60
6.3. Experiments 60
6.3.1. Accuracy Analysis on Constituents 62
6.3.2. Length- and Height-based Accuracy Analysis of Constituents . . 63
6.4. Discussion 65
7. CONCLUSION 67
7.1. Future Work 68
REFERENCES 71

LIST OF FIGURES

Figure 1.1. Sample constituency tree from the Penn treebank. 1

Figure 1.2. Sample n-gram tree where n=4. 3

Figure 2.1. Sample CFG rule and its representation in a tree structure. 8

Figure 2.2. Sample binarized parse tree. 9

Figure 2.3. Sample tree and its labelled spans. 10

Figure 2.4. Sample tree substitution grammar rules. 12

Figure 2.5. Sample lexicalized tree adjoining grammar rules. 13

Figure 4.1. All generatively accurate 4-gram trees extracted from the complete parse tree. 33

Figure 4.2. Algorithm to extract and store n-gram trees from a parse tree. . . 34

Figure 5.1. A graphical representation of the hierarchical model. 48

LIST OF TABLES

Table 2.1. F_1 scores of high-performance parsers on the Penn treebank. . . . 7

Table 2.2. Summarization of self-training performances. 15

Table 2.3. Number of sentences for each subversion of the Penn treebank. . . 16

Table 3.1. Lexicon feature templates that we used in this thesis. 23

Table 3.2. Word prefixes that we used in lexicon feature creation. 24

Table 3.3. Word suffixes that we used in lexicon feature creation. 24

Table 3.4. Grammar feature templates that we used in this thesis. 25

Table 3.5. Results on the development set of the Penn treebank. 27

Table 3.6. Results on the test set of the Penn treebank. 28

Table 3.7. Performance on the most common constituents in the WSJ15 dev set. 29

Table 3.8. Performance on the constituents by length in the WSJ15 dev set. . 30

Table 3.9. Performance on the constituents by height in the WSJ15 dev set. . 30

Table 4.1. Number of n-gram trees in each training set. 35

Table 4.2. Number of n-gram trees in each development and test set. 36

Table 4.3. Percentage of most common constituents in each training set. . . . 36

Table 4.4. Percentage of the most common constituents in each development set. 37

Table 4.5. Average number of features in each model. 39

Table 4.6. F_1 scores of the n-gram models on each development and test set. . 40

Table 4.7. Performance of the n-gram parsers (trained with 20000 instances) on the development set. 40

Table 4.8. F_1 scores on the most common constituents for each n-gram model from the development set. 41

Table 4.9. Performance of the n-gram models on the most common constituents in the test set. 42

Table 4.10. Length-based accuracy on constituents for each n-gram model. . . 43

Table 4.11. Height-based accuracy on constituents for each n-gram model. . . . 44

Table 4.12. Accuracy on the incomplete constituents in each n-gram models . . 45

Table 5.1. Averaged F_1 scores of full sentence parser jointly trained with each n-gram model. 53

Table 5.2. Statistics on the top performing jointly trained parsers along with the baseline parser. 54

Table 5.3. F_1 scores on the most common constituents for each n-gram model from the development set. 55

Table 5.4. Length-based accuracy on constituents of jointly trained full parser. 56

Table 5.5. Height-based accuracy on constituents of the jointly trained full parser. 57

Table 6.1. F_1 scores of the jointly trained full parser with self-training n-gram models. 61

Table 6.2. F_1 scores on the most common constituents for each n-gram model from the development set. 63

Table 6.3. Length-based accuracy on constituents of the jointly trained full parser. 64

Table 6.4. Height-based accuracy on constituents of the jointly trained full parser. 64

LIST OF SYMBOLS

$\mathcal{D}_b^{(i)}$	Batch Data in i^{th} Iteration
$\mathcal{L}$	Likelihood
m	Model
r	One-level subtree
$\mathcal{Z}_s$	Partition Function
$\mathcal{D}$	Training Data
$\mathcal{D}_m$	Training Data of mth Model
s	Sentence
$\mathcal{M}$	Set of Models
σ_d^2	Domain Variance
ρ	Grammar Rule
$\theta_{m,i}$	i^{th} Parameter of mth Model
$\theta_{*,i}$	i^{th} Parameter of the Top Model
η	Learning Rate
η_k	Learning Rate at k^{th} Iteration
μ	Mean
μ_i	Mean for i^{th} Parameter of the Top Model
τ	Pass Count
θ	Parameters
θ_m	Parameters of mth Model
θ_k	Parameters at k^{th} Iteration
η_0	Starting Learning Rate
θ_*	Top Model Parameters
σ_*^2	Top Model Variance
σ^2	Variance
σ_m^2	Variance of mth Model

LIST OF ACRONYMS/ABBREVIATIONS

ADJP	Adjective Phrase
ADVP	Adverb Phrase
CCG	Combinatory Categorial Grammar
CFG	Context Free Grammar
CLSP	The Center for Language and Speech Processing
CRF	Conditional Random Field
CRF-CFG	Conditional Random Field Context Free Grammar
CYK	Cocke-Younger-Kasami
DG	Dependency Grammar
DT	Determiner
DOP	Data Oriented Parsing
HJL	Hierarchical Joint Learning
HPSG	Head-driven Phrase Structure Grammar
IE	Information Extraction
JJ	Adjective
LFG	Lexical Functional Grammar
MT	Machine Translation
NA	Not Available
NANC	The North American News Text Corpus
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NN	Nominal Noun
NP	Noun Phrase
OOV	Out-of-Vocabulary
PCFG	Probabilistic Context Free Grammar
POS	Part-of-Speech
PP	Prepositional Phrase
QA	Question Answering

QP	Quantifier Phrase
RCV1	The Reuters Corpus Volume 1
S	Simple declarative clause
SBAR	Clause introduced by a subordinating conjunction
SGD	Stochastic Gradient Descent
TAG	Tree Adjoining Grammar
TSG	Tree Substitution Grammar
VP	Verb Phrase
WSJ	Wall Street Journal
WSJALL	All parse trees in the Penn Treebank
WSJOver15	Parse trees with more than 15 words in the Penn Treebank
WSJ15	Parse trees with no more than 15 words in the Penn Treebank
WSJ40	Parse trees with no more than 40 words in the Penn Treebank

1. INTRODUCTION

Computers are far from being able to understand what is conveyed in a natural language sentence. Detecting the subject or other syntactic structures of a given sentence, which is called syntactic parsing, is one of the required steps to reach that goal. Therefore, it has been one of the most studied topics in the field of Natural Language Processing (NLP).

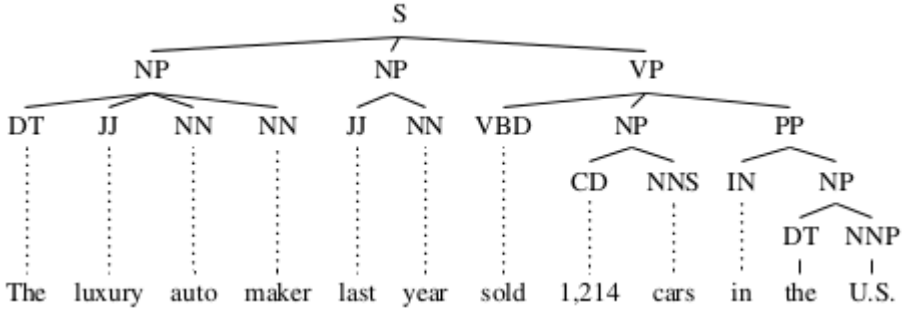

Figure 1.1. Sample constituency tree from the Penn treebank.

Parsing a sentence involves determining how the words in a given order are connected to each other to form that sentence based on the syntax of its language. The syntax describes the rules that govern any sentence structure. During training, a parser observes all the possible syntactic structures seen in the training data set and builds up a model of the syntax. Then, when a new sentence is given, it considers all possible combinations of syntactic rules allowed by that syntax and chooses the best possible combination that describes the whole sentence structure.

Syntax is defined by a grammar formalism, such as context free grammar (CFG) [1, 2], head-driven phrase-structure grammar (HPSG) [3], dependency grammar (DG) [4], or number of others. In this thesis, our parser employs the CFG formalism. As groups of words in CFG formalism form intermediate structures called constituents, parsers using CFG are called constituency parsers. Figure 1.1 shows a sample constituency tree from the Penn treebank [5]. It exemplifies how consecutive words are grouped together to form phrases like a noun phrase (NP), as well as how phrases

become parts of other phrases.

Early parsers in the literature employed rule-based approaches but they were unable to cope with the high level of disambiguity due to the large number of possible parse trees for a single sentence. Over time, researchers have adopted statistical approaches. Such techniques score parse trees and select the best possible one. Hence, they are more suitable for ambiguity resolution. Initially, generative approaches [3, 6] were introduced, which use maximum likelihood estimation to calculate the probability of context-free grammars. Then, discriminative approaches [7, 8] that consider the conditional distribution over the parse trees given a sentence are proposed. More lately, reranking approaches [9, 10] are used as an accompanying second model to the parsing model in order to rerank the top n possible parse tree for a sentence and choose the best one. In the parsing literature, the state-of-the-art constituency parsers achieve an accuracy between 80-95% on various data sets with different characteristics.

Today, we may be far from being able to determine what is conveyed in a sentence. Nevertheless before coming to that, the syntactic analysis of a sentence can be used by various high-level NLP tasks, such as summarization, Information Extraction (IE), Question Answering (QA) and Machine Translation (MT). The accuracy of all these high-level tasks directly depend on the accuracy of the parser. Hence, parsing plays an essential role in high-level NLP tasks.

1.1. Problem Statement

While statistical approaches for the supervised parsers reach their highs, semi-supervised approaches like self-training of parsers is starting to emerge as a next challenge in the field.

A self-trained parser starts its training with a seed set. At some point during its training, it uses what it has learned so far to process newly given sentences and adds the outputted parse trees into its existing training set. It continues to train on an extended training set. If it exceeds the accuracy of the original model, which is trained

on the initial seed training set only, self-training, that is learning from its own output, is achieved. Most of the research done on self-trained parsers failed to improve the original performance. Most recently, McClosky *et al.* [11] achieved an absolute 1.1 F_1 score improvement over the original model with a generative parser and a discriminative reranker. Despite this encouraging success, to our knowledge, there hasn't been any research that achieved self-training of a parser that runs on a discriminative approach with no reranker. This might be due to the fact that discriminative approaches are more complex to train with respect to the generative models.

1.2. Motivation and Objective

In this thesis, we mainly tackle the problem of self-training of a discriminative parser. We approach the self-training problem with the assumption that we can't expect the whole parse tree given by a parser to be completely correct but, rather, some parts of it are more likely to be. We hypothesize that instead of feeding the parser the whole guessed parse trees of its own, we can break down these parse trees into smaller subtrees and perform self-training on them. Throughout this thesis, we refer to those subtrees as n-gram trees.

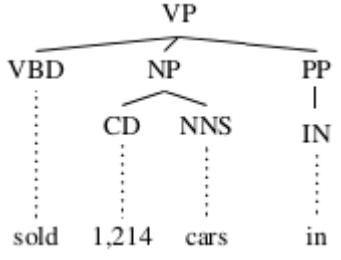

Figure 1.2. Sample n-gram tree where n=4.

Figure 1.2 shows an example n-gram tree (n=4) extracted from a complete parse tree in Figure 1.1 with our n-gram tree extraction algorithm described in Chapter 4. Due to their "not-complete" nature, we consider the n-gram parsing task different from the full sentence parsing task. Thus, we build special n-gram parsers to model them. This means that self-training is performed by an n-gram parser, instead of the full sentence parser. At this point, we employ a Hierarchical Joint Learning approach,

which is described by Finkel *et al.* [12], to train the n-gram parser and the full sentence parser with each other simultaneously. By doing this, we allow the full sentence and self-trained n-gram parsers to *"transfer"* their knowledge to each other during training. This also includes the transfer of updated knowledge of the n-gram parser gained from its own output.

In summary, our objective is to improve the accuracy of a state-of-the-art discriminative parser by using self-training. Instead of using the actual discriminative model of the parser, we use an n-gram based mode that, we believe, can boost the accuracy of the actual model in a joint learning setup.

1.3. Road Map

A brief overview of this thesis follows. First, in the next chapter, we start with giving relevant background information and point out the related works done in the literature. In Chapter 3, we describe how we build our discriminative parser ground-up, which becomes our baseline for the rest of the experiments in this thesis. In the same chapter, we analyze our experimental results in detail on full sentences parsed with our baseline parser. Then, in Chapter 4, we explain the concept of n-gram parsing and the definition of n-gram trees in the context of this thesis. In this chapter, we approach n-gram parsing as a stand-alone parsing task and analyze the results within themselves, as well as with respect to the baseline. After establishing the full sentence and n-gram parsing tasks, Chapter 5 explains how to build an environment where we can jointly train these two types of parser together. We explain the Hierarchical Joint Learning (HJL) approach and analyze the results of the jointly trained full sentence parser. Following that, in Chapter 6 , we show how to change the system from Chapter 5 so as to feed the n-gram parser the output of the full sentence parser and achieve a self-trained n-gram parser as well as a jointly self-trained full sentence parser. We conclude the thesis with further discussion and future work in Chapter 7.

2. BACKGROUND AND RELATED WORK

2.1. Natural Language Parsing

Parsing natural language is the process of characterizing the syntactic descriptions of a sentence. The initial studies on parsing were rule-based approaches with two important algorithms: CYK [13, 14, 15] and Earley [16]. Both use dynamic programming to build charts of a given sentence, considering all possible grammatical descriptions of that sentence allowed by the grammar formalism that the parser uses. However, manually designing grammar rules for such systems is a very tedious task. Furthermore, as the number of possible parses for a given sentence increases, it becomes impractical to implement rule-based parsers due to not being able to consider all possible cases, except for restricted domain-specific tasks. Nowadays, rule-based approaches have been replaced by statistical ones.

Parsers running on statistical approaches try to learn the repeating patterns in the syntactic structures of sentences and capture that information in the form of a statistical model in order to use it later to guess the syntactic structure of a new sentence. Such parsers are capable of ambiguity resolution by first assigning scores to different possible parses and then choosing the highest scored one as the best possible syntactic structure of a given sentence. The first statistical parsers [17, 18, 19] considered statistical parsing as the task of tagging part-of-speeches (POSes), and connecting and labelling constituents. Then, Charniak [6] introduced the probabilistic context free grammars (PCFGs). PCFG parsers use a probabilistic approach to disambiguate different parse trees for the same sentence. However, its strength of ambiguity resolution is limited as described by Briscoe and Carroll [20]. One of the reasons stems from not using word information. Hence, they are called unlexicalized parsers. Collins [3] extended the idea of PCFGs to lexicalized grammars by decorating the grammar rules with head words. Even though such lexicalized parsers achieve high accuracies, they use a large set of features, which cause scalability issues. In this thesis, we use an unlexicalized grammar to prevent such issues and, to still reach the accuracy level of

lexicalized parsers, we annotate our grammar rules with extra information.

Statistical models are grouped as generative and discriminative models. The generative models learn a model of the joint probability of the input and the labels. For example, PCFG-based generative models consider the distribution of input strings $P(x)$ as well as trees, that is labels, given strings $P(y|x)$. However, while the conditional distribution $P(y|x)$ is considered to be important for parsing, the marginal distribution $P(x)$ is not. Hence, generative models waste some of their parameters to model the marginal distribution $P(x)$. That is where discriminative models differs from generative models. Ratnaparkhi [7] introduced the first discriminative model for parsing. It employs a maximum entropy based classifier to choose a parsing action based on the current state and the parsing history. Later on, Charniak [8] presented his maximum entropy-inspired parser.

Relatively more recent studies focused on reranking algorithms, where a baseline model generates a set of candidate output parses and a second model reranks the top n candidates by using more complex features. Such reranking approaches started with the discriminative reranker of Collins [9]. Following that, Charniak and Johnson [10] came up with the maximum entropy based reranker. The most recently introduced reranking algorithm is the forest-based reranking method of Huang [21].

As described in more detail in Section 2.1.4, evaluation of all these mentioned parsers were reported on the Penn treebank corpus [5]. In addition to using the whole corpus, which is referred to as WSJALL, parsers are also trained and evaluated on sentences with no more than 15 words and 40 words, that is WSJ15 and WSJ40 sets respectively. Table 2.1 shows the reported F_1 scores for the parsers on each set. It also gives which specific property each parser has along with whether they employ generative(G) or discriminative(D) approach. To our knowledge, the best scoring parser on the Penn treebank achieves 92.6% F_1 score on the WSJALL set [22]. Apart from these top ranking parsers, we also included the discriminative parser of Finkel *et al.* [12], from which we implemented our discriminative parser. As illustrated in Table 2.1, their parser is not one of the top parsers in terms of accuracy. Nevertheless, the fact

that we need to implement a discriminative parser to show that self-training can be done with discriminative parsers was an important factor for choosing their design. Moreover, their parser's relatively lower accuracy also gives us a chance to improve their design further.

Table 2.1. F_1 scores of high-performance parsers on the Penn treebank.

Parser	WSJ15	WSJ40	WSJALL	G/D	Specific Property
Charniak [6]	-	87.4	86.6	g	PCFG
Collins [3]	-	90.2	89.7	g	Head-driven PCFG
Ratnaparkhi [7]	-	-	86.9	d	Max. Ent. based
Charniak [8]	-	90.1	89.5	d	Max. Ent. inspired
Collins [9]	-	90.3	89.7	g	Discr. reranking
Charniak & Johnson [10]	-	91.0	-	g	Max.Ent.based rerank
Huang [21]	-	-	91.7	g	Forest-based reranking
McClosky [11]	-	-	92.1	g	Rerank & self-training
Zhang *et al.* [22]	-	-	92.6	-	Multi-parser
Finkel *et al.* [12]	89.9	89.0	-	d	Cond. Random Fields

2.1.1. Grammar

The syntax of a language describes the rules and principles that govern any sentence structure. It is defined by a grammar formalism and parsers use this grammar to process sentences. In the literature, there are a number of different grammar formalisms which include context free grammar (CFG) [1, 2], head-driven phrase-structure grammar (HPSG) [3], dependency grammar (DG) [4], lexical functional grammar (LFG) [23], combinatory categorial grammar (CCG) [24] and tree adjoining grammar (TAG) [25]. Most of the research in the parsing literature have focused on constituency, such as CFG, and dependency grammars. In this thesis, our parser uses the CFG formalism.

Context-free grammar, which is also referred to as phrase structured grammar, is used to describe constituency relations. Hence, parsers using CFG are called constituency parsers. With CFG, we can show how phrases can be decomposed, or from

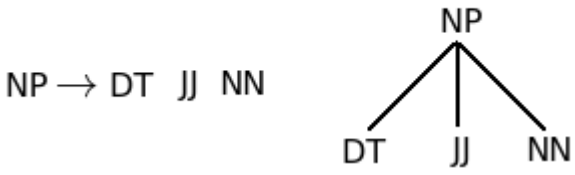

Figure 2.1. Sample CFG rule and its representation in a tree structure.

another stand point, be produced in generative manner. Each CFG rule is of the form $V \rightarrow w$, where V is single nonterminal symbol, and w can be single or a sequence of terminals and non-terminal symbols in any order. For example, NP $\rightarrow$ DT JJ NN describes how noun phrase (NP) is decomposed into or generated from a determiner (DT), an adjective (JJ) and a noun (NN). CFG rules can also be shown in a tree structure as illustrated in Figure 2.1.

Grammars can be either manually constructed or automatically extracted from an annotated corpus. In this thesis, we extracted CFG rules from the training set of the Penn treebank [5], as described in Section 2.5.1.

2.1.2. Grammar Binarization and Markovization

Derived from the early rule-based approaches, most of the parsers employ dynamic programming techniques in order to control the time complexity of the written program. One such technique used by constituency parsers, is called grammar binarization. It makes sure that the time complexity of the program doesn't exceed $\mathcal{O}(n^3)$, while preserving the parsing accuracy. That is why we use grammar binarization throughout this thesis.

Binarization of the grammar rules involves converting those with more than two children constituents into multiple rules, each having two children. Figure 2.2 shows an example context-free grammar rule with three children constituents and its binarized form in which the last two children of the original rule become part of a new rule, headed by the newly constructed constituent $@NP\text{-}DT$. Appending $\text{-}DT$ to the end

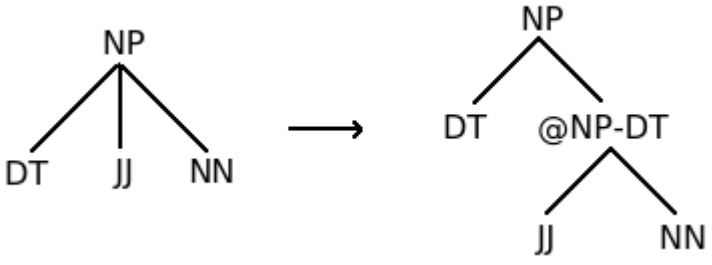

Figure 2.2. Sample binarized parse tree.

tells us that the previous child is a DT. The number of previous children included in the state is referred to as the level of horizontal Markovization. Deciding on that level is a design decision. In our experiments, we apply one level of horizontal markovization.

In the parsing terminology, constituents are called states in the constructed chart. Since $@NP\text{-}DT$ is actively being constructed, such constituents are called active states. Just like appending -DT, states can also be augmented with information of ancestors further up in the tree. This is called vertical markovization, or grandparent annotation if only one level is included. In our experiments, we use one level vertical markovization. The following section further explains which annotations we used in our experiments.

2.1.3. Grammar Annotation

Adding additional information to the grammar rules is called grammar annotation. It is used to specialize the grammar further with increased context information in order to increase the accuracy of the parser. Currently, most high performing parsers are lexicalized parsers which use grammars that are annotated with word information. However, despite their high performance, lexicalized parsers use too many features, which prevents practical good optimization. On the other hand, unlexicalized parsers offer more practical solutions in terms of a computationally more acceptable optimization. However, the context-free assumption of the naive Penn treebank makes unlexicalized models very weak because parse trees do not contain extra information regarding their context. Johnson [26] showed that the performance of an unlexicalized PCFG on the Penn treebank could be improved enormously by simply annotating each

state with its parent category. Such annotation schemes are even considered as partly complimentary to information derivable from lexicalization [8].

Klein and Manning [27] used a large set of simple yet linguistically motivated annotations that increase the accuracy of an unlexicalized parser to a degree of the state-of-the-art lexicalized counterparts. In all our experiments, we use most of these annotation schemes. These include UNARY-INTERNAL, UNARY-DT, UNARY-RB, TAG-PA, SPLIT-AUX, SPLIT-CC, SPLIT-%, TMP-NP, GAPPED-S, POSS-NP, SPLIT-VP, BASE-NP, and RIGHT-REC-NP. Detailed description of these annotations can be found in [27].

2.1.4. Evaluation

Borrowed from the Information Retrieval field, parsers are evaluated with precision, recall, and F_1 score metrics. In order to calculate these metrics, first labelled spans are extracted from the outputted parse trees. Figure 2.3 shows an example parse tree with three labelled spans extracted from it. However, parsing accuracy does not measure how a parser assign part-of-speech (POS) tags to words. Hence, labelled spans do not include spans, like noun tag covering the word 'dog'.

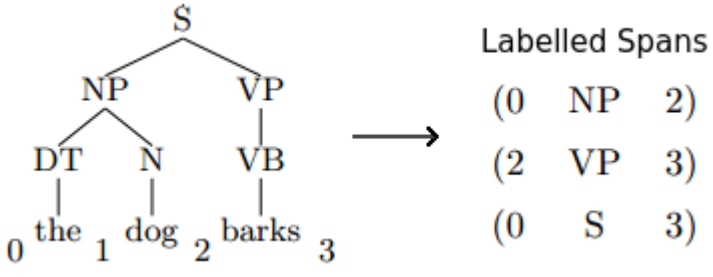

Figure 2.3. Sample tree and its labelled spans.

After extracting the labelled spans from the outputted parse trees, they are compared with actual labelled spans obtained from the golden trees. In this thesis, for development and testing purposes, we use the parse trees from the development and test sections of the Penn treebank as golden trees.

$$\text{Precision} = \frac{\text{\# of correct labelled spans}}{\text{\# of labelled spans in parser output}} \tag{2.1}$$

$$\text{Recall} = \frac{\text{\# of correct labelled spans}}{\text{\# of labelled spans in golden tree}} \tag{2.2}$$

$$F_1 \text{ Score} = \frac{\text{2 x Precision x Recall}}{\text{Precision + Recall}} \tag{2.3}$$

Regarding what each metric means, precision (Equation 2.1) shows what percentage of the outputted parse tree is correct, while recall (Equation 2.2) calculates what percentage of true constituents are actually found by the parser. F_1 score, on the other hand, takes the harmonic mean of these two metrics as shown in Equation 2.3. This way, when we talk about the accuracy of the parser, we use one metric instead of two, unless needed. In this thesis, we also report labelling and identification errors. These two measures are directly related to the precision and recall metrics, respectively. Labelling error occurs when the parser's output is wrong, whereas identification error occurs when the correct constituent is not given by the parser.

In the literature, along with the fundamental metrics described above, it is also customary to measure the percentage of completely correct trees, the average number of brackets crossing with actual correct spans, and the percent of guessed trees that have no crossing brackets with respect to corresponding gold tree. Bracket crossing happens when a span given by the parser crosses a true span of the golden tree. For example, a parser may output the parse tree (A (B C)), while the golden tree is ((A B) C), in which case the span that covers B and C crosses the span that covers A and B

in the golden tree. In order to calculate all these metrics including precision, recall and F_1 score, we use the *evalb* script[1] which is commonly used in the parsing literature.

2.2. n-gram Trees

To be explained in more detail in Section 4.1, an n-gram tree is, in basic terms, a portion of parse tree that covers consecutive n words. To the best of our knowledge, in the literature, n-gram parsing has never been considered as a stand-alone parsing task because n-gram trees have no particular use on their own. However, they have been used as features for statistical models in either lexicalized or unlexicalized forms. For example, McClosky *et al.* [11] used them to train the reranking model of the self-training pipeline.

There are other studies related to our notion of n-gram trees in the literature. One of them is stochastic tree substitution grammars (TSG) used in Data Oriented Parsing (DOP) models in Bod *et al.* [28]. TSG is similar to context-free grammars, with the only difference that a TSG is composed of subtrees of arbitrary length. Substitution operation is used to combine these subtrees and create a partial parse tree and, eventually a full parse tree for a given sentence. Figure 2.4 shows a set of TSG trees. Unlike TSG trees, our n-gram trees always have words at the terminal nodes.

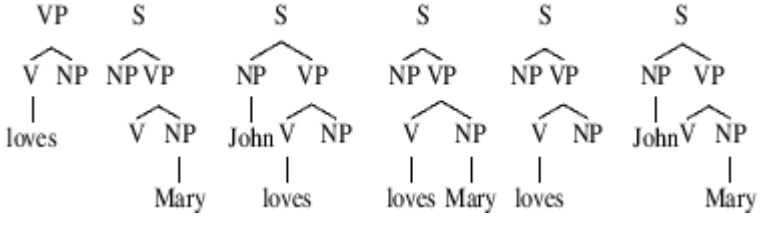

Figure 2.4. Sample tree substitution grammar rules.

Another related concept is the Tree Adjoining Grammar (TAG) and the concept of local trees proposed by Joshi *et al.* [29]. In TAG, each elementary tree is anchored to a lexical item. Figure 2.5 gives three sample TAG trees, with anchored words men,

[1] *evalb* script is available at http://nlp.cs.nyu.edu/evalb/

eat and think, respectively. This elementary tree or local tree contains all arguments of the anchored word and each local tree is independent from the others. As in the case of TSG trees, TAG local trees also differ from our n-gram trees by not having words at all terminal nodes but one.

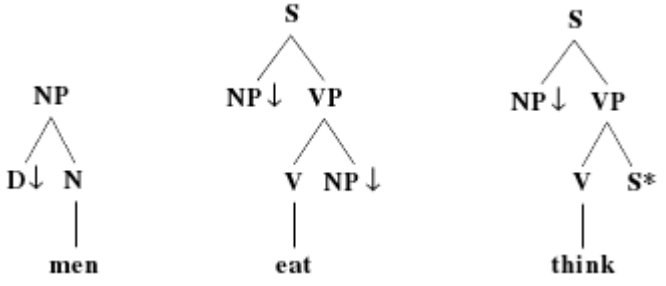

Figure 2.5. Sample lexicalized tree adjoining grammar rules.

2.3. Hierarchical Joint Learning and Parsing

Hierarchical models provide a scheme to do multi-task learning, which is a simultaneous learning approach where we try to learn a problem along with other related problem(s) at the same time, hoping that the other related problems help learn more about the main problem due to the commonality among the tasks. In the NLP field, the concept of domain adaptation is considered related since one model from one domain helps the other model from another domain. There have been studies on domain adaptation by building models of multi-domain learning from Daume III and Marcu [30] as well as Finkel and Manning [31].

More related to the parsing, Finkel and Manning [31] used hierarchical joint learning (HJL) approach to improve the joint model of parsing and named entity recognition. In their hierarchical learning setup, the joint model, which is trained on parse trees augmented with named entity information, is combined with base models of CRF-based parser and semi-CRF-based named entity recognizer. Even though the two base models do not have shared features, each has shared features with the joint model. As the prior of the hierarchical model encourages the learned weights for the different models to be similar to one another, when base model is trained on its own type data set, shared feature parameters of the joint model are influenced by their

correspondents from the base model. After the completion of training, they evaluate the joint model on the OntoNotes corpus [**?**] and measure substantial error reduction on both tasks, up to 6-8 gain on the F_1 score. They also observe that the hierarchical model helps smaller data sets more than large ones.

2.4. Self-Training for Parsing

In the literature on syntactic parsing, almost all of the studies have been based on supervised or semi-supervised methods, with a couple of exceptions of unsupervised approaches, such as Klein and Manning [32]. Even though supervised methods achieve the best results, in the absence of sufficient annotated data, they can be outperformed by semi-supervised methods.

Self-training and co-training are two approaches for semi-supervised learning. Co-training involves two different learners that can see the data from different perspectives so that if one of them is confident about something, that becomes the labelled data for the other learner. Whereas in self-training, small amount of labelled data is used to annotate unlabelled data, which becomes the labelled data for the learner at the next cycle of its training. In this thesis, we study a self-trained constituency parser.

Charniak [6] considers the first self-training approach by first training his parser on the Penn treebank [5] and parsing 30 million words of unparsed Wall Street Journal text. Then he uses both already labelled and newly labelled data for training. However, his self-trained model fails to outperform the original model.

Bacchiani and Roark [33] trained the Roark's parser on trees from the Brown treebank and then self-train and tested on data from Wall Street Journal. While they show some improvement on F_1 score going from 75.7 to 80.5, their parsing results were lower than the state-of-the-art levels.

Steedman *et al.* [34] investigated both self-training and co-training in the 2002 CLSP Summer Workshop at Johns Hopkins University. They considered several dif-

ferent parameter settings. In all cases, the number of sentences parsed per iteration of self-training was 30 sentences, which is relatively small. They performed many iterations of self-training. The largest seed size (amount of labelled training data) they used was 10000 sentences from the Wall Street Journal (WSJ), though many experiments used only 500 or 1000 sentences. They found that under these parameters, self-training did not yield a significant gain unless the baseline results were sufficiently bad, as in the case of [33].

Reichart and Rappoport [35] showed that one can self-train with a generative parser only if the seed size is small. The conditions are similar to [34], but only one iteration of self-training is performed. In this scenario, unknown words (words seen in the unlabelled data but not in training) were a useful predictor of when self-training improves performance.

Table 2.2. Summarization of self-training performances.

	Parser Type	Seed Size	Iterations	Improved?
Charniak [6]	Generative	Large	Single	No
McClosky *et al.* [11]	Gen.+Discr.	Large	Single	Yes
Steedman *et al.* [34]	Generative	Small	Multiple	No
Rappoport [35]	Generative	Small	Single	Yes

In McClosky *et al.* [11], they used a generative parser and a discriminative reranker and trained their system on the Penn treebank and used that trained parser to parse the North American News Text Corpus (NANC), which contains approximately 24 million sentences from various news sources such as the Los Angeles Times, the New York Times etc. After getting the parsed version of the NANC data set, they combined it with the Penn treebank data set and trained their parser on that combined data set for the second time. They observed that by retraining the first stage, they achieved better performance for both models. They obtained absolute 1.1 F_1 score improvement (12% error reduction) over the previous best result on the Penn treebank. Error analysis revealed that most improvement comes from sentences with lengths

between 20 and 40 words. Surprisingly, improvements were also correlated with the number of conjunctions but not with the number of unknown words in the sentence.

2.5. Data

This section introduces the two data sets that we use throughout this thesis. In following chapters, we give more detailed explanation and describe how we use these data sets according to our needs.

2.5.1. Penn Treebank

In the literature, the Penn treebank [5] is a widely used data set for the constituency parsing task. It contains approximately one million words (40000 sentences) of manually annotated sentences from the Wall Street Journal. It consists of 23 sections. In the literature, 23rd section is used for development, whereas 22nd section is for testing purposes. The other 21 sections are used for training. In the evaluation process of this thesis, we follow this split in order to get comparable results w.r.t. other results from the literature.

In addition to using all provided training data set of the Penn treebank, most of the research provides parser evaluations on smaller scale training sets, which are chosen based on the lengths of sentences in words. For example, it is general practice to use sentences with 15 words or less for both training and testing, which we refer to as WSJ15. To make the task a little bit harder, sentences with 40 words or less can also be used, i.e. WSJ40 in that case.

Table 2.3. Number of sentences for each subversion of the Penn treebank.

Portion	Training Set	Dev. Set	Test Set
WSJ15	9753	421	603
WSJ40	36740	1578	2245
WSJALL	39803	1700	2416

Table 2.3 gives the number of training, development and test sentences for each subversion of the Penn Treebank data set. Apart from this typical partitions, for the n-gram parsing task, we also define another subversion called WSJOver15, which contains those sentences having more than 15 words, that is completely exclusive of WSJ15 set. By doing this, we make sure that our n-gram models are trained on a different set, which is especially important when we jointly train n-gram and full parsing models. More detail is provided in Section 4.2.

2.5.2. Reuters Corpus

The Reuters Corpus Volume 1 (RCV1) [36] is an archive of 806791 English language news stories that is available to the research community via NIST, the National Institute of Science and Technology. It contains stories in English produced by Reuters journalists between 20/8/1996 and 19/8/1997. RCV1 is used in research and development of natural language processing, information retrieval or machine learning systems, most specifically for text categorization and clustering.

In this thesis, we use RCV1 for two different purposes. The first one is for learning word clusters based on distributional similarity as described in Section 3.3.3 and the second one is for extracting sample sentences to be used at self-training in Chapter 6.

3. DISCRIMINATIVE CONSTITUENCY PARSING ON FULL SENTENCES

The first step towards the self-trained parser in this thesis is to build our baseline parser, which all the parsers proposed in the upcoming chapters are compared with. This chapter describes how we build our baseline parser based on the work of Finkel *et al.* [12]. It is a discriminative constituency parser as it uses context-free grammar formalism and employs Conditional Random Fields (CRFs) approach as a statistical model to score the possible parse trees and choose the best possible one for the output. In the following sections, we first explain how a discriminative model like CRFs approach is adopted for parsing and then talk about the implementation details. We discuss the results on the Penn treebank at the end of this chapter.

3.1. Parsing with Conditional Random Fields

Proposed by Lafferty *et al.* [37], Conditional Random Fields (CRFs) is a log-linear model for segmenting and labelling sequence data. Unlike generative models, which optimize the joint likelihood of the training samples, CRFs is a discriminative model which directly optimizes the conditional likelihood of an unobserved variable given the observed data. Adopting the same comparison to parsing, constituency parsers with generative models maximize the joint likelihood of the parse tree and the sentence together and they do that by employing probabilistic context free grammar (PCFG) introduced by Charniak [6]. On the other hand, discriminative parsers maximize the conditional likelihood of the parse tree given the sentence, that is P($t|$ s;θ).

$$P(t|s;\theta) = \frac{1}{\mathcal{Z}_s} \prod_{r \in t'} \phi(r|s;\theta) \tag{3.1}$$

Equation 3.1 describes how to calculate the conditional likelihood of a parse tree t given sentence s. In this equation, CRF-based context-free grammar (CRF-CFG) is represented with local clique potentials ϕ(r|s;θ), where r is one-level subtree of a parse

tree t. Unlike conventional grammar rules, r also includes information about start and end positions of words that it spans, as well as the split position for the binary rules. $\mathcal{Z}_{\mathrm{s}}$, on the other hand, is the partition function.

$$\mathcal{Z}_s = \sum_{t \in \tau(s)} \prod_{r \in t'} \phi(r|s;\theta) \tag{3.2}$$

As given in Equation 3.2, the partition function $\mathcal{Z}_{\mathrm{s}}$ is calculated over all possible parse trees $\tau(s)$ for a given sentence and then used to normalize the probabilities in Equation 3.1. In case of generative models, however, probabilities are normalized locally, unlike the way partition function does it globally.

$$\phi(r|s;\theta) = exp \sum_i \theta_i f_i(r,s) \tag{3.3}$$

Local clique potentials are used to score rules in the Inside-Outside algorithm. Their values are not probabilities but non-negative numbers due to their exponential form. They are calculated by taking the exponent of the dot product of the feature vector $f(r,s)$ and parameter vector θ_i. The feature vector $f_i(r,s)$ is simply an indicator function that tells whether feature i is active for given rule r and sentence s.

3.1.1. Objective Function and Parameter Estimates

Given a set of training examples, the goal is to choose parameter values θ_{i} that minimize the conditional log likelihood of those examples, which is the objective function $\mathcal{L}$.

In order to estimate the parameter values, there are various optimization methods in the literature such as generalized iterative scaling or conjugate gradient methods. However, since the inference step of such methods takes time, it is not practical to use them when we need to do inference many times. In such cases, it is best to use stochastic optimization techniques, as they are considered to be efficient, especially when the objective function requires computationally expensive calculations. In our

case, the objective function is the log-likelihood of the training data $\mathcal{D}$ along with an additional L_2 regularization term to prevent over-fitting.

$$\mathcal{L}(\mathcal{D};\theta) = \sum_{(t,s)\in\mathcal{D}} \left(\sum_{r\in t} \langle f(r,s),\theta\rangle - logZ_{s,\theta} \right) - \sum_i \frac{\theta_i^2}{2\sigma^2} \tag{3.4}$$

When the partial derivative of our objective function with respect to model parameters is taken, the resulting gradient is basically the difference between the empirical counts and the model expectations.

$$\frac{\partial\mathcal{L}}{\partial\theta_i} = \sum_{(t,s)\in\mathcal{D}} \left(\sum_{r\in t} f_i(r,s) - E_\theta[f_i|s] \right) - \frac{\theta_i}{\sigma^2} \tag{3.5}$$

For stochastic gradient descent (SGD), we update the parameters based on a batch of samples randomly drawn from the training set. We calculate the partial derivatives for each batch, assuming that summing the partial derivatives of all batches adds up to the partial derivative of the overall data $\mathcal{D}$.

$$E\left[\sum_i^n \hat{\mathcal{L}}\left(\mathcal{D}_b^{(i)};\theta\right) \right] = \mathcal{L}(\mathcal{D};\theta) \tag{3.6}$$

Having said that, we use the partial derivative of the batch to update the model parameters, controlled by the learning rate η. The learning rate function is modelled based on Finkel *et al.* [12]. It is designed such that θ_i is halved after τ passes through the training set. We use $\tau = 5$.

$$\eta_k = \eta_0 \frac{\tau}{\tau + k} \tag{3.7}$$

$$\theta_{k+1} = \theta_k - \eta_k \nabla\mathcal{L}\left(\mathcal{D}_b^{(k)};\theta_k\right) \tag{3.8}$$

In order to calculate the partial derivatives in the inside-outside algorithm, for each rule, we multiply the outside score of the parent, the inside score of the children, and

the score of that rule and then divide that value by $\mathcal{Z}_s$ to get the globally normalized probability of that rule in that particular position. Then, we use the probability of each rule to calculate the expected feature values, which will be inserted into Equation 3.5 to get the partial derivatives and then to update the model parameters.

3.2. Optimizing the Parser

Any kind of stochastic parser considers hundreds of thousands of possibilities and picks the best of them. During this process, optimization of memory and time usage is critical to achieve a parser that runs and outputs in a reasonable time. The following subsections explain two techniques that we use in the implementation of our parser.

3.2.1. Parallelization

In the Conditional Random Fields approach, parameter updates are done by using the differences between empirical and expected values of those parameters. Empirical values are directly calculated by traversing over the golden trees, whereas expected values are calculated during the outside-stage of the inside-outside algorithm. Since we are able to calculate the expected values from each sentence separately, we employ a parallelization approach as follows. We create multiple threads and assign one sentence to each thread. Then, in the main thread, we sum up all the expected values calculated by each thread from their responsible sentence. We establish inter-process communication by memory created by the main thread and shared with the children processes.

For the experiments, we fire as many threads as the size of the batch. For full sentence parsing, for example, we create 15 threads as we use batches of 15 sentences. On the other hand, in n-gram parsing described in the next chapter, we have a bigger batch, hence firing 30 threads. The gain from the parallelization approach differs, based on the number of processes, their speeds as well as the lengths of the longest sentences in the batch.

3.2.2. Chart Caching

When a parser generates a parse forest, it may include hundreds of thousands of ways of forming up a parse tree on a given sentence, hence the name parse forest. Considering all possible cases and calculating the expected values on such a big structure might cause time- and memory-based problems. Apart from memory-based issues, which might be addressed with compact data structures, we are more interested in reducing the time we spend. In most of the cases produced constituents do not contribute to any complete parse tree, becoming dangling constituents not reachable from the *ROOT* constituent sitting at the top of the forest. Therefore, one solution is to cache the parts of the parse forest which lead to a complete parse of the given sentence. This way, when the same sentence is seen in the next iterations, the parser can look up the cache and quickly determine whether the grammar rule it is about to apply will be a part of a complete parse tree at the end.

This type of caching was also done by Finkel *et al.* [12]. However, our implementation is different from theirs. While they saved the charts into files, we keep the cached charts in the memory, which is expected to be a much faster solution. In our experiments, we observe that chart caching contributes to between 20-25% decrease in the run time of the full sentence parser. However, in case of the n-gram parsing task, described in the next chapter, the same approach provides between 5-10% decrease in the run time. This is due to the fact that the size of the forest for n-grams are considerably small and, thus, the number of dangling constituents in the forest is very low.

3.3. Features

As given in Equation 3.3, features are used at scoring all CFG-CFG rules that are generated over a given sentence. Most of the features we use for parsing throughout this thesis are borrowed from the study of Finkel *et al.* [12] along with the actual parser framework. The features are divided into lexicon and grammar features based on the type of grammar rules they are used for. The following subsections explain

these features along with the concepts used in the feature definitions.

3.3.1. Lexicon Features

Lexicon features are used to score the grammar rules that are involved with tagging words. Their definition may include words, part-of-speech (POS) tags, as well as their alternative forms. As we traverse over the golden trees in the training set, at the preterminal level, we use our lexicon feature templates to create lexicon features. Table 3.1 shows the complete list of the lexicon feature templates that we use in this thesis.

Table 3.1. Lexicon feature templates that we used in this thesis.

$\langle tag \rangle$	$\langle tag, dist\text{-}sim(word_{+1}) \rangle$
$\langle base(tag) \rangle$	$\langle base(tag), dist\text{-}sim(word) \rangle$
$\langle tag, word \rangle$	$\langle base(tag), dist\text{-}sim(word_{-1}) \rangle$
$\langle tag, lower\text{-}case(word) \rangle$	$\langle base(tag), dist\text{-}sim(word_{+1}) \rangle$
$\langle base(tag), word \rangle$	$\langle parent(tag), word \rangle$
$\langle base(tag), lower\text{-}case(word) \rangle$	$\langle tag, unk\text{-}class(word) \rangle$
$\langle tag, dist\text{-}sim(word) \rangle$	$\langle base(tag), unk\text{-}class(word) \rangle$
$\langle tag, dist\text{-}sim(word_{-1}) \rangle$	$\langle base(tag), prefix(word), suffix(word) \rangle$

To form a feature instance from these templates, in addition to using the bare part-of-speech tags, we can also use the base and parent annotations of these tags, as our grammar allows. As described in Section 2.1.1, in our grammar all states are annotated with parent information. Hence the base of a state corresponds to the original tag, whereas the parent annotation of the state corresponds to the parent tag. Apart from using the lower-cased form of a word or the previous and next words in the given sentence, we also substitute the word with its corresponding distributional similarity cluster and unknown class, which will be described in the following sections. This enables us to handle out-of-vocabulary (OOV) words better by using their abstract forms.

Table 3.2. Word prefixes that we used in lexicon feature creation.

7-chars	counter-
5-chars	extra-, inter-, intra-, intro-, micro-, multi-, retro-, super-, trans-, ultra-, under-
4-chars	auto-, anti-, fore-, giga-, mega-, mini-, mono-, nano-, over-, para-, poly-, post-, self-, semi-, tele-
3-char	bio-, dis-, mal-, mid-, mis-, neo-, non-, out-, pre-, pro-, sub-, tri-, top-
2-char	ab-, ad-, bi-, by-, co-, de-, en-, ex-, il-, im-, in-, ir-, no-, re-, up-, un-

One particular feature template that we added on top of the ones borrowed from [12] considers the base tag with a prefix-suffix pair extracted from the word which that tag covers. We introduce this feature template to deal with OOV words better. The idea is that the prefix and suffix of a word together help us define the type of that word better. With these features, we model a relation between the base tag and what prefix and suffix together may correspond to it. Table 3.2 and 3.3 lists these prefixes and suffixes that we look for for each word. If a word doesn't contain any of them, then the value for the corresponding affix is set to NA, indicating the absence.

Table 3.3. Word suffixes that we used in lexicon feature creation.

4-chars	-ness, -less, -ious, -able, -ible, -ical, -ship, -sion, -tion, -ance, -ence, -hood, -ling
3-chars	-ion, -ied, -ily, -lly, -sly, -ies, -ise, -ize, -ses, -ous, -ish, -ful, -ate, -ves -ing, -ify, -ity, -ist, -ism, -acy, -dom, -ive, -ant, -ent, -ery, -ess, -est, -ile, -let, -ish, -ful, -ate, -ing, -ify, -ity, -ism, -ist, -acy, -dom, -ive, -ure
2-chars	-en, -ed, -al, -ly, -or, -er, -ty, -ic, -es, -ar, -ce, -cy, -fy, -ry, -th
1-char	-s, -y

3.3.2. Grammar Features

Grammar features are used to score all grammar rules except the ones used for tagging words. They represent non-local properties of the parse structure, whereas lexicon features involve more local properties. Based on their definition, garmmar rules can associate different properties spanning number of words and subtrees. Table 3.4 shows the grammar feature templates employed in this thesis. Note that there are also

two subtypes of features. One that is only applicable to binary rules, and another that is applicable to unaries which span one word.

Table 3.4. Grammar feature templates that we used in this thesis.

ρ
$\langle base(parent(rule_p)), dist\text{-}sim(word_s)\rangle$
$\langle base(parent(rule_p)), dist\text{-}sim(word_e)\rangle$
$base(rule)$
dist. sim. bigrams below rule and base parent tag
dist. sim. trigrams below rule and base parent tag
Binary-specific feature templates :
$\langle base(parent(rule_p)), dist\text{-}sim(word_{s-1}), dist\text{-}sim(w_s)\rangle$
$\langle rule, word_s\rangle$, if right child is a PP
Unaries which span one word :
$\langle rule, word\rangle$
$\langle rule, dist\text{-}sim(word)\rangle$
$\langle base(parent(rule)), dist\text{-}sim(word)\rangle$
$\langle base(parent(rule)), word\rangle$

In these feature templates, $rule$ represents a particular rule along with span/split information, while ρ is the rule itself. And $base(rule)$ means simplified rule with base states and no span/split information. Also, $rule_p$ corresponds to the right hand side, that is the parent of the grammar rule. Since our grammar is binarized we have a split point unless it is a unary rule. Hence w_b and w_e are the first and last words covered by that rule, respectively, whereas w_s is the first word after the split.

3.3.3. Distributional Similarity Clusters

Out-of-vocabulary (OOV) words present a challenge for the parser, not only in terms of accuracy but also because of the fact that the parser has to consider all possible ways of tagging those unknown words. As Finkel *et al.* [12] used in their research, distributional similarity clusters is one way of coping with OOV words. This technique clusters words based on their distributional similarity in a given large corpus of raw text. Outputted clusters are used as abstract syntactic categories in our feature templates.

For our experiments, we used the implementation provided by [38]. We gathered an unlabelled data set of over 280 million words by combining Reuters RCV1 corpus (200M), Penn treebank (1M), and a large set of newswire articles downloaded over the Internet. We used default parameter settings and set the number of clusters to 200, as done in [12]. Even though we were unable to work with the same set of text corpora that they used in [12], we tried to keep the size and type of content comparable to theirs.

3.3.4. Unknown Word Classes

In addition to the distributional similarity clusters, we also make use of the orthogonal shapes of words in order to come up with more abstract form of words. Such forms are called unknown word classes. These classes are acquired by transforming the word character-to-character to more generic form by using the following rules.

- Spelled out versions of Greek letters are replaced with 'g'
- Digits are replaced with 'd'
- Upper-cased letters are replaced with 'X'
- Lower-cased letters are replaced with 'x'
- Punctuation remains
- Instances where the same character (g/d/x/X) is repeated more than three times in a row are truncated to only include the first three characters.

By such transformation, we can better identify words or compound words that are generated based on a specific format, especially when they contain numerical expressions.

3.4. Data

In our experiments, we use the Penn treebank described in Section 2.5.1. However, due to computational and time constraints bounded by this thesis, we use only

sentences that have no more than 15 words, that is the WSJ15 set. Regarding preprocessing, we start with the original version of the Penn treebank and remove unnecessary annotations like $*NONE*$ constituents and functional tags, except -TMP on temporal noun phrases. We extracted the grammar set from the training set of the WSJ15 and then annotated them with extra information as described in Section 2.1.3. We use the development and test sets of the WSJ15 for the evaluation.

3.5. Experiments

This section describes our experimental setup and the analysis of the results. For the evaluation of our baseline, we do detailed analysis of the causes of errors from different angles in order to enable us to make better comparison of the baseline and proposed parsers in the upcoming chapters.

We follow the experimental setup described by Finkel *et al.* [12]. We train our discriminative parser with all lexicon and grammar features described in Section 3.3. Our preliminary experiments show that when the learning factor η is set to 0.1 and the variance σ^2 to 0.1, our parser reaches the highest accuracy on the development set after 20 passes over the training set. At each pass, our parser randomly chooses a batch of sentences with replacement from the training set and updates the parameter values after processing each batch. Tables 3.5 and 3.6 show the results we get with our parser on the development and test sets of the Penn treebank, respectively. These tables also include corresponding results from [12] for comparison.

Table 3.5. Results on the development set of the Penn treebank.

Parser	Precision	Recall	F_1 Score	Exact	Avg CB	No CB
Our Parser	87.49	88.09	87.79	53.33	0.30	83.10
Finkel *et al.* [12]	90.4	89.3	89.9	59.5	0.24	88.3

Shown in Table 3.5, our parser achieves F_1 score of 87.79 on the development set of the WSJ15, which is low compared to 89.9 obtained by [12]. While there is

about absolute 1% difference on the recall measure, the difference of 3% at the precision metric between the two systems contributes to the performance difference. This means that, even though our parser is almost as good as the parser of Finkel *et al.* at labelling constituents correctly, it also outputs many incorrect constituents along with the correct ones, causing the precision drop.

Table 3.6. Results on the test set of the Penn treebank.

Parser	Precision	Recall	F_1 Score	Exact	Avg CB	No CB
Our Parser	86.35	86.35	86.35	50.67	0.36	79.87
Finkel *et al.* [12]	91.1	90.2	90.6	61.3	0.22	87.9

On the test set shown in Table 3.6, however, the difference between the two systems is getting larger. While our parser reaches F_1 score of 86.35, it is 90.6 for [12]. This time, in addition to outputting more incorrect constituents as seen in the development set, our parser also struggles to find the correct constituents, hence the recall difference of 4%.

3.5.1. Accuracy Analysis on Constituents

We analyze further the output of our parser for the development set of the WSJ15. First, we investigate at which type of constituents our parser fails most. The results for most common constituent types in Table 3.7 indicates that noun phrases (NPs) are involved in most of the errors in the output, which is not surprising considering that most of the constituents are noun phrases. 34.7% of wrongly labelled constituents are NPs, while 40.9% of correct constituents that our parser fails to identify are NPs. After NPs, verb phrases (VPs) contribute to most of the labelling and identification errors. Apart from these two, F_1 score of adjective phrases (ADJPs) points out the most problematic constituent for the baseline parser, despite their relatively small effect to the overall results.

Table 3.7. Performance on the most common constituents in the WSJ15 dev. set.

Constituent	Their % in Labelling Errors	Their % in Identification Errors	F_1 Score
NP	34.7	40.9	88.77
VP	20.4	14.9	89.81
PP	9.6	9.4	88.79
S	12.7	10.7	90.91
SBAR	3.5	3.2	80.56
ADVP	8.2	5.5	79.42
ADJP	6.3	7.9	59.31
QP	0.7	1.0	94.21

3.5.2. Length- and Height-based Accuracy Analysis of Constituents

As we jointly train the baseline full sentence parser with the n-gram parser in the upcoming chapters, we decide to analyze how accurately the baseline parser performs on constituents with different lengths and at different heights in the parse tree. Hence our second analysis involves measuring the identification accuracy of constituents based on their length and then their located height.

As we use the WSJ15, lengths of the constituents can range from one to 15. We divide the constituents into four groups: constituents that span only one word, except the preterminals on top of the words; short constituents of length two to four; mid-sized constituents of length five to nine; and long constituents of length 10-15. Table 3.8 lists the identification accuracy of the golden constituents for each constituent type. Long constituents, which correspond to 12.6% of all constituents, are identified with the highest accuracy by the baseline parser. In fact, from a separate analysis, we also measured that the baseline parser correctly identifies 96.4% of the top constituents in the given parse trees. On the other hand, while almost half of the golden constituents are short, 89.5% of them are identified correctly. Following these two, mid-size and one-word constituents come with around the same accuracy level of 85%. In case of the constituents that cover only one word, even though we exclude preterminals, the

low accuracy indicates how tough it is to identify them.

Table 3.8. Performance on the constituents by length in the WSJ15 dev. set.

Constituent Length Range	% of Constituents	Identification Accuracy (%)
1 (One-word)	19.13	85.01
2-4 (Short)	44.37	89.54
5-9 (Mid-sized)	23.91	85.17
10-15 (Long)	12.59	93.19

Height-based accuracy analysis on the constituents and how we divide them again into four groups is shown in Table 3.9. Results indicate that the accuracy of identifying golden constituents drops at the shallows of the parse trees compared to the constituents above the preterminals and in mid-range in the parse tree. Regarding the constituents located above the height of 10, they include only a smal fraction of the constituents and being able to identify all of them doesn't mean much for our current evaluation.

Table 3.9. Performance on the constituents by height in the WSJ15 dev. set.

Constituent Height Range	% of Constituents	Identification Accuracy (%)
2 (Above Preterminals)	42.12	88.14
3-4 (Shallows)	30.15	86.76
5-9 (Mid-range)	26.28	88.86
10-15 (Highs)	1.45	100.0

When we jointly train the baseline full sentence parser with the n-gram models, since n-gram trees are relatively shallower trees, it would be interesting to examine their effect on these statistics of the baseline parser.

3.6. Discussion

Evaluation of the results obtained using the development and test sets of the WSJ15 set shows a considerable difference between our baseline parser and the one we adapted from Finkel *et al.* [12]. Even though we follow the design specifications of their parser, we were unable to match their accuracy. The reason might be due to small implementation details that are not mentioned in their publication, which might cause some part of that difference. Moreover, the data set we use to calculate distributional similarity clusters of the words is also different. Considering that distributional similarity cluster based features lead to 6-7 point increase in the F_1 score, having different training sets for this task might also play a sizeable part at the end.

Our further analysis also reveals that most of the errors our baseline parser makes originate from noun and verb phrases. Adjective phrases are especially problematic. One possible reason of this is the fact that adjectives can be placed under both noun and adjective phrases, which might prevent the statistical model to do better on them. Our length- and height-based analysis of how accurately our baseline parser identifies constituents reveals that most errors happen with the mid-sized constituents and the ones located at the shallows of the parse tree.

4. N-GRAM PARSING

In the parsing literature, parsers are designed and trained to process sentences either completely as conventional parsers do or partially [39, 40]. n-gram parsing, however, is essentially no different than conventional parsing, yet, the input is defined as n consecutive words in the order that they occur in a natural language sentence. Even though there may be no point in building an n-gram parser, in this thesis, we treat n-gram parsing as an accompanying task for full sentence parsing within the setup of the Hierarchical Joint Learning (HJL) described in the next chapter. This chapter discusses n-gram parsing as a stand-alone parsing task and analyze the data properties and experimental results in order to better understand this task and compare, to some degree, with full sentence parsing. Throughout the thesis, we consider 3-, 4-, 5-, 6-, 7-, 8- and 9-gram parsing in order to examine the characteristics of n-gram parsing and their interaction with the full parsing model at different sizes.

4.1. n-gram Trees

An n-gram tree is part of a parse tree that spans n consecutive words in a sentence. As discussed in Section 2.2, different interpretations of n-gram trees are used in a number of studies in the literature. This section defines n-gram trees used in the context of this thesis.

Figure 4.1 shows all 4-gram trees extracted from the complete parse tree. In order to fit lengthwise, cutting the left or right hand-side of some constituents might be required, while extracting the subtree covering the consecutive words. That cut may sometimes remove the head of the constituent. The requirement during this extraction is to make sure that every syntactic constituent in the extracted n-gram tree keeps its head in the process. We apply this constraint in order to make sure that every n-gram tree is generatively accurate, following the head-driven constituency production process of Michael Collins [3].

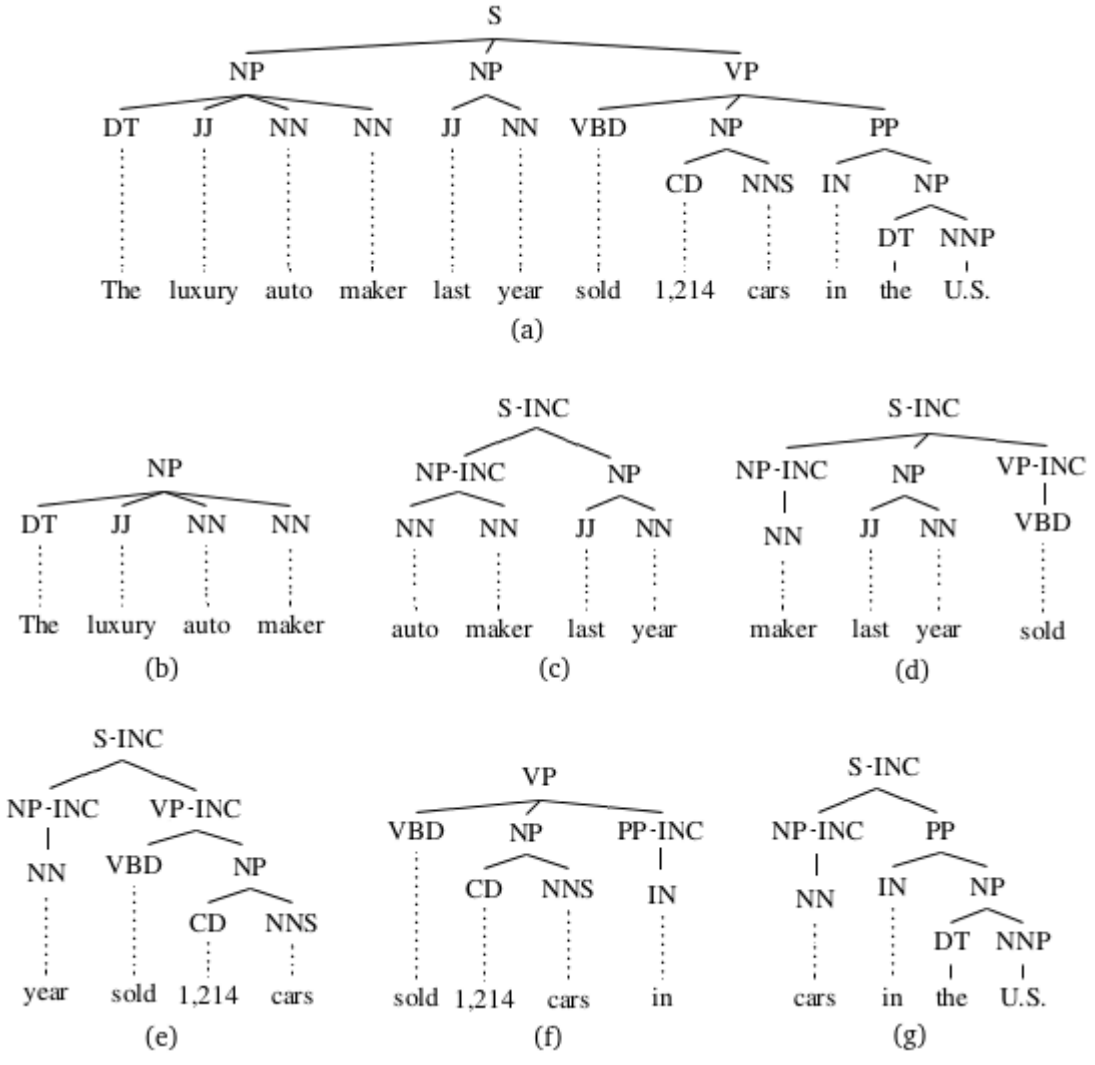

Figure 4.1. All generatively accurate 4-gram trees extracted from the complete parse tree.

Figure 4.2 shows the pseudo-code used to extract n-gram trees from a complete parse tree. It starts traversing the sentence from the first word and tries to preserve the subtree covering n consecutive words from the starting point. While doing that, it may trim the tree from the sides. After trimming, it checks whether each constituent in the extracted subtree has a head child. If not, it aborts the process and continues with the next n consecutive words. If all heads are preserved, it marks the trimmed constituents with -INC functional tag and filters out any incomplete chain of unary rules that can be reached from the $ROOT$. In this process, we also keep the parent information of the top constituent as an additional context information.

Require: n, width of the n-gram trees
Require: *tree*, parse tree of a sentence
  $len \leftarrow$ length of given sentence
  $i \leftarrow 0$
  while $i < len - n$ **do**
    $subtree \leftarrow$ get subtree that covers $[i, i+n]$ span
    $trimmed \leftarrow$ trim $subtree$'s constituents outside the $[i, i+n]$ span, if any
    if $trimmed$ has any constituent with no head child **then**
      i++ and continue
    end if
    $markedtree \leftarrow$ mark all trimmed constituents as incomplete
    $filtered \leftarrow$ filter out incomplete unary rule chain from the ROOT, if any
    save $filtered$ tree as n-gram tree
    i++
  end while

Figure 4.2. Algorithm to extract and store n-gram trees from a parse tree.

4.2. Data

As described in Section 2.5.1, the Penn treebank [5] is a widely used data set for research in constituency parsing. To perform both n-gram tree and full sentence parsing with the Penn treebank, we divide its training set into two parts; sentences having no more than 15 words (WSJ15) and the remaining sentences (WSJOver15).

The n-gram extraction algorithm was run on the training set of the WSJOver15. Table 4.1 shows the number of extracted eligible n-gram trees. As discussed in Section 4.1, not all n-gram trees are generatively accurate. Considering that we have 9753 training sentences for full parsing in WSJ15, the number of training instances for the n-gram parsing task is considerably higher. Nevertheless, due to computational constraints, it is impractical to use all extracted n-gram trees at the same time. Instead, we manage to use at most 20000 training instances for each model. Having hundreds

Table 4.1. Number of n-gram trees in each training set.

Model	Data Set	# of Trees (Avg. Height)
3-gram	WSJOver15	384699 (3.37)
4-gram	WSJOver15	318819 (4.05)
5-gram	WSJOver15	267155 (4.69)
6-gram	WSJOver15	227505 (5.28)
7-gram	WSJOver15	195229 (5.83)
8-gram	WSJOver15	168075 (6.35)
9-gram	WSJOver15	145040 (6.84)
Full	WSJ15	9753 (6.26)

of thousands of training instances allows us to not only randomly create the training set but also create multiple training sets for better evaluation. Hence, we randomly create five different training sets for each experiment that we make. When it comes to evaluation, we run the same experiment five times and get the average of the results, which enables us to make the evaluations more accurate.

Table 4.1 also shows the average height of n-gram trees for each training set. The calculation of average height is done by using the maximum number of edges from the *ROOT* to a preterminal node in the parse tree. As expected, height increases as n increases. Note that average height of full sentences is lower than the height of even 8-gram trees. This is due to the fact that those n-gram trees are extracted from WSJOver15 which contains relatively bigger parse trees that WSJ15 has.

We extract n-gram trees for training purposes from the WSJOver15. However, in case of development and testing purposes of n-gram parsing, we use the development and test sets of the WSJ15. We do this in order to make the results more comparable to the results from the baseline. Table 4.2 gives the number of n-gram trees extracted from the development and test sets of the WSJ15.

Another analysis viewpoint is to look at the distribution of constituent types in

Table 4.2. Number of n-gram trees in each development and test set.

Model	Data set	# of Dev. Trees (Avg. Height)	# of Test Trees (Avg. Height)
3-gram	WSJ15	1742 (3.33)	2495 (4.36)
4-gram	WSJ15	1341 (4.02)	1916 (5.06)
5-gram	WSJ15	1050 (4.65)	1506 (5.72)
6-gram	WSJ15	807 (5.23)	1158 (6.34)
7-gram	WSJ15	635 (5.75)	891 (6.89)
8-gram	WSJ15	486 (6.17)	667 (7.37)
9-gram	WSJ15	349 (6.58)	491 (7.84)
Full	WSJ15	421 (6.06)	603 (7.00)

each data set. Since n-gram trees are relatively smaller compared to the full parse trees and contain mostly local subtrees, it is worth examining which constituents are more likely to be seen in the n-gram data sets.

Table 4.3. Percentage of most common constituents in each training set.

Model	NP	VP	PP	ADJP	ADVP	S	SBAR	QP
3-gram	21.20	10.82	6.25	1.04	1.03	4.68	1.43	0.96
4-gram	20.69	10.87	6.44	1.05	1.07	4.93	1.64	0.85
5-gram	20.39	10.87	6.45	1.06	1.10	5.11	1.75	0.80
6-gram	20.11	10.81	6.49	1.04	1.10	5.21	1.84	0.79
7-gram	19.86	10.78	6.49	1.02	1.11	5.29	1.92	0.78
8-gram	19.68	10.74	6.50	1.01	1.11	5.35	1.99	0.77
9-gram	19.54	10.68	6.50	1.00	1.10	5.39	2.04	0.76
Full (WSJ15)	18.74	9.51	4.21	1.05	1.69	6.91	1.00	0.60

In Table 4.3, the percentages of the most common non-terminal constituent types in each data set are shown. According to these numbers compared to the full sentence data set, the n-gram tree data set contains more noun (NP), verb (VP) and prepositional (PP) phrases, while having less adverb (ADVP) and sentence (S) phrases in terms of percentage. From another standpoint, the percentages of the top constituents

like S and SQ are lower in the n-gram data sets, which is reasonable considering the fact that the n-gram tree extraction process disfavors the top spanning constituents, while favoring short subtrees. Nevertheless, the percentage of the SBAR constituent doubles in n-gram tree sets. We believe this can be explained with their structural resemblance to the prepositional phrases which we see more in n-gram trees.

The same analysis is also performed on the development sets for each model. The numbers indicate different characteristics from the ones we get from the training set because while we extract training n-grams from WSJOver15 set, we use WSJ15 for the extraction of development n-gram trees. Note that our n-gram tree extraction algorithm preserves smaller tree structures and discards large ones. This explains the reason why the percentages of long-range constituents like VP and S from the WSJOver15 set are lower, while the same constituents are relatively smaller in WSJ15 set.

Table 4.4. Percentage of the most common constituents in each development set.

Model	NP	VP	PP	ADJP	ADVP	S	SBAR	QP
3-gram	18.80	12.94	4.69	1.08	1.55	6.62	0.88	1.24
4-gram	18.16	12.98	4.99	1.07	1.65	7.11	1.10	1.00
5-gram	18.12	12.76	5.16	1.06	1.72	7.30	1.19	0.83
6-gram	18.24	12.51	5.20	1.07	1.61	7.27	1.20	0.89
7-gram	18.16	12.07	5.42	1.01	1.51	7.26	1.36	1.00
8-gram	18.07	11.84	5.52	1.00	1.35	7.21	1.41	1.09
9-gram	18.10	11.36	5.73	0.97	1.17	7.18	1.47	1.23
Full (WSJ15)	17.74	8.95	4.44	0.95	1.59	6.68	0.90	0.75

4.3. Features

The n-gram parsing task is essentially the same as the full sentence parsing task. Hence the feature templates described in Section 3.3 are also applicable for this task. Having the same set of features with full parsing is especially important considering the fact that the Hierarchical Joint Learning (HJL) approach described in Chapter 5 de-

pends on the existence of shared features between the tasks. Due to that commonality is how multiple tasks can share their experience and get help from each other.

n-gram and full sentence parsing are identical and use the same set of feature templates. However, in order to better help each other, it is best to have some different features for each task and enable to tasks to look at the problem from different viewpoints and support each other at the points where one of them is not good at. This means that the n-gram parsing task may require different features, specific to the n-gram parsing domain. The first such differentiation is the $-INC$ functional tag, which is used to mark incomplete constituents generated by the n-gram tree extraction process explained in Section 4.1. Such a functional tag creates a new grammar rule, which eventually leads to new features generated by the very same feature templates. So far we haven't added any other differentiating features for the n-gram parsing task yet.

As a future work, we would also like to consider not applying some feature templates for n-gram parsing. The motivation behind this is that some feature templates may be unnecessarily complex for n-gram parsing such that they cause overtraining as the size of the training set increases. Note that this is yet another way of differentiating n-gram parsing from full parsing.

For each n-gram model, we consider multiple training sets by increasing the number of training instances. Table 4.5 lists the average number of features for each training set. Note that we have five different versions of each set so that we get the results with each and take their averages in order to do better evaluation. Hence, in Table 4.5 we report the average number of features for each training set.

4.4. Experiments

For our n-gram parsing experiments, we use the same experimental setup of the baseline parser, but with more optimized parameter settings for the n-gram parsing domain. As described earlier, we work on seven different n-grams with increasing

Table 4.5. Average number of features in each model.

Model	1000	2000	5000	10000	20000
3-gram	28571	45433	81514	124534	185473
4-gram	39256	63201	115609	178022	269570
5-gram	50760	82086	150485	232988	354689
6-gram	61726	100280	185234	288009	440212
7-gram	72651	119345	220496	344598	527970
8-gram	83941	138008	256880	401733	613574
9-gram	95996	156698	293898	458137	695144

sizes from three to nine. By taking into account the future experiments to be done in Chapters 5 and 6, we also consider multiple training sets by increasing their sizes from 1000 to 20000 instances. Moreover, we create five versions of each training set by randomly picking them from hundreds of thousands of eligible n-gram trees extracted from the WSJOver15 set. Hence, all experimental results given in this section are calculated by taking the average of five runs for each experiment.

We use the same set of lexicon and grammar features of the baseline parser described in Section 4.3. From preliminary experiments, we decide to set the learning factor η to 0.05 and variance σ^2 to 0.1. However, unlike doing 20 iterations as in the case of full sentence parsing, we observe that 10 iterations are enough for n-gram parsers to reach their best performances. For the stochastic gradient descent approach, we choose a batch size of 30, instead of 15, since the training instances are relatively smaller compared to the ones in the full parsing task.

Table 4.6 shows the average F_1 scores obtained by all seven n-gram parsers on the n-gram development and test sets described in Section 4.2. Since there hasn't been any other research done on n-gram parsing, we are unable to compare our results. Due to the time constraints, we couldn't run the experiments on bigger training sets. Nevertheless, from the trend of increasing accuracy, it is highly likely that these are not the highest scores we can get for each n-gram model. Nevertheless, these results

Table 4.6. F_1 scores of the n-gram models on each development and test set.

	Results for Dev. Set of the WSJ15					Results for Test Set of the WSJ15				
Model	1K	2K	5K	10K	20K	1K	2K	5K	10K	20K
3-gram	71.95	76.01	80.72	83.86	86.49	72.68	77.13	80.98	84.78	87.14
4-gram	69.71	74.71	80.56	83.69	85.72	71.48	76.28	81.94	84.84	87.38
5-gram	69.51	75.68	80.48	83.61	86.60	71.86	76.51	81.81	84.72	87.04
6-gram	69.03	76.35	80.82	84.48	86.54	71.58	76.47	81.44	85.05	86.95
7-gram	70.96	75.99	81.91	84.24	87.24	71.50	76.73	81.87	84.73	86.57
8-gram	71.31	77.49	81.39	84.26	86.95	72.21	76.11	80.53	84.53	86.34
9-gram	72.37	76.67	82.49	84.08	87.01	71.70	76.76	81.26	84.07	86.44

show us a couple of interesting points regarding n-gram parsing. The first thing to notice is that when using very small training sets like the size of 1000, n-gram parsers perform very poorly. Unlike the full sentence parser, the n-gram parsers require a lot of training data. Another thing to notice is that the larger and more n-gram trees we use, the better results we get on the development set, which is reasonable considering the fact that larger n-gram trees exhibit less ambiguity and they produce more features per tree.

Table 4.7. Performance of the n-gram parsers (trained with 20000 instances) on the development set.

Model	Precision	Recall	F_1 Score	Exact	Avg CB	No CB	TagAcc
3-gram	86.37	86.60	86.49	73.13	0.02	98.15	86.80
4-gram	85.55	85.89	85.72	66.50	0.08	94.86	87.88
5-gram	86.91	86.29	86.60	61.68	0.10	93.02	89.95
6-gram	86.68	86.41	86.54	54.76	0.16	89.33	90.67
7-gram	87.49	87.00	87.24	51.29	0.21	86.52	92.17
8-gram	87.11	86.81	86.95	49.08	0.28	82.96	92.59
9-gram	87.50	86.53	87.01	46.12	0.35	79.45	92.97

More details for each n-gram parser trained on 20000 instances are shown in

Table 4.7. While the F_1 scores are relatively close to each other, the percentages of the exact parse trees found by the n-gram parsers drop rapidly as n increases. This trend is also the same for the average number of crossing bracket cases and the percentage of outputted trees with no crossing brackets. Another thing to notice from Table 4.7 is that precision increases as n increases, while recall stays relatively at the same level for all models. This means that increasing the size of the n-gram tree helps more about preventing mistakes than producing correct trees. The last column of Table 4.7 also shows that as n increases, tagging accuracy increases as well. This might be considered as yet another sign of increasing context information and its contribution for more accurate parsing.

4.4.1. Accuracy Analysis of Constituents

Table 4.8. F_1 scores on the most common constituents for each n-gram model from the development set.

Model	NP	VP	PP	S	SBAR	ADVP	ADJP	QP
3-gram	89.26	89.86	92.04	84.69	56.31	73.19	46.50	75.27
4-gram	88.02	89.79	90.72	83.52	61.43	73.23	48.23	75.77
5-gram	88.35	90.42	89.81	85.77	72.39	76.31	52.22	72.96
6-gram	87.65	91.04	89.03	86.11	71.49	75.16	52.00	74.34
7-gram	87.80	91.73	88.40	87.39	77.91	76.87	50.09	84.45
8-gram	87.22	92.29	87.81	87.18	76.43	77.84	49.05	86.73
9-gram	87.79	91.76	87.33	87.46	76.37	72.21	53.66	86.43
Full	88.77	89.81	88.79	90.91	80.56	79.42	59.31	94.21

As in the case of the evaluation of the full sentence parser in Section 3.5, we also analyze how n-gram parsers perform at the constituent level by calculating the F_1 scores for each constituent type. Table 4.8 shows average F_1 scores of the most common constituents for each n-gram model trained with 20000 instances. Comparing the scores with the corresponding ones obtained with the full sentence parser reveals a couple of points. In case of the noun (NP) and verb (VP) phrases, it is interesting to note that while NPs are getting harder to process, it is clearly the opposite case

of the VPs. In fact, like NPs, accuracy on the prepositional phrases (PPs) is also dropping, and like VPs, performance on the sentence phrases (S) is getting better, too. A reasonable explanation for this is that as the size of the n-grams increases, it becomes easier to handle long constituents like VPs and Ss because the parser sees more of them, whereas while n increases, it also introduces longer NPs and PPs which, in this case, increases the complexity of processing such phrases. Regarding the other constituents, their accuracy increases as well. This is again related to the increase in the number of training instances for such phrases. Comparing the numbers more closely, it is evident that n-gram parsers are having a difficult time processing long constituents like S and SBAR. Especially, these two constituents might be the main cause of why 4-gram model is the worst. Another interesting point is the accuracy on QPs, which is quite low compared to the performance of the full sentence parser on them.

Table 4.9. Performance of the n-gram models on the most common constituents in the test set.

Mdl	NP	VP	PP	S	SBAR	ADVP	ADJP	QP
3-g	30.4/31.5	25.2/15.9	6.4/5.2	15.2/15.9	5.5/6.2	4.9/7.3	5.3/10.0	5.3/4.4
4-g	31.0/31.8	24.4/14.8	7.0/6.5	16.3/17.4	6.6/6.1	5.0/7.3	4.8/9.0	3.5/3.6
5-g	34.5/31.0	22.6/16.0	9.6/7.0	12.9/18.2	4.6/5.4	5.0/7.0	5.1/8.5	4.2/3.1
6-g	36.1/32.7	18.6/15.8	10.3/7.4	13.3/16.8	4.7/5.6	5.7/6.2	5.0/8.8	4.6/2.9
7-g	38.9/34.0	19.7/13.5	10.5/10.1	12.5/16.7	3.7/5.7	5.2/5.6	4.1/9.2	3.7/1.7
8-g	39.5/34.9	16.5/13.1	11.3/10.3	12.4/16.6	4.6/5.7	5.5/4.4	5.5/9.1	3.3/1.6
9-g	35.7/36.3	18.2/12.5	12.3/11.7	12.4/15.6	4.5/5.8	6.4/3.9	4.9/8.3	3.7/2.0
Full	34.7/40.9	20.4/14.9	9.6/9.4	12.7/10.7	3.5/3.2	8.2/5.5	6.3/7.9	0.7/1.0

In addition to calculating F_1 scores, we also calculate what percentage each constituent contributes to the labelling and identification errors, as we also did for the full sentence parser in Section 3.5. Table 4.9 shows these two percentages respectively, separated by a slash. For example, in case of the performance of the 3-gram parser on NPs, the numbers from the table tells that 30.4% of the wrongly labelled constituents are NPs, while they contribute to 31.5% of the unidentified golden constituents. These

percentages indicate the same trends as F_1 scores reveal, however, it is interesting to notice how NPs and PPs are getting more and more problematic as the n-gram trees get bigger, surpassing the full sentence parser levels with the 5-gram parser.

Considering all these statistics on constituents, we may expect that when the full sentence parser is jointly trained with the n-gram model, it might benefit from the performance of n-gram parsers on verb phrases, especially as we use bigger n-gram trees. On the other hand, n-gram parsers trained on smaller n-gram trees might help the full parser with the NPs and PPs. And considering their high percentage of contribution to the errors, smaller n-grams are expected to give the best boost to the full sentence parser in the hierarchical joint learning setups described in the upcoming chapters.

4.4.2. Length- and Height-based Accuracy Analysis of Constituents

Like it is done for the evaluation of the full sentence parser, we also analyze how each n-gram parser performs based on the length and the height of the constituent in the parse tree. As described in Section 3.5, we group constituents into four groups for both analysis. The results obtained by each n-gram parser are given in Table 4.10 and 4.11, respectively.

Table 4.10. Length-based accuracy on constituents for each n-gram model.

Model	L=1	L=2-4	L=5-9
3-gram	82.0	89.1	-
4-gram	82.1	87.6	-
5-gram	82.2	86.8	89.9
6-gram	82.6	85.8	90.4
7-gram	83.5	86.9	88.5
8-gram	82.9	86.6	89.1
9-gram	83.3	86.2	88.5
Full	85.0	89.5	85.2

Length-based analysis of the accuracy on the constituents reveals that the n-gram parsers are unexpectedly not very good at constituents that cover just one word. One reason might be the difficulty of the task, especially in case of absent surrounding context information. It is also noted that even the full sentence parser is having a difficulty with them. On the other hand, n-gram models are also not very good with short constituents covering two to four words and the performance decreases as n increases. We can't think of any particular reason for this and, thus, it requires further analysis. In case of the mid-size constituents covering five to nine words, n-gram parsers perform considerably better than the full sentence parser.

Table 4.11. Height-based accuracy on constituents for each n-gram model.

Model	H=2	H=3,4	H=5-9	H=10-15
3-gram	86.9	86.4	84.2	-
4-gram	86.2	85.3	87.6	-
5-gram	86.4	84.7	89.2	-
6-gram	86.7	83.3	90.2	-
7-gram	87.9	83.6	88.7	100.0
8-gram	87.4	82.4	90.7	100.0
9-gram	87.6	82.1	89.3	96.55
Full	88.1	86.8	88.9	100.0

Considering how accurately each n-gram parser identifies constituents based on their height in the golden tree might not only give us clue about the characteristics of the n-gram parsing task, but also allows us to see how it might affect the full parser when they are jointly trained in the upcoming chapters. Table 4.11 indicates that with smaller n-grams, the parser performs poorly above the preterminal (H=2). But it gets better as n-gram trees get bigger. The same is true at the mid-range (H=5-9). However, it is the opposite case at the shallows (H=3,4). The bigger n-gram trees we use, the worse it performs on constituents located at the height of three and four in the parse tree. One possible explanation might be that the complexity of such constituents at that height is increasing as n increases. At the highs (H=10-15), n-gram parsers perform excellently as the full sentence parser does.

4.4.3. Analysis of Incomplete Constituents

Described in Section 4.3, n-gram models differentiate from the full sentence parsing model with the concept of the incomplete constituents, which are introduced by the n-gram tree extraction algorithm given in Section 4.1. As we cut the constituents from the sides, it creates grammatically incomplete constituents, even though we make sure that the head constituent is intact, keeping the constituent generatively accurate. In order to measure their level of contribution to the number of errors, we calculate the accuracy for incomplete constituents. Table 4.12 shows these statistics from the development set of each n-gram parser. It first gives the percentages of incomplete constituents with respect to all constituents in each set. Then, it lists the accuracies, in other words recalls of each n-gram parser on the incomplete constituents. The last column shows what percentage of unidentified constituents in overall is actually the incomplete constituents.

Table 4.12. Accuracy on the incomplete constituents in each n-gram models

Model	% of Incomplete Constituents in Golden Trees	Incomplete Constituent Accuracy	% of Unidentified Incomplete Const. w.r.t. All Unidentifieds
3-gram	22.0	86.65	26.2
4-gram	17.4	85.78	21.1
5-gram	14.5	84.20	16.7
6-gram	12.3	83.41	15.2
7-gram	10.8	83.32	13.9
8-gram	9.5	83.92	11.5
9-gram	8.7	84.94	10.4

From the results in Table 4.12, we observe that as size of the n-gram tree is getting bigger till 7-gram, accuracy on the incomplete constituents drops and then it starts to improve. Their contribution to the number of errors drops as n increases, but that is more attributed to the fact that their percentage with respect to all constituents drops as n increases. Despite its high performance, more than quarter of the errors

done by the 3-gram parser involves incomplete constituents.

4.5. Discussion

The task of n-gram parsing is in essence the same as the task of full sentence parsing. The major difference lies in the different feature templates and the characteristics of the data set. Even though n-grams are shorter than full sentences and we have many more training instances of them, this chapter reveals that parsing n-grams is not an easier job than parsing full sentences. The first reason that comes to mind is the absence of surrounding context. Especially for smaller n-gram trees, this increases the level of ambiguity. Despite such inherent characteristics, another source of poor performance for the n-gram parsers is the relatively low percentage of certain types of constituents like S, SBAR, and QP in the training set. That is actually why we keep incomplete constituents. If we discarded any n-gram that contains such constituents, the percentage of such phrases would drop considerably and that hurts the accuracy levels, not to mention the negative effect of this to the level of help that n-gram parsers provide to the full sentence parser in the course of joint learning described in the upcoming chapters.

Our further analysis also indicates that n-gram parsers are strangely not very good at handling short constituents as well as the ones seen towards the bottom of the parse tree. It might have to do with the increased complexity of these specific tasks in the absence of contextual information. It would be interesting to examine these findings further and determine the reasons before introducing new features specific to these types of deficiencies.

Another thing to notice is the performances of the n-gram parsers on incomplete constituents, which might also be considered as another source of the ambiguity for the n-gram parsing task. Their high contribution to the identification errors for 3- and 4-gram parsers exemplifies the increasing ambiguity level as n-gram trees get smaller.

To sum up, the results show that the accuracy levels of the n-gram parsers is close

to the level we reach with the full sentence parser. However they require a larger set of training data compared to the full sentence parser. Even though this does not seem like a problem due to the abundance of the extracted n-gram trees, using large numbers of training instances increases the running time. Despite the mentioned deficiencies, the obtained performance is promising for the applications that can make use of the n-gram parsers.

5. HIERARCHICAL JOINT LEARNING OF N-GRAM AND FULL PARSING TASKS

In Chapter 3, we build a discriminative constituency parser as our baseline parser to parse full sentences. In Chapter 4, we use the same setup with additional feature templates to parse n-grams. In this chapter, we combine these two different types of models with the Hierarchical Joint Learning (HJL) approach. As introduced by Finkel *et al.* [31], HJL provides a multi-task training environment in which related tasks help each other learn more their task due to the commonality among the tasks. Here, our goal is to examine whether we can improve our baseline parser with the help of n-gram parsers using the HJL approach.

5.1. Hierarchical Joint Learning

HJL makes use of multiple related base models along with their corresponding training data sets. The only requirement for HJL to work is that the base models need to have the common features with some other base models, while they can still have private features of their own.

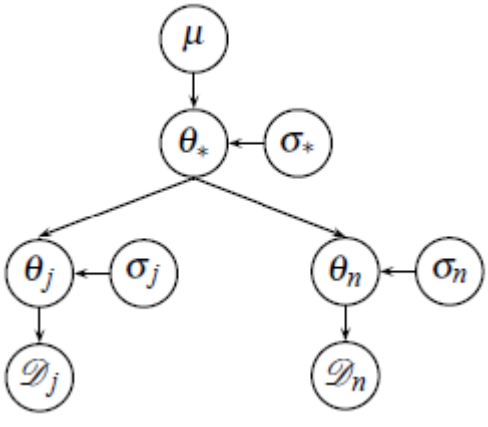

Figure 5.1. A graphical representation of the hierarchical model.

Shown in Figure 5.1 is a graphical representation of a hierarchical joint model. Nodes at the bottom corresponds to each base model with their own set of parameters

θ and a variance σ^2. On the other hand, the top node connected to the bases represents the top-level model with again its own set of parameters, a variance and a mean. This connection corresponds to the shared features between the base models through the top model and it encourages the learned weights for the different models to be similar to one another.

5.1.1. Formal Model

As described in [31], we have a set M of base models with corresponding log-likelihood functions $L_m(D_m;\theta_m)$, where D_m is the model-specific training data and θ_m is the model-specific feature vector for the mth model. These likelihood functions do not include priors over the θs, instead these feature vectors are all drawn from a hierarchical prior which connects these base models. The parameters $\theta_\star$ for this prior have the same dimensionality as the model-specific parameters θ_m and are drawn from another, top-level prior. We use zero-mean Gaussian for this top-level prior as in [12]. Note that since each specific data set can have its own set of features, those features that are specific to one model have no affect on other model's likelihood function because they supposedly cannot be seen in the other model's training set.

Having defined the hierarchical model, the log-likelihood of this model is given as follows.

$$\mathcal{L}_{hier-joint}(\mathcal{D};\theta) = \sum_{m\epsilon\mathcal{M}} \left(\mathcal{L}_m(\mathcal{D}_m;\theta_m) - \sum_i \frac{(\theta_{m,i} - \theta_{*,i})^2}{2\sigma_m^2} \right) - \sum_i \frac{(\theta_{*,i} - \mu_i)^2}{2\sigma_*^2} \quad (5.1)$$

The first part of Eq. 5.1 adds up the log-likelihood of each model and subtracts the prior likelihood of that model's parameters, based on a Gaussian prior centered around the top-level parameters $\theta_\star$, and with model-specific variance σ_m. The second part in the equation calculates the prior likelihood of the top-level parameters $\theta_\star$ according to a Gaussian prior with variance $\sigma_\star$ and mean μ, which is set to zero. In other words, the feature weights of each base model tend to get closer to the top-level parameters

and, thus, to each other through the top-level. While the variance $\sigma_\star$ controls how much the hierarchical model cares about feature weights diverging from μ_i, the model-specific variance σ_m controls the divergence of domain-specific parameters from each other. Hence the lower σ_m is, the more similar values the shared features among the base models have, and visa versa.

In order to calculate the optimum parameters of this model, we take the partial derivatives for the parameters of each base model m and for the top-level parameter θ, which are respectively:

$$\frac{\partial \mathcal{L}_{hier}(\mathcal{D};\theta)}{\partial \theta_{m,i}} = \frac{\partial \mathcal{L}_{hier}(\mathcal{D}_m,\theta_m)}{\partial \theta_{m,i}} - \frac{\theta_{m,i}-\theta_{*,i}}{\sigma_d^2} \tag{5.2}$$

$$\frac{\partial \mathcal{L}_{hier}(\mathcal{D};\theta)}{\partial \theta_{*,i}} = \left(\sum_{m\epsilon\mathcal{M}} \frac{\theta_{*,i}-\theta_{m,i}}{\sigma_m^2}\right) - \frac{\theta_{*,i}-\mu_i}{\sigma_*^2} \tag{5.3}$$

Gradient descent and L-BFGS are two well-known optimization algorithms. However, their requirement of using the entire training set to compute the objective function (both its actual value and partial derivative) is computationally not practical. Hence, stochastic gradient descent, which uses a small subset (batch) of each training set to compute the estimate of the real objective function is used. The stochastic versions of both equations above for batch size of one are given in [12] as follows, where $|\mathcal{D}_{m(d)}|$ is the number of instances $\{d\}$ selected from the model-specific training set $\mathcal{D}_m$:

$$\frac{\partial \mathcal{L}_{hier-stoch}(\mathcal{D};\theta)}{\partial \theta_{m(d),i}} = \frac{\partial \mathcal{L}_{m(d)}(\{d\};\theta_{m(d)})}{\partial \theta_{m(d),i}} - \frac{1}{|\mathcal{D}_{m(d)}|}\left(\frac{\theta_{m(d),i}-\theta_{*,i}}{\sigma_d^2}\right) \tag{5.4}$$

$$\frac{\partial \mathcal{L}_{hier-stoch}(\mathcal{D};\theta)}{\partial \theta_{*,i}} = \frac{1}{|\mathcal{D}_{m(d)}|}\left(\frac{\theta_{*,i}-\theta_{m(d),i}}{\sigma_m^2}\right) - \frac{1}{\sum_{m\epsilon\mathcal{M}}|\mathcal{D}_m|}\left(\frac{\theta_{*,i}}{\sigma_*^2}\right) \tag{5.5}$$

When the batch size is greater than one, the value of each function is calculated for

each datum and then all of these values are summed up.

5.2. Hierarchical Joint Learning of Full and n-gram Parsing

In our hierarchical joint learning setup, we jointly learn full sentence parsing and n-gram parsing in the same way as described in [31]. However, in our case, it is not required to have a joint model as in [31]. Because in [31], Finkel *et al.* have two fundamentally different models for parsing and named-entity recognition and, hence, they created a joint model of these two models, which has features from both tasks. That is how they form commonality between two disjoint tasks. In our case, n-gram and full sentence parsers use almost the same set of feature templates. Hence, they inherently have common features.

We train the models in the hierarchical learning setup and optimize the parameters with stochastic manner. At each batch, we randomly pick N instances, which may include instances from the training set of any model, and calculate updated parameter values for both base models and the top model at the end of each batch. As one model learns the feature weights from its own training set, the hierarchical prior transfers that change in the parameter value, by causing the corresponding shared feature from the other model get influenced by that change. At the end of training, we retain the parameter values of the full sentence parser and evaluate it on the Penn treebank [5], as previously done in Chapter 3.

5.3. Data

To be consistent with previous experiments, for the two different parsing models we use the same data sets from Chapters 3 and 4, that is the WSJ15 set for full sentence parsing and the WSJOver15 set for each n-gram model. While using the complete set of WSJ15 for the full parsing model, in order to examine how the size of the n-gram training set affects the result of joint learning, we use four different sets for each n-gram model with increasing numbers of training instances. We randomly pick 1000, 2000, 5000, and 10000 training instances and create three sets of each so that at the

end we can take the average of the results and get more accurate estimation on the experiments.

5.4. Experiments

In this experimental setup, we use the full sentence parsing model from Chapter 3 and all n-gram models from Chapter 4 one-by-one. Based on the hierarchical joint learning approach, for each conducted experiment, a top model is created, which consists of all features that exist in both the full and the n-gram model. Each of these three models has their own set of parameters but they affect each other through the parameter update formulas defined by the HJL approach.

As there are multiple simultaneously trained models, the number of parameter setting combinations is quite high. For the full and n-gram models, we use the same variance values from the previous corresponding chapters, that is 0.1. We also set the top model variance ${\sigma_*}^2$ to 0.1. We set the learning factor for the n-gram and the top model to 0.1, while using 0.05 for the full parsing model. With this, we make sure that the full parsing model starts to learn in slower pace than usual so that it doesn't directly get into the effect of the accompanying n-gram model. For all experiments in this chapter, we perform 20 iterations as our preliminary experiments show that this would be enough to get the best performance for the jointly trained full sentence parser. Within each iteration, we apply the stochastic gradient descent approach by updating the parameters after processing a batch of 40 training instances. The training instances are selected from the training set of all models in the experiment. As the training instances are selected uniformly, the ratio of instances from each model depends on the relative sizes of the training sets.

With these parameter settings, we train the full model with each n-gram model one by one. We use four different training sets for the n-gram model in order to evaluate the effect of the n-gram model size. We execute each experiment three times with three versions of training sets, all randomly picked. Table 5.1 shows the averaged F_1 scores obtained by the jointly trained full sentence parser on the development and

test sets of the WSJ15. The rows correspond to the accompanying n-gram model in the joint learning, whereas the columns represent the size of the training set for the n-gram model. Note that we use the same training set for the full model from Chapter 3, which contains 9753 instances.

Table 5.1. Averaged F_1 scores of full sentence parser jointly trained with each n-gram model.

	Results for Dev. Set of the WSJ15				Results for Test Set of the WSJ15			
Model(s)	1K	2K	5K	10K	1K	2K	5K	10K
B+3-gram	87.60	87.69	87.97	87.91	86.37	86.07	86.23	86.21
B+4-gram	87.93	87.98	**87.99***	87.70	86.52	**86.42*****	86.44	86.54
B+5-gram	87.72	87.67	88.00	87.72	86.36	85.84	86.35	86.33
B+6-gram	87.88	87.73	**88.12****	87.66	86.55	85.95	86.24	86.31
B+7-gram	87.83	87.94	88.05	87.72	86.58	86.16	86.24	86.42
B+8-gram	87.93	87.91	87.96	87.78	**86.57****	86.45	86.16	86.43
B+9-gram	**88.19***	87.89	87.89	87.86	86.46	86.42	86.44	**86.59*****

Scores in bold in Table 5.1 indicate that the value is significantly[2] better than the baseline value. When the results on the development set are compared with the F_1 score of 87.79 of the baseline full sentence parser from Chapter 3, we observe slight improvement at certain configurations. More specifically, the jointly trained full parser outperforms the baseline parser when it is trained alongside an n-gram parser that uses a small training set, like 5000 instances. However, observed improvements on the development and test set do not happen with the same configuration. Based on the results taken on the development set, we can conclude that the observed improvement on the jointly trained full parser dissipates as the n-gram models use larger training sets. In case of the test set, there is no such established pattern.

When evaluating and comparing the jointly trained full parser with the baseline at their peak performances, we can also examine how fast or slow the jointly trained

[2]The superscript * adjacent to the F_1 scores indicates that its significance is $p<0.01$. In case of the ** and ***, it is $p<0.005$ and $p<0.001$, respectively.

full parser reaches its peak compared to the baseline parser. Considering the fact that till iteration 10, the baseline parser achieves an F_1 score of 87.52 at most and the results in Table 5.1 are taken with 10 iterations, it can be said that the jointly trained full parser reaches its best performance faster compared to the baseline parser.

Table 5.2. Statistics on the top performing jointly trained parsers along with the baseline parser.

Model(s)	Precision	Recall	F_1 Score	Exact	Avg CB	0 CB
B+9G:1K	88.15	88.22	88.19	54.22	0.28	84.34
B+6G:5K	87.93	88.31	88.12	54.53	0.30	83.54
B+4G:5K	87.64	88.33	87.99	54.17	0.30	83.32
Baseline (B)	87.49	88.09	87.79	53.33	0.30	83.10

Table 5.2, which compares the top performing jointly trained full parsers with respect to the baseline parser, reveals another difference. The increase in the precision measure is higher than the increase in the recall measure. The interpretation of this finding is that joint learning helps the parser slightly more about preventing mistakes than teaching new things.

5.4.1. Accuracy Analysis on Constituents

In order to better analyze how the jointly trained full parser is affected by the accompanying n-gram model, we also evaluate the performance of the parser on the most common constituents. The same analysis is also done for the baseline parser in Chapter 3 and each n-gram parser in Chapter 4. Thanks to those analysis, we could compare and see more clearly what changes after the joint learning process.

Table 5.3 shows the averaged F_1 scores for the most common constituents obtained by the jointly trained full parser on the development set for each different n-gram model. We choose the ones that are trained with 5000 instances in order to make them more comparable with each other and, also, most of them are the best

performing ones.

Table 5.3. F_1 scores on the most common constituents for each n-gram model from the development set.

Model(s)	NP	VP	PP	S	SBAR	ADVP	ADJP	QP
B+3-gram	88.91	89.87	89.39	90.20	80.09	80.05	64.51	92.44
B+4-gram	88.82	90.30	89.43	90.26	81.00	79.51	61.65	92.14
B+5-gram	89.10	90.21	89.23	90.22	79.44	79.56	61.23	92.68
B+6-gram	89.19	90.15	89.30	90.26	80.55	79.75	63.18	93.07
B+7-gram	89.04	90.19	89.15	90.35	80.73	79.31	62.93	93.03
B+8-gram	88.96	90.17	88.90	90.27	80.91	79.26	61.11	92.48
B+9-gram	88.86	90.14	89.06	90.12	79.45	79.07	62.38	92.74
Baseline (B)	88.77	89.81	88.79	90.91	80.56	79.42	59.31	94.21

The results shown in Table 5.3 indicate couple of interesting reasons behind the slight performance improvement of the jointly trained full parser. The first one is the slight improvement on noun phrases (NPs) as the n-grams are getting bigger, which is especially visible with the highest performing configuration among them, that is the 6-gram model. Like NPs, prepositional (PPs) and verb (VPs) phrases are also better processed with almost all n-gram models. However, the biggest improvement is seen with the adjective phrases (ADJPs), especially with the smaller n-grams. Even though the percentage of ADJPs is low compared to the others like NPs and PPs, this improvement is worth mentioning especially given the fact that the same analysis on the stand-alone n-gram parsers reveals that they are not that good with ADJPs. Another thing to notice is the degrading performance over the QPs, which is expected given that we know how bad the n-gram parsers are with them. In case of SBAR, there is no particular pattern that tells us when they are handled better than the baseline does. Nevertheless, compared to the way that the jointly trained parser handles Ss, they are relatively better processed.

5.4.2. Length- and Height-based Accuracy Analysis of Constituents

Our another analysis technique involves looking at the performance of the parser on constituents with different length and at different heights. We perform the same analysis for the full sentence parser in Chapter 3 and the n-gram parsers in Chapter 4. Hence the analysis of the jointly trained full parser with respect to the same criteria is expected to reveal more things about how n-gram models affect the full parser. Obtained with the jointly trained full parsers, the averaged accuracies for each type of constituent in terms of length and height are given in Tables 5.4 and 5.5, respectively.

Table 5.4. Length-based accuracy on constituents of jointly trained full parser.

Model(s)	L=1	L=2-4	L=5-9	L=10-15
B+3-gram	85.57	89.34	85.27	93.26
B+4-gram	85.70	89.38	85.61	93.81
B+5-gram	85.05	89.43	85.02	92.87
B+6-gram	85.67	89.30	85.56	94.04
B+7-gram	85.32	89.41	85.45	93.51
B+8-gram	85.73	89.06	85.81	93.81
B+9-gram	85.68	89.27	85.50	93.74
Baseline (B)	85.0	89.5	85.2	93.2

The comparison of results with respect to the length of constituents reveals that when the baseline parser is trained along with the n-gram parser, it almost always performs better on all types of constituents, especially with the ones that span one word only (L=1). They all struggle with the ones that cover 2-4 words (L=2-4). We have no explanation for that, even though it is obvious that it cannot be direct effect of an n-gram model. The parser jointly trained with the 6-gram parser obtains high performance compared to the baseline in all cases , except for the one with L=2-4. Especially, its performance on the longest constituents (L=10-15) especially stands out. However, considering all these, there is no distinct pattern that indicates whether using smaller and larger n-gram trees results in better or worse results.

Table 5.5. Height-based accuracy on constituents of the jointly trained full parser.

Model(s)	H=2	H=3,4	H=5-9	H=10-15
B+3-gram	88.53	86.45	88.80	100.0
B+4-gram	88.50	86.75	89.23	100.0
B+5-gram	87.32	86.58	88.35	100.0
B+6-gram	88.46	86.94	88.99	100.0
B+7-gram	88.37	86.90	88.75	100.0
B+8-gram	88.38	86.62	89.26	100.0
B+9-gram	88.49	86.60	89.09	100.0
Baseline (B)	88.1	86.8	88.9	100.0

The second analysis is done based on at what height the constituent is located. According to the averaged accuracies given in Table 5.5, the n-gram models help the full parser do a slightly better job with the constituents that are located above the preterminals (H=2). The same slight improvement, however, is not that much observable for constituents located at higher locations in the parse tree. Nevertheless, the jointly trained parser with the 6-gram parser achieves higher performance than the baseline in all cases. The highest improvement is seen when H=2. It is also worth mentioning the high performance with the 8- and 9-gram parsers when H=5-9.

5.5. Discussion

In this chapter, we take the full sentence parser from Chapter 3 and n-gram parsers from Chapter 4 and put them into the Hierarhical Joint Learning (HJL) setup of Finkel *et al.* [31]. By doing this, we hope that n-gram models help the jointly trained full parser achieve better results than the original baseline full parser. In our analysis, we observe slight improvement over the baseline scores. We also find out that the new jointly trained full parser is better at processing noun phrases (NPs), prepositional phrases (PPs), and especially adjective phrases (ADJPs). Based on more detailed analysis over the constituents, we also observe that it performs better with constituents that span one word and located over the preterminals. Looking into the

statistics more carefully also reveals that the jointly trained full sentence parser achieves its best performance faster than the baseline parser, indicating yet another benefit of this process.

Despite the slight improvement we get with the jointly trained parser, the fact that we obtain the best results with the n-gram models that are trained on small sets of instances, like 5f000, is discouraging. As discussed in Section 4.2, even though the available size of training sets for n-gram models are considerably higher than the size of the full sentence training set, not being able to use such large training set nullifies that advantage.

As future work, there are number of things that can be done. For example, what would happen when we use multiple n-gram parsers in this setup is the first question that comes to mind. Moreover, considering that we have multiple models in this setup and many ways to set their parameters, we may try large number of parameter combinations to reach the best result that this setup can deliver.

6. SELF-TRAINING WITH N-GRAM TREES

In Chapters 3 and 4, we experiment with full sentence and n-gram parsing tasks, respectively. In Chapter 5, we train both parsers together in a supervised setting in order to boost the accuracy of the full sentence parser. In this chapter, we alter this supervised setting by replacing a portion of the used training n-gram trees with the ones extracted from the output of the full sentence parser. In other words, instead of the common approach using the complete guested trees of the full sentence parser for self-training, we use the n-gram trees extracted from those trees, which are expected to be more accurate than the complete ones based on our initial assumption. In addition using the Hierarchical Joint Learning Approach, we expect to transfer the distinct expertise of the n-gram parser to the full sentence parser. This way, while self-training the n-gram parser, we also indirectly self-train the full sentence parser, which we call a jointly self-trained parser at the end.

6.1. Self-training Pipeline

To build the self-training pipeline, we extend the setup of the previous chapter by making it a two stage process. First, we train n-gram and full models for N iterations. Then, the system stops in order to parse the given sentences to generate new training data for self-training of the n-gram parser in the setup. The number of parsed sentences depends on how many n-gram trees are required as new additions to the training set of the n-gram model. So far, other than n-gram trees, we did not consider full parse trees as new training instances, however. As new n-gram training instances arrive, the training set of the n-gram model increases, which affects the joint learning equations for the base and top model parameters given in Section 5.1.1. This time n-gram models start to have more influence on the full model as they have more data compared to the beginning. After the n-gram training sets are expanded with guessed n-grams, training continues for M more iterations. As in the previous chapter, at the end, we retain the parameter values of the full parser and evaluate it on the development and test sets of the Penn treebank.

6.2. Data

As we mirror the experiments from the previous chapter, we use the same data sets for the full sentence and n-gram parsing tasks. For the self-training purpose, we use sentences that are similar to the ones seen in the Penn treebank. Therefore, we use Reuters RCV1 corpus which contains newswire articles from Reuters about finance and economy. Previously done self-training studies used sentences from the Wall Street Journal, which is also the source of the Penn Treebank. However at the time of experiments, we didn't have the access to this data. Nevertheless, in order to make the sentences from RCV1 compatible with the format of the WSJ15 data set, we perform tokenization and sentence splitting as well as further preprocessing which discards any sentence that contains no more than one word that does not occur in the training set of the Penn treebank. We allow one word to be unknown to keep the data set still challenging for the parser. Then, we choose those sentences that have no less than three words and no more than 15 words. The minimum size of a sentence is set based on our smallest n-gram model, which is 3-gram model. Even though Reuters RCV1 corpus contains about 810000 articles, the number of sentences extracted with this process is irrelevant since our pipeline only uses a fraction of it to generate the required self-training data.

6.3. Experiments

We set up the experiments so that we can assess the contribution of the self-training data by comparing the results with the corresponding ones where all the used n-gram trees for training are real. We copy each experiment from the previous chapter but substitute a subset of the real n-gram trees with guessed ones extracted from the output of the full sentence parser. For example, in the previous chapter, one set of experiments use 2000 real n-gram trees for training the n-gram parser in the joint learning setup. Here, instead of using 2000 real n-gram trees, we start with 1000 real n-gram trees and add 1000 new ones later. The new n-gram trees are extracted from the output of the full sentence parser, which is the one trained along with the n-gram parser in joint learning setup. In case of the experiments with 5000 n-gram trees

from the previous chapter, we again start with 1000 real ones and add 4000 extracted ones later on. We keep the seed n-gram training set small, which gets even smaller as more extracted trees are used. In this way, we can evaluate the increasing size of self-training data both with respect to the results from the stand alone full sentence parser of Chapter 3, which achieves F_1 score of 87.79, and the jointly trained parsers from the previous chapter.

Table 6.1 shows results with different n-gram models and increasing size of self-training data. Note that we copy development set results from Table 5.1, where we train the n-gram parser with 2000, 5000 and 10000 n-gram trees. Other columns list F_1 scores when we use 1000 real n-gram trees and expand with 1000, 4000, and 9000 guessed n-gram trees respectively in course of the self-training process. In order to get these results, we run the first stage for five iterations, where no self-training is present in the system. We then pause and generate self-training data from the output of the full sentence parser, which is also trained for five iterations in the same setup. After generating the self-training data at once we continue training for 10 iterations, adding up to 15 iterations in total. We also run each experiment with two randomly chosen n-gram training sets and then take the averages.

Table 6.1. F_1 scores of the jointly trained full parser with self-training n-gram models.

Models	2+0K	1+1K	5+0K	1+4K	10+0K	1+9K
B + 3-gram	87.88	87.79	87.97	88.12	87.91	87.80
B + 4-gram	87.89	87.72	87.99	87.70	87.70	87.57
B + 5-gram	87.79	87.78	88.00	87.73	87.72	76.55
B + 6-gram	88.12	87.72	88.12	87.81	87.66	74.39
B + 7-gram	88.05	87.79	88.05	87.84	87.72	87.15
B + 8-gram	88.13	**87.98***	87.96	87.87	87.78	87.78
B + 9-gram	87.96	87.66	87.89	87.69	87.86	67.95

The score in bold in Table 6.1 indicates its significance[3] over the baseline score.

[3]The superscript * adjacent to the F_1 scores indicates that its significance is $p<0.01$. In case of the ** and ***, it is $p<0.005$ and $p<0.001$, respectively.

Based on the results, it is evident that when we use guessed n-gram trees instead of real ones, accuracy improvement of the jointly trained parser disappears. Compared with the results from the previous chapter, in almost all cases except one, the jointly self-trained parser performs similar or worse than not only the jointly trained parser but also the baseline. In only two cases, where we use 1000 guessed 8-gram trees and 5000 3-gram trees, it achieves better than the baseline. And, surprisingly, the second one is even better than the one when all n-gram trees are real. Even though we haven't been able to try various parameter settings for this setup, from these results, it can be concluded that when we use the full sentence parser's output to self-train the n-gram parsers, it becomes a lot harder than just doing supervised joint learning.

6.3.1. Accuracy Analysis on Constituents

In order to better understand what went wrong with this setup, we further investigate how the jointly self-trained parser performs on each type of constituent and compare the results with the performance of the baseline. The results from the configuration where the full parser is trained with an n-gram parser that expands its training set with 5000 guessed instances are given in Table 6.2. Compared to the ones obtained with the jointly trained parser in Table 5.3, there is no particular pattern of which type of n-grams achieve better or worse for particular constituents. Nevertheless, when reading between the lines, there are couple of things to notice.

The very first thing to notice is that all parsers fail with S and ADVP constituents, which is also the case with the supervised setup. However, in case of SBAR, the numbers with 5- and 6-gram parsers are not consistent with the ones given in Table 5.3, especially with the 5-gram parser which had the worst accuracies with SBAR in the supervised setting. At this point, it has to be noted that within the time frame of this thesis, we have been able to run each experiment only twice. Hence, we may need to have more runs for each experiment to get smoother results.

When we look at the results of the jointly self-trained parser that is trained with 6-gram parser, which was the best performing setup from the previous chapter, it can

Table 6.2. F_1 scores on the most common constituents for each n-gram model from the development set.

Models	NP	VP	PP	S	SBAR	ADVP	ADJP	QP
B + 3-gram	89.23	90.40	89.29	90.43	80.30	79.17	60.90	94.12
B + 4-gram	88.95	89.61	89.05	89.87	79.57	78.34	60.45	94.96
B + 5-gram	88.86	89.99	88.51	90.36	81.28	77.60	58.95	93.28
B + 6-gram	89.05	90.04	88.97	89.65	81.24	78.52	59.69	94.12
B + 7-gram	89.17	90.11	89.75	89.68	78.87	78.03	58.67	92.88
B + 8-gram	88.98	90.24	88.94	89.98	79.85	79.33	60.21	93.28
B + 9-gram	88.84	89.99	89.65	89.69	75.18	77.94	60.00	94.12
Baseline (B)	88.77	89.81	88.79	90.91	80.56	79.42	59.31	94.21

be observed that it performs better than the baseline in case of noun (NP), verb (VP), and prepositional (PP) phrases, as well as SBARs. However, its poor performance with Ss might be the main reason why it cannot get significantly better F_1 score than the baseline. Recall from Table 4.4 that number of S constituents in the development set is more than the number of PPs and close to the number of VPs. Hence, the performance on S-type constituents is decisive.

6.3.2. Length- and Height-based Accuracy Analysis of Constituents

As in the case of previous chapters, we again analyze the constituency accuracy based on their lengths and their heights in the parse tree. These two sets of results obtained with the full parser which is trained with an n-gram parser that expands its training set with 5000 guessed instances are given in Tables 6.3 and 6.4, respectively.

When we compare the length-based results of the jointly self-trained parsers with the baseline one-by-one, we observe that this parser generally has more problems in most of the cases. For example, when the parser is jointly trained with an n-gram parser in a supervised setup, it performs better than the baseline in case of the constituents that cover only one word (L=1). However, this observation doesn't hold in general for the jointly self-trained parser. For L=2-4, the results are consistent with the ones from

Table 6.3. Length-based accuracy on constituents of the jointly trained full parser.

Model(s)	L=1	L=2-4	L=5-9	L=10-15
B + 3-gram	85.46	89.48	85.26	93.06
B + 4-gram	83.95	89.06	84.78	93.42
B + 5-gram	85.12	88.83	84.97	93.66
B + 6-gram	84.91	89.38	84.58	93.65
B + 7-gram	85.07	89.15	85.00	93.65
B + 8-gram	85.15	89.51	84.45	93.18
B + 9-gram	85.54	88.91	84.88	93.18
Baseline (B)	85.1	89.5	85.2	93.2

the previous chapter. In both cases, the resulting parser is worse than the baseline at handling such constituents. Maybe the most distinguishable difference coming out of using guessed n-gram trees instead of real ones is visible in case of mid-size constituents, that is L=5-9. While the jointly trained parser from the previous chapter handles these constituents better than the baseline, here it is the opposite in almost all cases.

Table 6.4. Height-based accuracy on constituents of the jointly trained full parser.

Model(s)	H=2	H=3,4	H=5-9	H=10-15
B + 3-gram	88.49	86.58	88.77	100.0
B + 4-gram	87.84	86.30	88.06	100.0
B + 5-gram	87.84	86.55	88.51	100.0
B + 6-gram	88.10	86.68	88.38	100.0
B + 7-gram	88.10	86.68	88.49	100.0
B + 8-gram	88.31	86.63	88.15	100.0
B + 9-gram	88.03	86.82	88.04	100.0
Baseline (B)	88.1	86.8	88.9	100.0

The height-based analysis of the jointly self-trained parser on the most common constituent types, is shown in Table 6.4. If we compare these results with the ones obtained in the previous chapter, one thing stands out quickly. That is, for the con-

stituents that are just above the preterminal level (H=2), the supervised jointly trained parser of the previous chapter performs better than the self-trained one of this chapter. This is also true for H=5-9. Other than that, unlike the length-based results, where we see some improvement over the baseline for certain constituents, here the results are almost the same or worse than the ones obtained with the baseline parser.

6.4. Discussion

This chapter is the final step of the road to the self-trained discriminative constituency parser. Instead of directly feeding the parser its own output, we first introduced the concept of n-gram trees in Chapter 4 and build an n-gram parser. Then, we train our full sentence parser with the n-gram parser together in a Hierarchical Joint Learning setup described in Chapter 5. Finally, we altered the joint learning setup in this chapter by feeding the n-gram parser with the output of the full sentence parser instead of using real n-gram trees. However, the results obtained with the jointly self-trained parser are not statistically significantly better than the baseline parser.

Our analysis shows that using guessed n-gram trees still helps the full sentence parser achieve better performance than the baseline parser in certain cases, such as when parsing noun, verb, and prepositional phrases. Like the supervised jointly trained parser, in most of the cases, it handles the one-word constituents better than the baseline. However, it is not as good as before in case of longer constituents. It is possible that the quality of such constituents coming from the guessed parse trees is not good, thus it affects the full sentence parser badly. Moreover, when we analyze the constituent accuracy based on their located height in the parse tree, in almost all cases the jointly self-trained parser achieves similar or worse results than the baseline parser. This shows that when we use guessed n-gram trees, we loose he advantage of n-gram parser teaching the full sentence parser how to do better with constituents located above the preterminals. Also, in general, the guessed parse trees do not help the full sentence parser in any specific way for parsing constituents based on their height.

In general our results show no improvement over the baseline. This might be due

to the fact that we haven't been able to try all possible scenarios. Due to the joint learning setup, there are many ways of setting the parameters in order to reach the best result that this particular setup can deliver. We would also like to run experiments to see the performance of the self-trained n-gram parser on its own, that is outside this joint learning setup. Analysis of these experiments might give us more clues to make the jointly self-trained parser better.

7. CONCLUSION

In the context of this thesis, initially we built a baseline full sentence parser, which is a discriminative parser based on the Conditional Random Field approach. We follow the work of Finkel *et al.* [12] in order to design and implement the experimental setup of our baseline parser. We also give detailed analysis of results, such as how accurately our baseline parser handles different types of constituents and what percentage of labelling and identification errors are caused by which constituents. The purpose of giving such detail is to prepare a test bed for the evaluation of new parsers introduced in the upcoming chapters. Our analysis reveals that the baseline parser fails most with noun phrases. In addition it is also not very good at identifying constituents spanning only one word.

After establishing our baseline, we introduce the concept of n-gram parsing in Chapter 4, which is essentially the same task as full sentence parsing, but performs on n consecutive words rather than all the words of a sentence. We conduct the same analysis on n-gram parsing and find out that n-gram parsing is difficult task due to absent surrounding context and the incomplete constituents that are generated by the n-gram extraction process. That is why the n-gram parsers perform slightly worse than the baseline in most of our evaluations. Nevertheless, we see that the n-gram parsers are generally better than the baseline parser when it comes to processing noun and prepositional phrases.

As we build the baseline and n-gram parsers, we start training them together in Chapter 5 within the setup of the Hierarchical Joint Learning (HJL) approach introduced by Finkel *et al.* [31]. Our analysis indicates that the jointly trained full parser slightly outperforms the baseline parser with the help of the accompanying n-gram model in the joint learning process. We observe that the new jointly trained full parser becomes better at handling noun and prepositional phrases as n-grams are getting bigger. This shows that the HJL approach is successful at transferring the ability of the n-gram parsers onto the baseline full parser, since the stand-alone n-gram parsers

perform better than the baseline on such phrases.

In the last chapter, we take the HJL experimental setup of Chapter 5 and replace the n-gram training instances with guessed n-gram trees extracted from the output of the baseline full parser. While we perform supervised learning in Chapter 5, we switch to unsupervised learning and our parser starts to self-train itself with its own output. However, the full parser does not directly do self-training on its own, but the accompanying n-gram parser in the HJL setup does that. Hence, in a sense, the whole HJL setup can be considered as doing self-training within itself because every model in the setup is expected to get benefit from the new generated training data. Our results show that the resulting jointly self-trained parser performs similarly to the baseline parser.

7.1. Future Work

Within the time frame of this thesis, we implemented a discriminative constituency parser from scratch and adopted that for n-gram parsing. Then we set up the Hierarchical Joint Learning environment and then changed it to do self-training. In fact, we explored only a small portion of the parameter combinations that give the best results for each experiment type. Hence, each chapter of this thesis requires further optimization, starting with the baseline.

One area for optimization is to figure out ways to assess the quality and usefulness of each parameter and reduce the size of the parameter space. Currently our baseline parser uses more than 600000 features and their optimization takes a lot of time. Considering that this is just for the sentences that have no more than 15 words, it is quite clear that we need more scalable design for our discriminative parser. Other than designing better feature templates, we may look for more adaptive feature selection mechanisms as well as making the training phase faster by iterating less.

Related to picking the best features, we may also represent the parameters as nodes of a network and employ social network analysis techniques to figure out which

parameters "interact" with each other most. We may use the observed relations for filtering the features or for creating more representative feature definitions.

To improve n-gram parsing, we should definitely add new types of features that differentiate n-gram parsing from the conventional full sentence parsing. One possible improvement is using more contextual information in the features. We may use the deep analysis of the full sentence and n-gram parsers and design features that improve any observed deficiency. We can also make n-gram parsers more scalable so that we can use all the training set we have by using computationally reasonable time and memory. Another thing to improve is to make more detailed analysis of the results and figure out where n-gram parsing experiences the most ambiguity.

In case of the improvement of the joint learning setup, we can do deep analysis of the shared features and investigate which of them cause better or worse results. It is possible that some may degrade the performance while others are more suitable for being used as shared feature. Another thing to improve is to use more n-gram trees for better results. Currently the jointly trained parser achieves better results than the baseline when the n-gram parser uses 1000 or 5000 n-gram trees in that joint learning setup. However, this is just a fraction of the training sets of the n-gram trees. This is also the case for self-training setup. For both cases, we need to investigate what makes us use such small amount of training data and how to improve it. Apart from these new studies, we can also try using more than one n-gram parser in the joint learning setup. Different n-gram models may help the full sentence parser in different ways and improve its accuracy more.

For the last chapter in our thesis, we just scratched the surface. There are a lot of things left for future research. The first thing that comes to mind is to look at how the n-gram parsers self-train on their own as a stand-alone task and how to improve their performance. We also haven't been able to explore all possible parameter combinations to get the best performance this setup can deliver, which is again left as a future work. Apart from these, we would also like to make self-training a more continuous process rather than expanding the training set at once and doing a couple

of more iterations on top of that. It might be possible to use new training instances to jitter the parameter values to prevent over-fitting as we do more and more iterations. In other words, we would like to investigate such case where we run our parser for large number of iterations without experiencing any over-fitting issue.

In addition to all of these improvements to the existing system, we also consider other possibilities like dividing long sentences into small n-grams and then parsing them with n-gram parser(s) before combining the results to get the parse tree of the long sentence given in the first place. This can be seen as extension of the n-gram parsing concept.

So, this thesis hopefully pave the way for many research opportunities that make the self-trained discriminative constituency parser the state-of-the-art parser in the syntactic parsing literature.

REFERENCES

1. Chomsky, N., *Syntactic Structures*, Mouton de Gruyter, Berlin, 1957.

2. Collins, M., "Three Generative, Lexicalised Models for Statistical Parsing", *Proceedings of the 35th Meeting of the Association for Computational Linguistics (ACL)*, pp. 16–23, 1997.

3. Collins, M., *Head-driven Statistical Models for Natural Language Parsing*, Ph.D. Thesis, University of Pennsylvania, 1999.

4. McDonald, R., K. Crammer and F. Pereira, "Online Large-margin Training of Dependency Parsers", *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 91–98, 2005.

5. Marcus, M., B. Santorini and M. A. MarcinKiewicz, "Building a Large Annotated Corpus of English: The Penn Treebank", *Computational Linguistics*, Vol. 19, No. 2, pp. 313–330, 1993.

6. Charniak, E., "Statistical Parsing with a Context-free Grammar and Word Statistics", *Proceedings of The Fourteenth National Conference on Artificial Intelligence (AAAI)*, pp. 598–603, 1997.

7. Ratnaparkhi, A., "Learning to Parse Natural Language with Maximum Entropy Models", *Machine Learning*, Vol. 34, pp. 151–175, 1999.

8. Charniak, E., "A Maximum-Entropy-Inspired Parser", *Proceedings of the North American Association of Computational Linguistics (NAACL)*, 2000.

9. Collins, M., "Discriminative Reranking for Natural Language Parsing", *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pp. 175–182, 2000.

10. Charniak, E. and M. Johnson, "Coarse-to-fine N-best Parsing and MaxEnt Discriminative Reranking", *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL)*, pp. 173–180, 2005.

11. McClosky, D., E. Charniak and M. Johnson, "Effective Self-training for Parsing", *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*, 2006.

12. Finkel, J. R., A. Kleeman and C. D. Manning, "Efficient, Feature-based Conditional Random Field Parsing", *Proceedings of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*, 2008.

13. Kasami, T., "An Efficient Recognition and Syntax-analysis Algorithm for Context-free Languages", *Technical Report, Air Force Cambridge Research Lab*, 1965.

14. Younger, D. H., "Recognition and Parsing of Context-free Languages in Time n3", *Information and Control*, Vol. 10, No. 2, pp. 189–208, 1967.

15. Cocke, J. and J. T. Schwartz, "Programming Languages and Their Compilers: Preliminary Notes", *Technical Report, Courant Institute of Mathematical Sciences, New York University*, 1970.

16. Earley, J., "An Efficient Context-free Parsing Algorithm", *Communications of the Association for Computational Linguistics (ACL)*, Vol. 13, No. 2, pp. 94–102, 1970.

17. Black, E., F. Jelinek, J. Lafferty, D. M. Magerman, R. Mercer and S. Roukos, "Towards History-based Grammars: Using Richer Models for Probabilistic Parsing", *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL)*, 1993.

18. Jelinek, F., J. Lafferty, D. M. Magerman, R. L. Mercer, A. Ratnaparkhi and S. Roukos, "Decision Tree Parsing using a Hidden Derivation Model", *Proceedings of the Human Language Technology (HLT)*, 1994.

19. Magerman, D. M., "Statistical Decision-tree Models for Parsing", *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 1995.

20. Briscoe, T. and J. Carroll, "Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unication-based Grammars", *Computational Linguistics*, Vol. 19, No. 1, pp. 25–59, 1993.

21. Huang, L., "Forest Reranking: Discriminative Parsing with Non-local Features", *Proceedings of Ninth International Workshop on Parsing Technology*, pp. 53–64, 2005.

22. Zhang, H., M. Zhang, C. L. Tan and H. Li, "K-best Combination of Syntactic Parsers", *Proceedings of the Empirical Methods on Natural Language Processing (EMNLP)*, pp. 1552–1560, 2009.

23. Kaplan, R. M. and J. Bresnan, "Lexical-functional Grammar: A Formal System for Grammatical Representation", *In J. Bresnan, editor, The Mental Representation of Grammatical Relations*, pp. 173–281, 1982.

24. Steedman, M. J., *The Syntactic Process*, MIT Press, Cambridge, Massachusetts, 2000.

25. Schabes, Y., "Stochastic Lexicalized Tree-adjoining Grammars", *Proceedings of the International Conference on Computational Linguistics (ICCL)*, pp. 426–432, 1992.

26. Johnson, M., "PCFG Models of Linguistic Tree Representations", *Computational Linguistics*, Vol. 24, 1998.

27. Klein, D. and C. D. Manning, "Accurate Unlexicalized Parsing", *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics (ACL)*, Vol. 1, pp. 423–430, 2003.

28. Bod, R., R. Scha and K. Sima'an, "Data Oriented Parsing", *CSLI Publications, Stanford University*, 2003.

29. Joshi, A., L. Levy and M. Takahashi, "Tree Adjunct Grammars", *Journal of the Computer and System Sciences*, Vol. 10, No. 1, pp. 136–163, 1975.

30. III, H. D. and D. Marcu, "Domain Adaptation for Statistical Classifiers", *Journal of Artificial Intelligence Research*, 2006.

31. Finkel, J. R. and C. D. Manning, "Hierarchical Joint Learning: Improving Joint Parsing and Named Entity Recognition with Non-Jointly Labeled Data", *Proceedings of the Association for Computational Linguistics (ACL)*, 2010.

32. Klein, D. and C. D. Manning, "Corpus-based Induction of Syntactic Structure: Models of Dependency and Constituency", *Proceedings of the 42th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

33. Roark, B. and M. Bacchiani, "Supervised and Unsupervised PCFG Adapation to Novel Domains", *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 205–212, 2003.

34. Steedman, M., S. Baker, J. Crim, S. Clark, J. Hockenmaier, R. Hwa, M. Osborne, P. Ruhlen and A. Sarkar, "CLSP WS-02 Final Report: Semi-Supervised Training for Statistical Parsing.", *Technical Report, Johns Hopkins University*, 2003.

35. Reichart, R. and A. Rappoport, "Self-training for Enhancement and Domain Adaptation of Statistical Parsers Trained on Small Datasets", *Proceedings of the 45th Annual Meeting of the Associations of Computational Linguistics (ACL)*, pp. 616–623, 2007.

36. Rose, T., M. Stevenson and M. Whitehead, "The Reuters Corpus Volume 1 - from Yesterday's News to Tomorrow's Language Resources", *Proceedings of the 3rd*

International Conference on Language Resources and Evaluation (LREC), 2002.

37. Lafferty, J., A. McCallum and F. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data", *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pp. 282–289, 2001.

38. Clark, A., "Combining Distributional and Morphological Information for Part-of-Speech Induction", *Proceedings of the tenth Annual Meeting of the European Association for Computational Linguistics (EACL)*, Vol. 1, pp. 59–66, 2003.

39. Abney, S., "Part-of-Speech Tagging and Partial Parsing", *Corpus-Based Methods in Language and Speech Processing, Kluwer Academic Publishers*, 1999.

40. Molina, A., F. Pla, L. Moreno and N. Prieto, "APOLN: A Partial Parser of Unrestricted Text", *Proceedings of the Spanish Symposium on Pattern Recognition and Image Analysis (SNRFAI)*, 1999.